



# **Admas University**

**Faculty of Informatics**

**Department of Computer Science**

## **Module for Comprehensive Core Competency in Computer Science I**



**Prepared by**

Dagne Minda (MSc)

Mekonen Mulugeta (MSc)

Teklay Hagos (MSc)

**Editor:** Mezgebu Aynabeba (MSc)

**November 2022**

**The module contains five parts**

**Part I: Computer Programming**

- computer programming
- Object-oriented programming
- Advanced programming

**Part II: Data Structure and Algorithms, Theory of Algorithms**

**Part III: Database Systems**

- Fundamentals of Database Systems
- Advanced Database Systems

**Part IV: Software Engineering**

- Fundamentals of software Engineering
- Object-oriented software engineering
- Software project Management

**Part V: Internet Programming**

- Internet Programming, I
- Internet Programming II



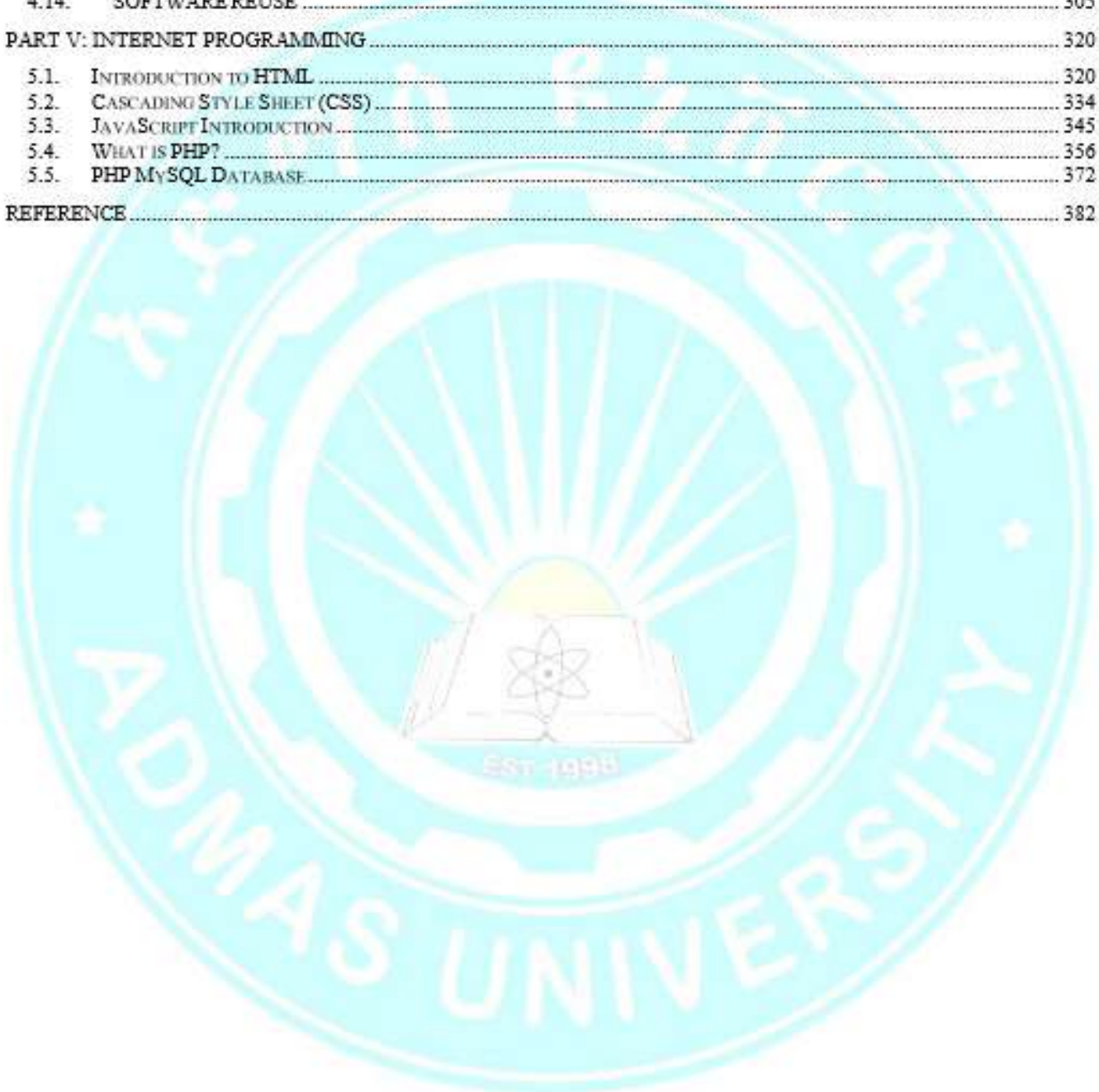


## Table of Contents

<b>PART I: COMPUTER PROGRAMMING</b>	<b>4</b>
1.1. INTRODUCTION TO COMPUTER PROGRAMMING	4
1.2. HISTORY AND ORIGIN OF C++	8
1.3. CONTROL STRUCTURE/CONTROL STATEMENTS	19
1.4. ARRAYS	27
1.5. POINTERS	30
1.6. FUNCTIONS	30
1.7. C++ FILES	42
1.8. OBJECT ORIENTED PROGRAMMING	43
<b>PART II: DATA STRUCTURE AND ALGORITHMS, THEORY OF ALGORITHMS</b>	<b>88</b>
2.1. DATA STRUCTURES AND ALGORITHMS ANALYSIS	88
2.1.1. Introduction to Data Structures and Algorithms Analysis	88
2.1.2. Simple Sorting and Searching Algorithms	90
2.1.3. Accessing Members of Structure Variables	93
2.1.4. Singly Linked Lists	93
2.1.5. Doubly Linked Lists	107
2.1.6. Array Implementation of Stacks: The PUSH operation	110
2.1.7. Operations on Binary Search Tree	115
2.2. DESIGN AND ANALYSIS OF ALGORITHM	125
2.2.1. Divide and Conquer Approach	125
2.2.2. Greedy Technique	126
2.2.3. Dynamic Programming	127
2.2.4. Backtracking Algorithm	127
2.2.5. Kruskal's Algorithm	128
<b>PART III: DATABASE SYSTEMS</b>	<b>133</b>
3.1. INTRODUCTION	133
3.1.1. Data, Information and Information System	133
3.1.2. Levels of Data	134
3.1.3. What is a Database System?	135
3.1.4. Components of a Database System	137
3.1.5. The Database Management System	140
3.1.6. Database Systems Architecture	140
3.1.7. Database Models	143
3.1.8. The File Management System (FMS)	143
3.1.9. The Hierarchical Database System (HDS)	144
3.1.10. The Network Database System	144
3.1.11. The Relational Model (RM)	144
3.1.12. The Relational Database Systems	145
3.1.13. Relational Data Integrity	148
3.1.14. The Structured Query Language (SQL)	151
3.1.15. Database Design Fundamentals	165
3.1.16. Distributed Database and Client/Server Systems	190
3.2. ADVANCED DATABASE SYSTEMS	192
3.2.1. Concepts for Object-Oriented Databases	192
3.2.2. Object Oriented Concepts	193
3.2.3. Query Processing and Optimization	198
3.2.4. Transaction Processing Concepts	204
3.2.5. Concurrency Control Techniques	211
<b>PART IV: SOFTWARE ENGINEERING</b>	<b>224</b>
4.1. INTRODUCTION TO SOFTWARE ENGINEERING	224
4.2. SOFTWARE DEVELOPMENT LIFE CYCLE	226
4.3. REQUIREMENTS ANALYSIS AND SPECIFICATION	236
4.4. DECISION TREE	241



4.5.	SOFTWARE DESIGN.....	243
4.6.	SOFTWARE DESIGN STRATEGIES.....	246
4.7.	SOFTWARE ANALYSIS & DESIGN TOOLS.....	249
4.8.	CODING.....	272
4.9.	TESTING.....	277
4.10.	SOFTWARE MAINTENANCE.....	283
4.11.	SOFTWARE QUALITY.....	288
4.12.	SOFTWARE PROJECT PLANNING.....	290
4.13.	RISK MANAGEMENT.....	299
4.14.	SOFTWARE REUSE.....	305
PART V: INTERNET PROGRAMMING.....		320
5.1.	INTRODUCTION TO HTML.....	320
5.2.	CASCADING STYLE SHEET (CSS).....	334
5.3.	JAVASCRIPT INTRODUCTION.....	345
5.4.	WHAT IS PHP?.....	356
5.5.	PHP MySQL DATABASE.....	372
REFERENCE.....		382





## Executive summary

This module introduces students to fundamental and advanced concepts in computer science that leads students in preparing the national exit examination in relevant areas of computer programming, introduces the skills required to develop a computer program to solve a given problem ,database systems to learn the fundamentals of how to design the structure of data within a relational database, data structure and algorithms explores the various algorithms and data structures used to solve the most common computational problems,, theory of algorithms, software engineering examine the steps and processes involved in software design and development. Students learn engineering principles and approaches, including analyzing the requirements, exploring alternatives, and evaluating the final product and internet programming provides a thorough grounding in the rapidly evolving area of web technologies. With equal focus on user interface design on the 'client-side' or 'front-end' and on security and persistence in 'server-side' or 'back-end' scripting.



## Module Objective

On completion of the module successfully, students will be able to:

- Grasp the main idea and concepts of programming,
- Learn the software development process and tools such as editors and compilers,
- Explain the syntax and semantics of programming languages,
- Demonstrate the major building blocks of computer programs,
- Determine the basic skills of programming & solve problems using computers,
- Identify the major programming techniques and implement in using the C++ programming language, and
- Demonstrate the basic concepts and tools in C++.

### 1.1. Introduction to Computer programming

#### 1.1.1. What is a computer?

A computer is an electronic device, operating under the control of instructions stored in its own memory that can accept data (input), process the data according to specified rules, produce information (output), and store the information for future use. Any kind of computers consists of hardware and software. A computer stores any data in the form of 0's and 1's.

#### 1.1.2. What is Programming?

**Programming** is a skill that can be acquired by a computer professional that gives him/her the knowledge of making the computer perform the required operation or task. Early we try to see what a computer is? The next question is: How can computer perform a particular task?

**Programming Skill(s):** It is a skill that can be acquired by a computer professional that gives him/her the knowledge of making the computer perform the required operation or task. By itself, a computer will not do anything useful; there must be a set of instructions called a program that directs the computer to perform some specific tasks, which is developed by these professionals.

- A **programmer** is a person who creates programs that solve a problem using the computer
- A **program** is a sequence of steps that a computer understands and executes.
- A **programming language** is an environment/notation used to write instructions into a computer.
- A **bug** is an error in a program.
- **Debugging** is the process of removing errors, testing and revising a program to make sure that it performs as expected.
- **Syntax** is rules. In every programming Language there are sets of rules that govern how the instruction is written in a programming language



- **Semantic rules**, which describe the meaning, associated with the syntax, symbol etc.
- In order to solve/ perform a given problem, computers must be given the correct instruction about how they can solve it.

### 1.1.3. Problem Solving Technique

A problem is an *undesirable situation* that prevents the organization from fully achieving its purpose, goals and objectives or problem can also be defined as the *gap between the existing and the desired situation* where problem solving will try to fill this gap. Problem solving is the process of transforming the description of a problem into the solution. by using our knowledge of the problem domain and by relying on our ability to select and use appropriate problem-solving strategies, techniques, and tools.

### 1.1.4. Algorithm

An algorithm is defined as a step-by-step sequence of instructions that must terminate and describe how the data is to be processed to produce the desired outputs. A finite set of steps that specify a sequence of operations to be carried out in order to solve a specific problem. An algorithm is a sequence of unambiguous instructions for solving a problem.

#### 1.1.4.1. Characteristics of an algorithm

A good algorithm should have the following characteristics

- **Input:** there should be at least one input
- **Output:** there should be at least one output
- **Well-ordered:** the steps are in a clear order
- **Unambiguous:** the operations described are understood by a computing agent without further simplification
- **Effectively computable:** the computing agent can actually carry out the operation
- **Finiteness/Termination:** an algorithm must terminate after finite number of steps
- **Correctness:** Everything in the algorithm should be correct

#### 1.1.4.2. Tools used to develop algorithm

There are two commonly used tools to develop algorithm

##### a. Flowcharts

A Flowchart is a graphical representation of an algorithm or a process flowchart is used for representing algorithm in pictorial form. Flowcharts work well for **small problems**

##### b. Pseudo-code

Pseudo-code is useful for describing algorithms in a structured way. It makes use of simple English-like statements. Pseudo-code is used for larger problems.

### 1.1.5. Program

A program is a set of instructions that is written in the language of the computer. A program is used to



make a computer perform a specific task. A computer program is one concrete implementation of an algorithm using a particular computer language, an implementation of an algorithm in some programming language, like C++, Java, C#, VB, Python, Perl

#### **Desirable characteristics of a program**

- i) Correct: - A program should do what it is supposed to do
- ii) Clear: - The program logic should be easily understood
- iii) Modular: - It should be broken down in sub-routines that interact in clear ways
- iv) It should be easy to modify

#### **1.1.6. Types of programming Language**

There are three types of programming language:

- Machine language (Low-level language)
- Assembly language (Low-level language)
- High-level language

##### **1.1.6.1. Machine language:** Low-level languages- is machine dependent

Any computer can directly understand only its own **machine language**. Machine language is the "natural language/mother tongue" of a computer and as such is defined by its **hardware design**. [Note: Machine language is often referred to as **object code**.] Machine languages generally consist of strings of numbers (ultimately reduced to 1's and 0's) that instruct computers to perform their most elementary operations one at a time. Machine languages are machine

##### **1.1.6.2. Assembly language:** (Still Low-level languages). Meaning it is machine dependent

Machine-language programming was simply too slow, tedious and error-prone for most programmers. Instead of using the strings of numbers (1's and 0's) programmers began to use English-like abbreviations to represent elementary operations. These abbreviations formed the basis of assembly **languages**. **Translator programs** called **assemblers** were/ is developed to convert assembly-language programs to machine language.

##### **1.6.3 High-level languages- machine independent language levels**

Computer usage increased rapidly with the advent of assembly languages, but programmers still had to use many instructions to accomplish even the simplest tasks. To speed the programming process, **high-level languages** were developed in which single statements could be written to accomplish substantial tasks. Translator programs called **compilers** convert high-level language programs into machine language.



C, C++, Microsoft's .NET languages (e.g., Visual Basic .NET, Visual C++ .NET and C#) and Java are among the most widely used high-level programming languages.

### 1.1.7. Language translation Programs

There are three language translators:

- a. **Assembler:** is a language translators required to translate the assembly languages into machine code.
- b. **Compiler:** it is a language translator that translates the entire program of high-level language (known as a source program) in to a machine language, before the computer executes the program. Example: C++ compiler
- c. **Interpreter:** This type of language translators converts high level programming languages into machine language line-by-line. They translate a statement in a program and execute the statement immediately, i.e. before translating the next source language statement.

### 1.1.8. Major Programming Paradigms

The major land marks in the programming world are the different kinds of features or properties observed in the development of programming languages. Among these the following are worth mentioning:

*Procedural, Structured and Object-Oriented Programming Paradigms.*

#### 1.1.8.1. Procedural Programming.

**Procedural programming** is a programming paradigm based upon the concept of **procedure call**. Procedural programming is often a better choice than simple sequential programming in many situations which involve moderate complexity or which require significant ease of maintainability. Possible benefits: the ability to re-use the same code (function or procedure) at different places, an easier way to keep track of program flow than a collection of "GO TO" or "JUMP" statements.

#### 1.1.8.2. Structured Programming.

Process of writing a program in small, independent parts. This makes it easier to control a program's development and to design and test its individual component parts. Structured programs are built up from units called modules, which normally correspond to single procedures or functions. It can be seen as a subset or sub discipline of procedural programming. It is most famous for removing or reducing reliance on the GO TO statement.

#### 1.1.8.3. Object-Oriented Programming.

The idea behind OOP is that, a computer program is composed of a collection of individual units, or objects as opposed to traditional view in which a program is a list of instructions to the computer. Object-oriented programming is claimed to give more flexibility, easing changes to programs. The OOP approach is often simpler to develop and maintain. Examples of object-oriented oriented (programming languages are C++, Java, C#, Python



### 1.1.9. Generation of programming Languages

**First generation** -Machine Language (uses 0 and 1)

**Second generation** -Assembly Language (uses symbols like ADD, SUB, MUL.)

**Third generation Programming Languages (3GL)**

A grouping of programming language that introduce significant enhancements to second generation languages, primarily intended to make the programming language more programmer – friendly. English words are used to denote variables, programming structures and commands, commonly known 3GLs are FORTRAN, BASIC, Pascal, Java and the C family (C, C+, C++, C#, Objective C) of languages. Also known as high level languages.

**Fourth generation (4GL) Example SQL**

### 1.2. History and origin of C++

What is the first major programming Language? Answer: FORTRAN. Who is the world's first programmer? Answer: Ada Lovelace++ is evolved from C. C is evolved from two previous programming languages, BCPL and B by Dennis Ritchie at Bell Laboratories. BCPL (Basic Combined Programming Language) was developed in 1967 by Martin Richards. C++ began as an extended of C. It was created by **Bjarne Stroustrup** in 1979 at Bell Laboratories Murray Hill, New Jersey. **Bjarne Stroustrup** initially called the new language "C with Classes", In 1983 the name was changed to C++.

#### 1.2.1. Features of C++

- C++ is a high-level programming Language
  - You can use C++ to develop high level applications and low-level libraries very closed to the hardware
- C++ is an object-oriented programming language
- C++ is a block structured programming language

#### Why should study C++?

- The invention of C++ was necessitated by one major programming factor-increasing complexity
- For better understanding of OOPs
- C++ is efficient
- C++ is used in
  - operating systems, device drivers, web servers, cloud-based applications, search engines etc.
- C++ is often the language of choice for creating other programming languages

#### 1.2.2. Typical C++ development environment



C++ programs typically go through six phases:

Phase 1. creating a program(edit)

Phase 2. preprocess

Phase 3. compile

Phase 4. link

Phase 5. load

Phase 6. execute.

### 1.2.3. C++ Basic concepts

C++ is object-oriented general purpose programming language. It is **case sensitive** and the file extension of C++ is **.cpp**. To write and run C++ programs, you need to have a text editor and a C++ compiler installed on your computer. A text editor is a software system that allows you to create and edit text files on your computer. Programmers use text editors to write programs in a programming language such as C++.

Example of C++ text editors

➤ *Code blocks, Dev, Falcon, Turbo C++, Borland C++, Quincy*

**// my first program in C++**

```
#include <iostream>
using namespace std;
int main ()
{
    cout << "Hello World!";
    return 0;
}
```

**Output:**

**Hello World!**

The program shows the source code for our first program. Then output is given once the program is compiled and executed. The way to edit and compile a program depends on the compiler you are using. Depending on whether it has a Development Interface or not and on its version. Consult the compilers section and the manual or help included with your compiler if you have doubts on how to compile a C++ console program.

**Description about the above program is given below:**

**// my first program in C++**

This is a comment line. All lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program. The programmer can use them to include short

explanations or observations within the source code itself. In this case, the line is a brief description of what our program is.

**#include <iostream>**

Lines beginning with a pound sign (#) is a directive for the preprocessor. They are not regular code lines with expressions but indications for the compiler's preprocessor. In this case the directive `#include <iostream>` tells the preprocessor to include the `iostream` standard file. This specific file (`iostream`) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later in the program.

**using namespace std;**

All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name `std`. So in order to access its functionality we declare with this expression that we will be using these entities. This line is very frequent in C++ programs that use the standard library, and in fact it will be included in most of the source codes.

**int main ( )**

This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C++ programs start their execution, independently of its location within the source code. Every program has only one main function.

**cout << "Hello World";**

This line is a C++ statement. A statement is a simple or compound expression that can actually produce some effect. In fact, this statement performs the only action that generates a visible effect in our first program.

`cout` represents the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters (in this case the `Hello World` sequence of characters) into the standard output stream (which usually is the screen).

**<<**

The symbol `<<` is called the output operator in C++. (It is also called the put operator or the stream insertion operator.) It inserts values into the output stream that is named on its left.

**return 0;**

The return statement causes the main function to finish. `return` may be followed by a return code (in our example is followed by the return code 0). A return code of 0 for the main function is generally interpreted as the program worked as expected without any errors during its execution.

**// my second C++ Program**

**#include <iostream>**

**using namespace std;**



```
int main ()
{
cout << "Hello World! "<<endl;
cout << "I'm a C++ program "<<endl;
return 0;
}
```

#### Output:

Hello World!

I'm a C++ program

#### 1.2.4. Comments

Comments are parts of the source code disregarded by the compiler. They simply do nothing. Their purpose is only to allow the programmer to insert notes or descriptions embedded within the source code. C++ supports two ways to insert comments:

**// line comment**

**/\* block comment \*/**

The first of them, known as **line comment**, discards everything from where the pair of slash signs (//) is found up to the end of that same line. The second one, known as **block comment**, discards everything between the /\* characters and the first appearance of the \*/ characters, with the possibility of including more than one line. We are going to add comments to our second program:

#### 1.2.5. Types of errors in C++ programming

You may encounter three types of errors in C++ programming:

1. Syntax errors
2. Run time errors
3. Logic errors

#### Syntax error

An error in the format of a statement in a computer program that violates the rules of the programming language employed. Also known as syntactic error. If a syntax error is encountered during **compilation** it must correct if the **source code** is to be successfully compiled.

An attribute that often separates commercial quality compilers from academic projects is the extent to which an attempt is made to automatically correct the error and continue processing the source code.

Syntax errors may also occur when an invalid equation is entered into a **calculator**. This can be caused by opening brackets without closing them, or less commonly, using several **decimal points** in one number.

#### Runtime errors

- It is after the programs has successfully compiled and is running.
- Types of runtime errors:
  - ✓ Abnormal program termination. (no enough memory to execution)
  - ✓ Divide error (dividing an integer by zero)
  - ✓ Floating point errors
    - Divide by zero(1.0/0.0)
    - Domain (not a number 0.0/0.0)
    - Over flow (infinity assigning a high value)

### Logical errors

An error in programming that is caused by faulty reasoning, resulting in the program's functioning incorrectly if the instructions containing the error are encountered. In **computer programming**, a **logic error** is a **bug** in a program that causes it to operate incorrectly, but not to fail. Because a logic error will not cause the program to stop working, it can produce incorrect data that may not be immediately recognizable.

#### 1.2.6. Variables and Data Types

A variable can be defined as a portion of memory to store a determined value. A variable is a symbol that represents a storage location in the computer's memory. Each variable needs an identifier that distinguishes it from the others

**Variable = expression**

**Variable names: "Identifiers"**

A valid identifier is a sequence of one or more letters, digits or underline symbols ( \_ ). The length of an identifier is not limited, although for some compilers only the 32 first characters of an identifier are significant (the rest are not considered).

- *Ex: sum\_of\_squares, box\_22A, GetData, count---valid*
- *Get Data cannot be an identifier as blanks are not allowed in identifiers*
- *int.....cannot be an identifier as it is a reserved word in C++*

Neither spaces nor marked letters can be part of an identifier. A valid identifier should begin

1. with letter or underscore
2. Keywords cannot be used as identifiers

Which of the following is a valid identifier?

```
2ab //invalid identifier, because it begins with digit
_34 // valid identifier
Ab2 // valid identifier
If // invalid identifier, because if is C++ keyword
```



```
Switch // invalid identifier, because switch is C++ keyword  
_hi // valid identifier
```

### 1.2.7. Keywords

Keywords are words whose meaning are defined to the compiler

#### Standard Keywords

asm, auto, **bool**, **break**, **case**, catch, **char**, **class**, **const**, **const\_cast**, **continue**, default, delete, **do**, **double**, **dynamic\_cast**, **else**, enum, explicit, extern, false, **float**, **for**, friend, **goto**, **if**, **inline**, **int**, long, mutable, **namespace**, new, operator, **private**, **protected**, **public**, register, reinterpret\_cast, return, short, signed, sizeof, **static**, **static\_cast**, **struct**, **switch**, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar\_t, **while**

#### Very important:

The C++ language is "**case sensitive**", that means that the same identifier written in capital letters is not equivalent to another one with the same name but written in small letters. Thus, for example the variable **RESULT** is not the same one that the variable **result** nor variable **Result**.

### 1.2.8. Data Types

When programming, we store the variables in our computer's memory, but the computer must know what we want to store in them since it is not going to occupy the same space in memory to store a simple number, a letter or a large number. Our computer's memory is organized in bytes. A **byte** is the minimum amount of memory which we can manage. A byte can store a relatively small amount of data, usually an integer between 0 and 255 or one single character. But in addition, the computer can manipulate more complex data types that come from grouping several *bytes*, like long numbers or numbers with decimals.

Standard C++ has 14 different **fundamental types**: 11 **integral types** and 3 **floating-point types**. These are outlined in the diagram shown above. The integral types include the boolean type **bool**, enumeration types defined with the enum keyword, **three-character types**, and **six explicit integer types**. The three floating-point types are **float**, **double**, and **long double**. The most frequently used fundamental types are **bool**, **char**, **int**, and **double**.

#### 1.2.8.1. Declaration of variables

In order to use a variable in C++, we must first declare it specifying which of the data types above we want it to be. The syntax to declare a new variable is to write the data type specifier that we want (like **int**, **short**, **float**...) followed by a valid variable identifier.

**Example:** the following are variable declarations

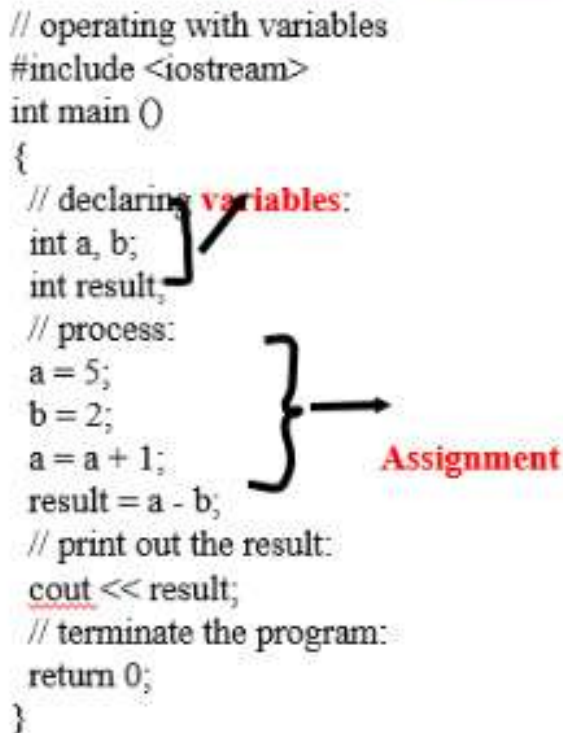
```
int a; // variable a takes integer value  
float mynumber; // the variable mynumber takes float value
```

```
int a, b, c; // the variables a, b, and c take integer value
unsigned short NumberOfSons; // the variable takes positive integer value
signed int MyAccountBalance; // the variable takes negative integer value
```

All the variables that we are going to use must have been previously declared. An important difference between the C and C++ languages, is that in C++ we can declare variables anywhere in the source code, even between two executable sentences, and not only at the beginning of a block of instructions, like happens in C.

```
// operating with variables
#include <iostream>
int main ()
{
    // declaring variables:
    int a, b;
    int result;
    // process:
    a = 5;
    b = 2;
    a = a + 1;
    result = a - b;
    // print out the result:
    cout << result;
    // terminate the program:
    return 0;
}
```

**Assignment**



#### 1.2.8.2. Constants

Constant is any expression that has a fixed value. They can be divided in to: -

1. Integer constant
2. Floating-Point constant
3. Character
4. Strings constant

##### Integer constant

- The simplest way to write a constant is to write the integer number.  
E.g 23,45,101,55
- When we write an integer constant, the compiler assigns it a c++ data types. (Default int)
- C++ programmers append l or L for long integer type.  
E.g. 23L,45L,101L,55L



By default, an integer variable is assumed to be signed. (Positive as well as Negative values)

Unsigned variables only takes positive values.

E.g. unsigned short age = 20;

unsigned int salary = 65000;

unsigned long price = 4500000;

Also, a Constant can be specified to be unsigned using the suffix U or u.

For example: 1984L, 1984l, 1984U, 1984u, 1984LU, 1984ul

### Float Constant

- For floating numbers constant, the types is always double.
- Using suffix f, F, l, L we can specify it as double or Long.  
(E.g. 23.4f, 0.21L, 45.3F, 7.46L)
- It is possible to use standard scientific notation representation. (mantissa X 10<sup>exponent</sup>)  
E.g. 1.23 X 10<sup>-4</sup> (1.23 E -4) 1.23 X 10<sup>10</sup> (1.23 e 10)

### Character constant

Example

'A', 'b'

Character constant is Witten in single quotes

### String constant

A string constant is a sequence of zero or more characters enclosed in double quotes.

E.g.. "Hello world"; "" String constant with zero character is called Null string.

Sometimes we will need the representation of a single character.

E.g. 'a', '\', '+', '\*'

### 1.2.9. Operators

- ♦ Most program perform arithmetic calculation.
  - E.g Integer1 + Integer2
  - Contain the binary operation (+) and two operands.

#### 1.2.9.1.1. Arithmetic operators:

Addition (+)	e.g., F+g
Subtraction (-)	e.g., F-g
Multiplication (*)	e.g., f*g
Division (/)	e.g., f/g
Modulus (%)	e.g., f % g (integers only)

Except for remainder (%) all other arithmetic operators can accept a mix of integer and real operands. If both operands are integers, then the result will be an integer. However, if one or both of the operands are real then the result will be a real.

E.g., 2 + 2 (result is int)

2.0 + 2 (result is float)

9 / 2 // gives 4, not 4.5!

9 / 2.0 // gives 4.5, not 4!

It is illegal to divide a number by zero. This results in a run-time *division-by zero* failure which typically

causes the

program to

terminate.

**Example:** write a c++ program that performs arithmetic operation of two numbers and display their result

**Solution**

```
#include <iostream>
using namespace std;
int main()
{
    int num1 = 25;
    int num2 = 20;
    cout << num1 << "+" << num2 << "=" << num1 + num2 << endl;
    cout << num1 << "-" << num2 << "=" << num1 - num2 << endl;
    cout << num1 << "*" << num2 << "=" << num1 * num2 << endl;
    cout << num1 << "/" << num2 << "=" << num1 / num2 << endl;
}
```

**Out put**

25 + 20 = 45

25 - 20 = 5

25\*20 = 500

25/20 = 1

### 1.2.10. Increment (++) and decrement (--)

Shortening even more some expressions, the increase operator (++) and the decrease operator (--) increase or reduce by one the value stored in a variable. They are equivalent to +=1 and to -=1,

- Prefix (++a or --a)
- Postfix (a++ or a--)



## Examples

// increment

```
#include <iostream>
using namespace std;
int main()
{
    int a = 10;
    cout<<a++;
    cout<<a;
    return 0;
}
```

Out put

10

11

// decrement

```
#include <iostream>
using namespace std;
int main()
{
    int a = 10;
    cout<<a--;
    cou<<a;
    return 0;
}
```

Out put

10

9

### 1.2.11. Compound assignment

Compound assignment (+=, -=, \*=, /=, %=, >=, <=, &=, ^=, |=)

When we want to modify the value of a variable by performing an operation on the value currently stored in that variable, we can use compound assignment operators:

**Example 1.**

Expression	is equivalent to
value += increase;	value = value + increase;
a -= 5;	a = a - 5;
a /= b;	a = a / b;
price *= units + 1;	price = price * (units + 1);

## Example 2.

```
// compound assignment operators
#include<iostream>
using namespace std;
int main ()
{ int a, b=3;
  a = b;
  a+=2; // equivalent to a = a+2
  cout << a;
  return 0;
}
```

**Output : 5**

### 1.2.11.1. Relational operators

In order to evaluate a comparison between two expressions we can use the relational and equality operators. The result of a relational operation is a Boolean value that can only be true or false, according to its Boolean result. We may want to compare two expressions, for example, to know if they are equal or if one is greater than the other is. Here is a list of the relational and equality operators that can be used in C++:

Symbol	Expression
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Fig: Relational Operators



## Example

### Relational operators ( ==, !=, >, <, >=, <= )

(7 == 5) would return false.

(5 > 4) would return true.

(3 != 2) would return true.

(6 >= 6) would return true.

(5 < 5) would return false.

### Suppose that a=2, b=3 and c=6,

(a == 5) would return false.

(a\*b >= c) would return true since (2\*3 >= 6) is it.

(b+4 > a\*c) would return false since (3+4 > 2\*6) is it.

((b=2) == a) would return true

### Logical operators ( !, &&, || )

- The Operator! is the C++ operator to perform the Boolean operation NOT, it has only one operand, located at its right, and the only thing that it does is to inverse the value of it, producing false if its operand is true and true if its operand is false. Basically, it returns the opposite Boolean value of evaluating its operand.

#### Example 1:

!(5 == 5) // evaluates to false because the expression at its right (5 == 5) is true.

!(6 <= 4) // evaluates to true because (6 <= 4) would be false. !true // evaluates to false !false // evaluates to true

#### Example 2:

((5 == 5) && (3 > 6)) // evaluates to false (true && false).

((5 == 5) || (3 > 6)) // evaluates to true ( true || false ).

## 1.3. Control Structure/Control Statements

A running program spends all of its time executing instructions or statements in that program. The order in which statements in a program are executed is called flow of that program. Programmers can control which instruction to be executed in a program, which is called **flow control**.

Normally, statement in a program execute one after the other in the order in which they are written. This is called Sequential execution. Transfer of control enables us to specify that the next statement to execute may be other than next one in the sequence.

*C++ provides three types of selection structures.*

- If (single selection structure)
- If/else (double selection structure)
- Switch (multiple selection structure)

C++ provides three types of repetition structures (also called looping structures).

- While
- Do/while
- for

C++ has only seven control structures: sequence, three types of selection (if, if/else , switch) and three types of repetition (while , do/while, For )

### 1.3.1. The If statements

The if statement allows conditional execution.

Its syntax is

*if (condition) statement*

*Example*

```
#include<iostream>
using namespace std;
int main()
{ int age = 16;
  if (age<=17)
    Cout<<"you cannot vote so go home";
}
```

#### 1.3.1.1. The if.... else statement

- The if. ...else statement causes one of two alternative statements to execute depending upon whether the condition is true.
- Its syntax is  
*if (condition) statement1*  
*else statement2*

**Example 1**

```
int main()
{ int n, d;
  cout << "Enter two positive integers: ";
  cin>>n>>d;
  if (n%d) cout << n << " is not divisible by " << d << endl;
```



```
else cout << n << " is divisible by " << d << endl;
}
```

### **Example 2.**

```
#include<iostream>
using namespace std;
int main ()
{int age = 16;
if (<=17) {
cout<< "you cannot vote so go home" << endl;
}
else {cout<< "yes yes yes you vote" <<endl;
}
}
```

### **Example 3**

```
int main ()
{int m, n;
cout << "Enter two integers: ";
cin >> m >> n;
if (m < n)
cout << m << " is the minimum." << endl;
else
cout << n << " is the minimum." << endl;
}
```

#### **1.3.1.2. Multiple selection**

##### **Example 1**

This program converts a test score into its equivalent letter grade:

```
int main ()
{int score;
cout << "Enter your test score: "; cin >> score;
if (score > 100) cout << "Error: that score is out of range.";
else if (score >= 90) cout << "Your grade is an A." << endl;
else if (score >= 80) cout << "Your grade is a B." << endl;
```

```

else if (score >= 70) cout << "Your grade is a C." << endl;
else if (score >= 60) cout << "Your grade is a D." << endl;
else if (score >= 0) cout << "Your grade is an F." << endl;
else cout << "Error: that score is out of range.";
}

```

**Example 2:** To find the commission for a sales agent based on his sales.

If the sales is 1000 or less no commission.

Commission for 1000<sales<=2000 is 5%

Commission for 2000<sales<=5000 is 8%

Commission for sales above ETB 5000 is 10%

Use 'if else' ladder (nested if else statements)

```

#include<iostream>
using namespace std;
int main ()
{
float sales, Om;
closer();
cout<<"Enter the sales value \n";
cin>>sales;
if(sales <=1000)
    com=0;
else if(sales<=2000)
    com=sales*0.05;
else if (sales<=5000)
    com=sales*0.08;
else
    com=sales*0.10;
cout<<"\nCommission is ETB " <<com;
}

```

### 1.3.2. THE switch STATEMENT

The switch statement can be used instead of the else if construct to implement a sequence of parallel alternatives. Its syntax is

switch (expression)



```
{case constant1: statementList1;
case constant2: statementList2;
case constant3: statementList3;
:
:
case constantN: statementListN;
default: statementList0;
}
```

The switch statement has four components:

- Switch
- Case
- default
- Break

Where default and break are optional

### Example

```
int main ()
{int score;
cout << "Enter your test score: "; cin >> score;
switch (score/10)
{case 10:
case 9: cout << "Your grade is an A." << endl; break;
case 8: cout << "Your grade is a B." << endl; break;
case 7: cout << "Your grade is a C." << endl; break;
case 6: cout << "Your grade is a D." << endl; break;
case 5:
case 4:
case 3:
case 2:
case 1:
case 0: cout << "Your grade is an F." << endl; break;
default: cout << "Error: score is out of range.\n";
}
cout << "Goodbye." << endl;
}
```

## Output

Enter your test score: 83

Your grade is a B.

Goodbye.

First the program divides the score by 10 to reduce the range of values to 0–10. So, in the test run, the score 83 reduces to the value 8, the program execution branches to case 8, and prints the output shown. Then the break statement causes the program execution to branch to the first statement after the switch block. That statement prints “Goodbye.”. Note that scores in the ranges 101 to 109 and -9 to -1 produce incorrect results.

### 1.3.3. Repetition Statements/Iteration /Loop

Repetition statements control a block of code to be executed repeatedly for a fixed number of times or until a certain condition fails. There are three C++ repetition statements:

- 1) The For loop
- 2) The While loop
- 3) The do...while loop

#### 1.3.3.1. The *for* loop

**for** (*initialization; condition; increase*) *statement*; its main function is to repeat *statement* while *condition* remains true, like the *while* loop. But in addition, **for** provides places to specify an *initialization* instruction and an *increase* instruction. So this loop is specially designed to perform a repetitive action with a counter.

It works in the following way:

1. **initialization** is executed. Generally, it is a initial value setting for a counter variable. This is executed only once.
2. **condition** is checked, if it is true the loop continues, otherwise the loop finishes and *statement* is skipped.
3. **statement** is executed. As usual, it can be either a single instruction or a block of instructions enclosed within curly brackets { }.
4. **finally**, whatever is specified in the *increase* field is executed and the loop gets back to step 2.

#### Example 1: Using a for Loop to Compute a Sum of Consecutive Integers

```
int main ()
{
    int n;
    cout << "Enter a positive integer: ";
```



```

cin >> n;
long sum=0;
for (int i=1; i <= n; i++)
    sum += i;
cout << "The sum of the first " << n << " integers is " << sum;
}

```

#### EXAMPLE 2, Reusing for Loop Control Variable Names

```

int main ()
{
    int n;
    cout << "Enter a positive integer: ";
    cin >> n;
    long sum=0;
    for (int i=1; i < n/ 2; i++) // the scope of this i is this loop
        sum += i;
    for (int i=n/2; i <= n; i++) // the scope of this i is this loop
        sum += i;
    cout << "The sum of the first " << n << " integers is "
    << sum << endl;
}

```

#### Example 3. For loop multiplication table

```

#include <iomanip> // defines setw()
#include <iostream> // defines cout
using namespace std;
int main()
{ for (int x=1; x <= 12; x++)
{ for (int y=1; y <= 12; y++)
    cout << setw(4) << x*y;
    cout << endl;
}
}

```

### 1.3.3.2. While loop

Example 1 , Using a while Loop to Compute a Sum of Consecutive Integers

This program computes the sum  $1 + 2 + 3 + \dots + n$ , for an input integer  $n$ :

```
int main()
{
    int n, i = 1;
    cout << "Enter a positive integer: ";
    cin >> n;
    long sum=0;
    while (i <= n)
        sum += i++;
    cout << "The sum of the first " << n << " integers is " << sum;
}
```

EXAMPLE 2 Using a while Loop to Compute a Sum of Reciprocals

This program computes the sum of reciprocals  $s = 1 + 1/2 + 1/3 + \dots + 1/n$ , where  $n$  is the smallest integer for which  $n \geq s$ :

```
int main ()
{
    int bound;
    cout << "Enter a positive integer: ";
    cin >> bound;
    double sum=0.0;
    int i=0;
    while (sum < bound)
        sum += 1.0/++i;
    cout << "The sum of the first " << i
    << " reciprocals is " << sum << endl;
}
```

### 1.3.3.3. Do.... while loop

The do while loop.

**do statement while (condition);**



Its functionality is exactly the same as the *while* loop except that *condition* in the *do-while* is evaluated after the execution of *statement* instead of before, granting at **least one execution** of *statement* even if *condition* is never fulfilled.

#### Example 1 Using a do, while Loop to Compute a Sum of Consecutive Integers

This program has the same effect as the one in Example 41 on page 3:

```
int main ()
{
    int n,i=0;
    cout << "Enter a positive integer: ";
    cin >> n;
    long sum=0;
    do
    sum += i++;
    while (i <= n);
    cout << "The sum of the first " << n << " integers is " << sum;
}
```

## 1.4. Arrays

An array is a collection of data which shares a common identifier and data type. A collection of identical data objects, which are stored in consecutive memory locations under a common heading or a variable name. In other words, an array is a group or a table of values referred to by the same name. The individual values in array are called elements. Array elements are also variables. Set of values of the same type, which have a single name followed by an index. In C++, square brackets appear around the index right after the name. A block of memory representing a collection of many simple data variables stored in a separate array element, and the computer stores all the elements of an array consecutively in memory.

### 1.4.1. Properties of Arrays

Arrays in C++ are zero-bounded; that is the index of the first element in the array is 0 and the last element is  $N-1$ , where  $N$  is the size of the array. It is illegal to refer to an element outside of the array bounds, and your program will crash or have unexpected results, depending on the compiler. Array can only hold values of one type.

### 1.4.2. Array declaration

Declaring the name and type of an array and setting the number of elements in an array is called dimensioning the array. The array must be declared before one uses it like other variables. In the array declaration one must define:

1. The type of the array (i.e. integer, floating point, char etc.)
2. Name of the array,
3. The total number of memory locations to be allocated or the maximum value of each subscript.  
i.e. the number of elements in the array. So the general syntax for the declaration is:

**`DataTypename arrayname [array size];`**

The expression array size, which is the number of elements, must be a constant such as 10 or a symbolic constant declared before the array declaration, or a constant expression such as `10*sizeof(int)`, for which the values are known at the time compilation takes place.

**Note:** array size cannot be a variable whose value is set while the program is running.

Thus, to declare an integer with size of 10 having a name of num is:

```
int num [10];
```

This means: ten consecutive two-byte memory location will be reserved with the name num. That means, we can store 10 values of type int without having to declare 10 different variables each one with a different identifier. Instead of that, using an array we can store 10 different values of the same type, int for example, with a unique identifier.

### 1.4.3. Initializing Arrays

You can initialize C++ array elements either one by one or using a single statement as follows –

```
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

The number of values between braces `{ }` cannot be larger than the number of elements that we declare for the array between square brackets `[ ]`. Following is an example to assign a single element of the array –

If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write –

```
double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

### Multidimensional Arrays

Multidimensional arrays can be described as arrays of arrays. For example, a bi-dimensional array can be imagined as a bi-dimensional table of a uniform concrete data type.

		0	1	2	3	4
matrix	0					
	1					
	2					



Matrix represents a bi-dimensional array of 3 per 5 values of type int. The way to declare this array would be:

```
int matrix[3][5];
```

For example, the way to reference the second element vertically and fourth horizontally in an expression would be:

Multidimensional arrays are not limited to two indices (two dimensions). They can contain so many indices as needed, although it is rare to have to represent more than 3 dimensions. Just consider the amount of memory that an array with many indices may need.

For example:

```
char century [100][365][24][60][60];
```

Assigns a char for each second contained in a century, that is more than 3 billion chars! What would consume about 3000 megabytes of RAM memory if we could declare it? Multidimensional arrays are nothing else than an abstraction, since we can simply obtain the same results with a simple array by putting a factor between its indices:

```
int matrix [3][5]; is equivalent to
```

```
int matrix [15]; (3 * 5 = 15)
```

#### 1.4.4. Accessing Array Elements

An element is accessed by indexing the array name. This is done by placing the index of the element within square brackets after the name of the array. For example

```
double salary = balance[9];
```

The above statement will take 10<sup>th</sup> element from the array and assign the value to salary variable. Following is an example, which will use all the above-mentioned three concepts viz. declaration, assignment and accessing arrays –

```
#include <iostream>
using namespace std;

#include <iomanip>
using std::setw;

int main () {

    int n[ 10 ]; // n is an array of 10 integers

    // initialize elements of array n to 0
    for ( int i = 0; i < 10; i++ ) {
        n[ i ] = i + 100; // set element at location i to i + 100
    }
    cout << "Element" << setw( 13 ) << "Value" << endl;
```

```
// output each array element's value
for ( int j = 0; j < 10; j++ ) {
    cout << setw( 7 ) << j << setw( 13 ) << n[ j ] << endl;
}

return 0;
}
```

This program makes use of `setw()` function to format the output. When the above code is compiled and executed, it produces the following result –

Element	Value
0	100
1	101
2	102
3	103
4	104
5	105
6	106
7	107
8	108

9            109

## 1.5. Pointers

A pointer is a variable which stores the address of another variable. The only difference between pointer variable and regular variable is the data they hold. There are two pointer operators in C++:

**& the address of operator**

**\* The dereference operator**

### Example explained

Create a pointer variable with the name `ptr`, that **points to a string** variable, by using the asterisk sign `*(string* ptr)`. Note that the type of the pointer has to match the type of the variable you're working with.

Use the `&` operator to store the memory address of the variable called `food`, and assign it to the pointer.

Now, `ptr` holds the value of `food`'s memory address.

## 1.6. Functions

A function is a **block of code which only runs when it is called**. You can pass data, known as parameters, into a function. Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

So as shown above, each function has:



- **Return type:** It is the value that the functions return to the calling function after performing a specific task.
- **function Name :** Identifier used to name a function.
- **Parameter List:** Denoted by param1, param2,... paramn in the above syntax. These are the arguments that are passed to the function when a function call is made. The parameter list is optional i.e. we can have functions that have no parameters.
- **Function body:** A group of statements that carry out a specific task.

### 1.6.1. Function Declaration

A function declaration tells the compiler about the return type of function, the number of parameters used by the function and its data types. Including the names of the parameters in the function, the declaration is optional. The function declaration is also called as a function prototype.

We have given some examples of the function declaration below for your reference.

```
int sum (int, int);
```

Above declaration is of a function 'sum' that takes two integers as parameters and returns an integer value.

```
void swap (int, int);
```

This means that the swap function takes two parameters of type int and does not return any value and hence the return type is void.

```
void display 0;
```

The function display does not take any parameters and also does not return any type.

### 1.6.2. Function Definition

A function definition contains everything that a function declaration contains and additionally it also contains the body of the function enclosed in braces ({}).

In addition, it should also have named parameters. When the function is called, control of the program passes to the function definition so that the function code can be executed. When execution of the function is finished, the control passes back to the point where the function was called.

**For the above declaration of swap function, the definition is as given below:**

```
void swap (int a, int b){
    b = a + b;
    a = b - a;
    b = b - a;
}
```

Note that declaration and definition of a function can go together. If we define a function before referencing it then there is no need for a separate declaration.

Let us take a complete programming Example to demonstrate a function.

```
#include <iostream>
using namespace std;

void swap(int a, int b) { //here a and b are formal parameters
    b = a + b;
    a = b - a;
    b = b - a;
    cout<<"\nAfter swapping: ";
    cout<<"a = "<<a;
    cout<<"\tb = "<<b;
    return;
}
int main()
{
    int a,b;
    cout<<"Enter the two numbers to be swapped: "; cin>>a>>b;

    cout<<"a = "<<a;
    cout<<"\tb = "<<b;
    swap(a,b); //here a and b are actual parameters
}
```

**Output:**

Enter the two numbers to be swapped: 5 3

a = 5 b = 3

After swapping: a = 3 b = 5

In the above example, we see that there is a function swap that takes two parameters of type int and returns nothing. Its return type is void. As we have defined this function before function main, which is a calling function, we have not declared it separately.

In the function main, we read two integers and then call the swap function by passing these two integers to it. In the swap function, the two integers are exchanged using a standard logic and the swapped values are printed.

### 1.6.3. Calling A Function

When we have a function in our program, then depending on the requirement we need to call or invoke this function. Only when the function is called or invoked, the function will execute its set of statements to provide the desired results.



The function can be called from anywhere in the program. It can be called from the main function or from any other function if the program is using more than one function. The function that calls another function is called the "Calling function".

In the above example of swapping numbers, the swap function is called in the main function. Hence the main function becomes the calling function.

#### 1.6.4. Formal and Actual Parameters

We have already seen that we can have parameters for the functions. The function parameters are provided in the function definition as a parameter list that follows the function name. When the function is called, we have to pass the actual values of these parameters so that using these actual values the function can carry out its task.

The parameters that are defined in the function definition are called **Formal Parameters**. The parameters in the function call which are the actual values are called **Actual Parameters**.

In the above example of swapping numbers, we have written the comments for formal and actual parameters. In the calling function i.e. main, the value of two integers is read and passed to the swap function. These are the actual parameters.

We can see the definitions of these parameters in the first line of the function definition. These are the formal parameters.

Note that the type of formal and actual arguments should match. The order of formal and actual parameters should also match.

#### 1.6.5. Return Values

Once the function performs its intended task, it should return the result to the calling function. For this, we need the return type of the function. The function can return a single value to the calling function. The return type of the function is declared along with the function prototype.

Let's take an Example of adding two numbers to demonstrate the return types.

```
#include <iostream>
using namespace std;

int sum(int a, int b){
    return (a+b);
}

int main()
{
    int a, b, result;
    cout<<"Enter the two numbers to be added: "; cin>>a>>b;
```

```
result = sum(a,b);  
cout<<"\nSum of the two numbers : "<<result;  
}
```

### Output:

Enter the two numbers to be added: 11 11

Sum of the two numbers: 22

In the above example, we have a function `sum` that takes two integer parameters and returns an integer type. In the main function, we read two integers from the console input and pass it to the `sum` function. As the return type is an integer, we have a result variable on the LHS and RHS is a function call.

When a function is executed, the expression  $(a+b)$  returned by the function `sum` is assigned to the result variable. This shows how the return value of the function is used.

## Void Functions

We have seen that the general syntax of function requires a return type to be defined. But if in case we have such a function that does not return any value, in that case, what do we specify as the return type? The answer is that we make use of valueless type “void” to indicate that the function does not return a value.

In such a case the function is called “void function” and its prototype will be like

```
void functionName(param1,param2,...param 3);
```

**Note:** It is considered as a good practice to include a statement “return;” at the end of the void function for clarity.

### 1.6.6. Passing Parameters to Functions

We have already seen the concept of actual and formal parameters. We also know that actual parameters pass values to a function which is received by the formal parameters. This is called the passing of parameters.

In C++, we have certain ways to pass parameters as discussed below.

#### Pass by Value



In the program to swap two integers that we discussed earlier, we have seen that we just read integers 'a' and 'b' in main and passed them to the swap function. This is the pass by value technique.

In pass by value technique of parameter passing, the copies of values of actual parameters are passed to the formal parameters. Due to this, the actual and formal parameters are stored at different memory locations. Thus, changes made to formal parameters inside the function do not reflect outside the function.

**We can understand this better by once again visiting the swapping of two numbers.**

```
#include <iostream>
using namespace std;

void swap(int a, int b) { //here a and b are formal parameters
    b = a + b;
    a = b - a;
    b = b - a;

    cout<<"\nAfter swapping inside Swap:\n ";
    cout<<"a = "<<a;
    cout<<"\tb = "<<b;
    return;
}

int main()
{
    int a,b;
    cout<<"Enter the two numbers to be swapped: "; cin>>a>>b;

    cout<<"a = "<<a;
    cout<<"\tb = "<<b;
    swap(a,b);
    cout<<"\nAfter swapping inside Main:\n ";
    cout<<"a = "<<a;
    cout<<"\tb = "<<b;
}
```

**Output:**

```
Enter the two numbers to be swapped: 3 2
a = 3 b = 2
After swapping inside Swap:
a = 2 b = 3
After swapping inside Main:
a = 3 b = 2
```

We have simply modified the earlier program to print the values of formal parameters & actual parameters before and after the function call.

As seen from the output, we pass values **a=3 and b=2** initially. These are the actual parameters. Then after swapping inside the swap function, we see that the values are actually swapped and **a=2 and b=3**.

However, after the function call to swap, in the main function, the values of a and b are still 3 and 2 respectively. This is because the actual parameters passed to function where it has a copy of the variables. Hence although the formal parameters were exchanged in the swap function they were not reflected back.

Though Pass by value technique is the most basic and widely used one, because of the above limitation, we can only use it in the cases where we do not require the function to change values in calling the function.

### Pass by Reference

Pass by reference is yet another technique used by C++ to pass parameters to functions. In this technique, instead of passing copies of actual parameters, we pass references to actual parameters.

**Note:** References are nothing but aliases of variables or in simple words, it is another name that is given to a variable. Hence a variable and its reference share same memory location. We will learn references in detail in our subsequent tutorial.

In pass by reference technique, we use these references of actual parameters and as a result, the changes made to formal parameters in the function are reflected back to the calling function.

**We modify our swap function for our readers to understand the concept better.**

```
#include <iostream>
#include <string>
using namespace std;

void swap(int &a, int &b){
    int temp = a;
    a = b;
    b = temp;
}

int main()
{
    int a,b;
    cout<<"Enter the two numbers to be swapped: "; cin>>a>>b;
```



```

cout<<"a = "<<a;
cout<<"\tb = "<<b;
swap(a,b);
cout<<"\nAfter swapping inside Main:\n ";
cout<<"a = "<<a;
cout<<"\tb = "<<b;
}

```

#### Output:

Enter the two numbers to be swapped: 25 50

a = 25 b = 50

After swapping inside Main:

a = 50 b = 25

Note: The pass by reference technique shown in the above example. We can see that the actual parameters are passed as it is. But we append an '&' character to the formal parameters indicating that it's a reference that we are using for this particular parameter.

Hence the changes made to the formal parameters in the swap function reflect in the main function and we get the swapped values.

#### Pass by Pointer

In C++, we can also pass parameters to function using pointer variables. The pass by pointer technique produces the same results as that of pass by reference. This means that both formal and actual parameters share the same memory locations and the changes made in function are reflected in the calling function.

The only difference that in a **pass by reference** we deal with references or aliases of parameters whereas in a pass by pointer technique we use pointer variables to pass the parameters.

Pointer variables differ with the references in which pointer variables point to a particular variable and unlike references, we can change the variable that it points to. We will explore the details of the pointer in our subsequent tutorials.

We present the swapping of two integers again to demonstrate the Pass by Pointer technique.

```

#include <iostream>
#include <string>
using namespace std;

void swap(int *a, int *b)

```

```

{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main()
{
    int a,b;
    cout<<"Enter the two numbers to be swapped: "; cin>>a>>b;
    cout<<"a = "<<a;
    cout<<"\tb = "<<b;
    swap(a,b);
    cout<<"\nAfter swapping inside Main:\n ";
    cout<<"a = "<<a;
    cout<<"\tb = "<<b;
}

```

### Output:

Enter the two numbers to be swapped: 23 54

a = 23 b = 54

After swapping inside Main:

a = 54 b = 23

Thus as already said, there is no difference in the output of the program. The only difference is in the way in which the parameters are passed. We can notice that formal parameters are pointer variables here.

### Default Parameters

In C++, we can provide default values for function parameters. In this case, when we invoke the function, we don't specify parameters. Instead, the function takes the default parameters that are provided in the prototype.

The following Example demonstrates the use of Default Parameters.

```

#include <iostream>
#include <string>
using namespace std;

int mathoperation(int a, int b = 3, int c = 2){

    return ((a*b)/c);
}

int main()
{
    int a,b,c;

    cout<<"Enter values for a,b and c: "; cin>>a>>b>>c;
}

```



```

cout<<endl;
cout<<"Call to mathoperation with 1 arg : "<<mathoperation(a);
cout<<endl;
cout<<"Call to mathoperation with 2 arg : "<<mathoperation(a,b);
cout<<endl;
cout<<"Call to mathoperation with 3 arg : "<<mathoperation(a,b,c);
cout<<endl;
}

```

#### Output:

Enter values for a,b and c: 10 4 6

Call to mathoperation with 1 arg: 15

Call to mathoperation with 2 arg: 20

Call to mathoperation with 3 arg: 6

As shown in the code example, we have a function 'mathoperation' that takes three parameters out of which we have provided default values for two parameters. Then in the main function, we call this function three times with a different argument list.

The first call is with only one argument. In this case, the other two arguments will have default values. The next call is with two arguments. In this case, the third argument will have a default value. The third call is with three arguments. In this case, as we have provided all the three arguments, default values will be ignored.

Note that while providing default parameters, we always start from the right-most parameter. Also, we cannot skip a parameter in between and provide a default value for the next parameter.

Now let us move onto a few special function related concepts that are important from a programmer's point of view.

### Const Parameters

We can also pass constant parameters to functions using the 'const' keyword. When a parameter or reference is const, it cannot be changed inside the function.

Note that we cannot pass a const parameter to a non-const formal parameter. But we can pass const and non-const parameter to a const formal parameter.

Similarly, we can also have const return-type. In this case, also, the return type cannot be modified.

Let us see a code Example that uses const references.

```
#include <iostream>
#include <string>
using namespace std;

int addition(const int &a, const int &b){
    return (a+b);
}

int main()
{
    int a,b;
    cout<<"Enter the two numbers to be swapped: "; cin>>a>>b;
    cout<<"a = "<<a;
    cout<<"\tb = "<<b;
    int res = addition(a,b);
    cout<<"\nResult of addition: "<<res;
}
```

#### Output:

```
Enter the two numbers to be swapped: 22 33
a = 2 b = 33
Result of addition: 55
```

In the above program, we have const formal parameters. Note that the actual parameters are ordinary non-const variables which we have successfully passed. As formal parameters are const, we cannot modify them inside the function. So we just perform the addition operation and return the value.

If we try to modify the values of a or b inside the function, then the compiler will issue an error.

#### 1.6.7. Inline Functions

We know that in order to make a function call, internally it involves a compiler storing the state of the program on a stack before passing control to the function.

When the function returns, the compiler has to retrieve the program state back and continue from where it left. This poses an overhead. Hence, in C++ whenever we have a function consisting of few statements, there is a facility that allows it to expand inline. This is done by making a function inline.

So inline functions are the functions that are expanded at runtime, saving the efforts to call the function and do the stack modifications. But even if we make a function as inline, the compiler does not guarantee that it will be expanded at runtime. In other words, it's completely dependent on the compiler to make the function inline or not.



Some compilers detect smaller functions and expand them inline even if they are not declared inline.

**Following is an Example of an Inline Function.**

```
inline int addition(const int &a,const int &b){  
    return (a+b);  
}
```

As shown above, we precede the function definition with a keyword “inline” in order to make a function inline.

**Using Structs in Functions**

We can pass structure variables as parameters to function in a similar way in which we pass ordinary variables as parameters.

**This is shown in the following Example.**

```
#include <iostream>  
#include <string>  
using namespace std;  
  
struct PersonInfo  
{  
    int age;  
    char name[50];  
    double salary;  
};  
  
void printStructInfo(PersonInfo p)  
{  
    cout<<"PersonInfo Structure:";  
    cout<<"nAge:"<<p.age;  
    cout<<"nName:"<<p.name;  
    cout<<"nSalary:"<<p.salary;  
}  
  
int main()  
{  
    PersonInfo p;  
    cout << "Enter name: ";  
    cin.get(p.name, 50);  
    cout << "Enter age: "; cin >> p.age;  
    cout << "Enter salary: "; cin >> p.salary;  
  
    printStructInfo(p);  
}
```

**Output:**

```
Enter name: Vedang  
Enter age: 22
```

```
Enter salary: 45000.00
PersonInfo Structure:
Age:22
Name: Vedang
Salary:45000
```

Get URL

options compilation execution

Enter name: Vedang  
Enter age: 22  
Enter salary: 45000.00  
PersonInfo Structure:  
Age:22  
Name:Vedang  
Salary:45000

Exit code: 0 (normal program termination)

As shown in the above program, we pass a structure to function in a similar manner as other variables. We read values for structure members from the standard input and then pass a structure to a function that displays the structure.

## 1.7. C++ Files

The **fstream** library allows us to work with files.

To use the **fstream** library, include both the standard **<iostream>** AND the **<fstream>** header file:

### Example

```
#include <iostream>
#include <fstream>
```

There are three classes included in the **fstream** library, which are used to create, write or read files

Class	Description
<b>ofstream</b>	Creates and writes to files
<b>ifstream</b>	Reads from files
<b>fstream</b>	A combination of ofstream and ifstream: creates, reads, and writes to files

### Create and Write to a File

To create a file, use either the **ofstream** or **fstream** class, and specify the name of the file.

To write to the file, use the insertion operator (**<<**).

### Example

```
#include <iostream>
#include <fstream>
```



```
using namespace std;
int main() {
    // Create and open a text file
    ofstream MyFile("filename.txt");
    // Write to the file
    MyFile << "Files can be tricky, but it is fun enough!";
    // Close the file
    MyFile.Close();
}
```

### Why do we close the file?

It is considered good practice, and it can clean up unnecessary memory space.

### Read a File

To read from a file, use either the `ifstream` or `fstream` class, and the name of the file.

Note that we also use a `while` loop together with the `getline()` function (which belongs to the `ifstream` class) to read the file line by line, and to print the content of the file:

### Example

```
// Create a text string, which is used to output the text file
string myText;
// Read from the text file
ifstream MyReadFile("filename.txt");
// Use a while loop together with the getline() function to read the file line by line
while (getline (MyReadFile, myText)) {
    // Output the text from the file
    cout << myText;
}
// Close the file
MyReadFile.close();
}
```

## 1.8. Object oriented Programming

### Module Objective

Object-oriented programming (OOP) is a way to organize and conceptualize a program as a set of interacting objects.

- The programmer defines the types of objects that will exist.
- The programmer creates object instances as they are needed.
- The programmer specifies how these various objects will communicate and interact with each other.

## What java?

Java is an **object-oriented**, class-based, concurrent, secured and general-purpose computer-programming language. It is a widely used robust technology.

Java is a programming language and a platform. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

## Application

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

## Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

### 1) Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

### 2) Web Application



An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

### 3) Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

### 4) Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

### Java Platforms / Editions

There are 4 platforms or editions of Java:

#### 1) Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as `java.lang`, `java.io`, `java.net`, `java.util`, `java.sql`, `java.math` etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

#### 2) Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

#### 3) Java ME (Java Micro Edition)

It is a micro platform that is dedicated to mobile applications.

#### 4) JavaFX

It is used to develop rich internet applications. It uses a lightweight user interface API.

### Java Programming Paradigm (Modeling)

Java is based on the concept of Object-Oriented programming. As the name suggests, at the center of it all is an object. Objects contain both data and the functionality that operates on that data. This is controlled by the following four **paradigms**, *Encapsulation*, *Inheritance*, *Information hiding* and *polymorphism*.

#### 1.8.1. Basic Concepts of Object-oriented Programming

##### Object

Object is a piece of code which represents the real-life entity

Represents any instance of a class

Object has three characteristics

- State
- Behavior

➤ Identifier

Real-world objects have *attributes* and *behaviors*.

Examples:

➤ Dog

- Attributes: breed, color, hungry, tired, etc.
- Behaviors: eating, sleeping, etc.

➤ Bank Account

- Attributes: account number, owner, balance
- Behaviors: withdraw, deposit

## Class

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. The definitions of the attributes and methods of an object are organized into a *class*. Thus, a class is the generic definition for a set of similar objects (i.e., *Person* as a generic definition for *Jane*, *Mitch* and *Sue*)

- A class can be thought of as a template used to create a set of objects.
- A class is a static definition; a piece of code written in a programming language.
- One or more objects described by the class are *instantiated* at runtime.
- The objects are called *instances* of the class.
- Class has three parts
  - Name
  - Variables
  - Methods

## Abstraction

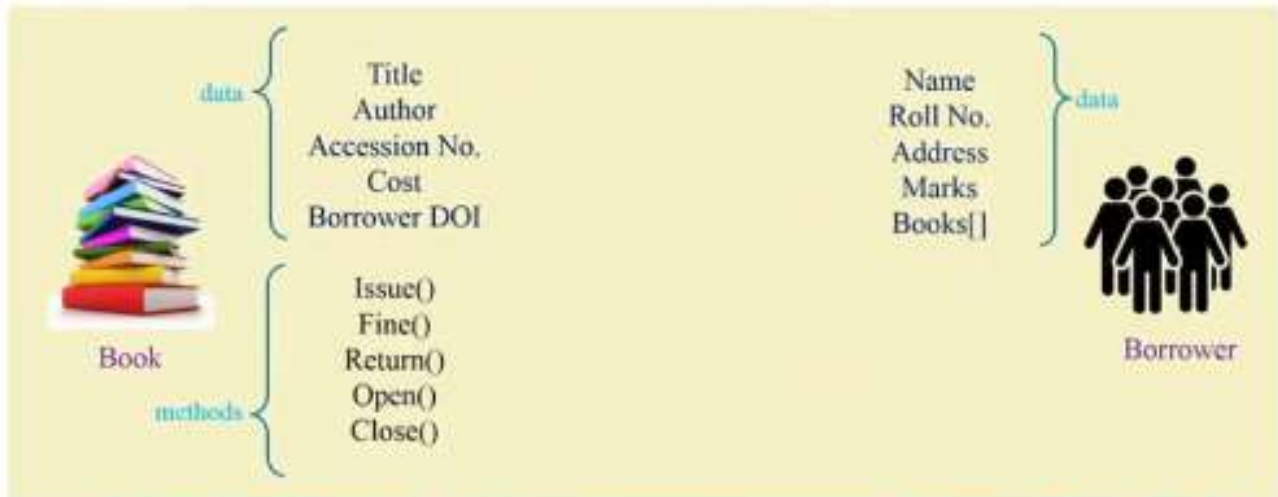
Showing only essential parts and hiding the implementation details. Example: downloading android application from play store.(.apk , .exe).

## Encapsulation

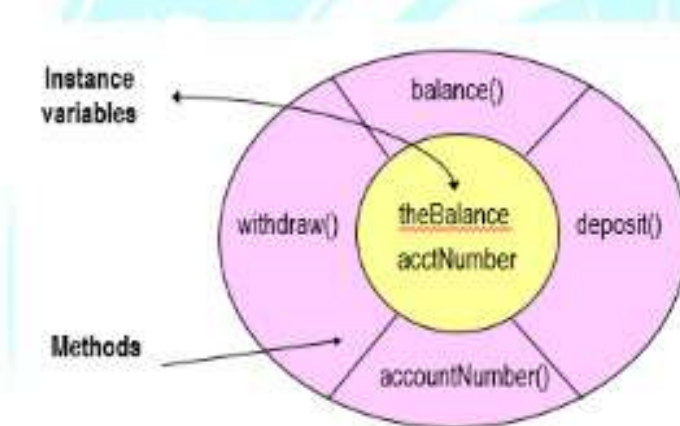
**Encapsulation** is the process of binding variables and methods under a single entity. Implementation of operations and object structure hidden, binding properties with functions When classes are defined, programmers can specify that certain methods or state variables remain hidden inside the class. These variables and methods are accessible from within the class, but not accessible outside it. The combination of collecting all the attributes of an object into a single class definition, combined with the ability to hide some definitions and type information within the class, is known as **encapsulation**.



## Encapsulation example



## Graphical Model of an Object



State variables make up the nucleus of the object. Methods surround and hide (encapsulate) the state variables from the rest of the program.

### Instance Methods and Instance Variables

The methods and variables described in this module so far are known as instance methods and instance variables.

- These state variables are associated with the one instance of a class; the values of the state variables may vary from instance to instance.
- Instance variables and instance methods can be public or private.
- It is necessary to instantiate (create an instance of) a class to use its instance variables and instance methods.

### Class Methods and Class Variables

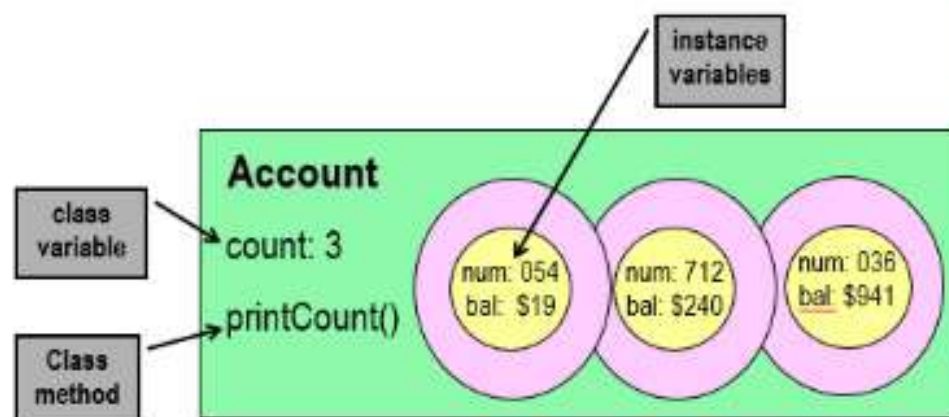
In addition to instance methods and instance variables, classes can also define *class methods* and *class variables*.

- These are attributes and behaviors associated with the class as a whole, not any one instance.
- Class variables and class methods can be public or private.
- It is not necessary to instantiate a class to use its class variables and class methods.

## Class Variables

A **class variable** defines an attribute of an entire class.

In contrast, an **instance variable** defines an attribute of a single instance of a class.



## Inheritance

Sharing of data within hierarchy scope, supports code reusability. Allows one class (subclass) to be defined as a special case of a more general class (super class). The process of forming a super class is referred to as **generalization**. The process of forming a subclass is referred to as **specialization**. The advantage of making a new class a subclass is that it will inherit attributes and methods of its parent class (also called the *superclass*).

- Subclasses extend existing classes in three ways:
  - By defining new (additional) attributes and methods.
  - By overriding (changing the behavior) existing attributes and methods.
  - By hiding existing attributes and methods.

## Subclasses

When a new class is developed a programmer can define it to be a **subclass** of an existing class. Subclasses are used to define special cases, extensions, or other variations from the originally defined class.

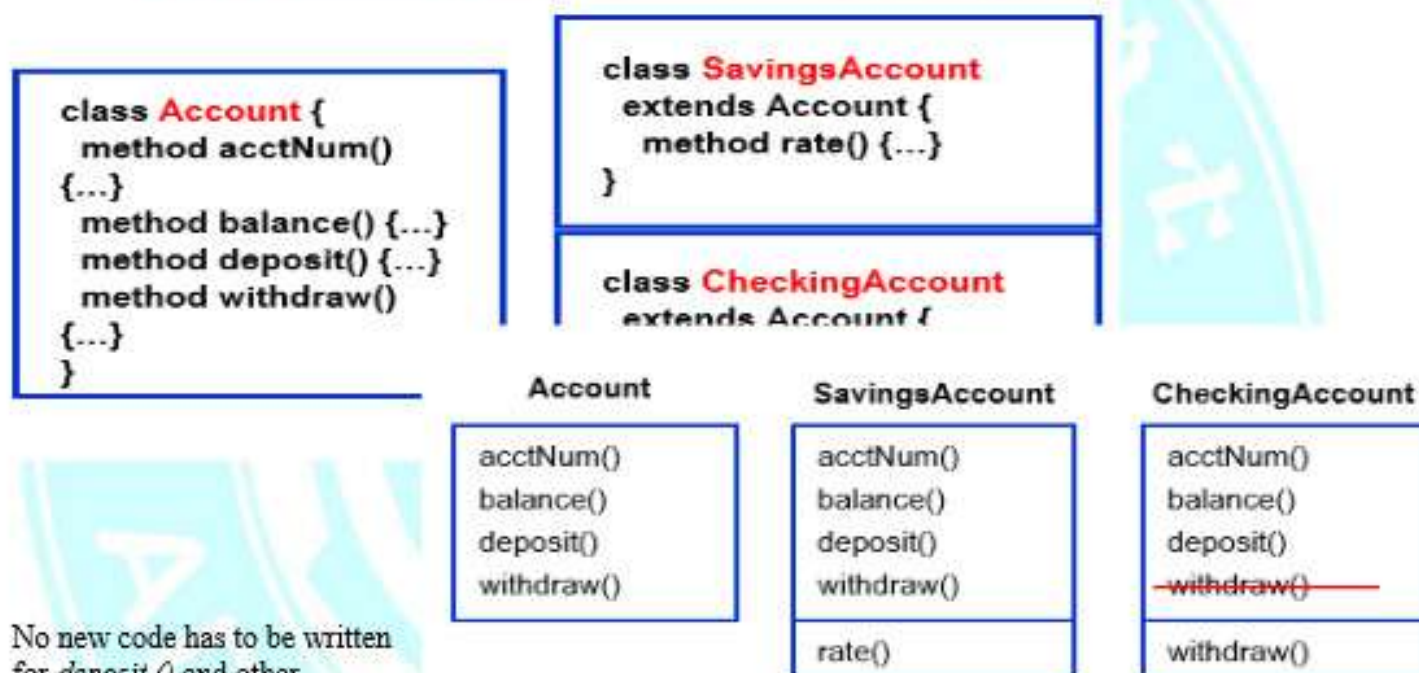
## Example



- **Terrier** can be defined as a subclass of **Dog**.
- **SavingsAccount** and **CheckingAccount** can be derived from the **Account** class (see following slides).



## New Account Types



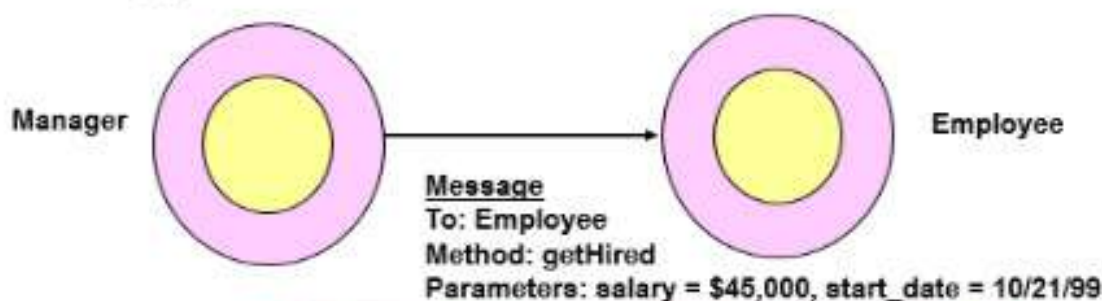
No new code has to be written for *deposit()* and other methods; they are inherited from the superclass.

**Inheritance can be**

- Single inheritance
- Multiple inheritance
- Hierarchal inheritance

## Messages

- Messages are information/requests that objects send to other objects (or to themselves).
- Message components include:
  - The name of the object to receive the message.
  - The name of the method to perform.
  - Any parameters needed for the method.



### Message Passing Benefits

Message passing supports all possible interactions between two objects. Message passing is the mechanism that is used to invoke a method of the object. Objects do not need to be part of the same process or on the same machine to interact with one another. Message passing is a run-time behavior; thus, it is not the same as a procedure call in other languages (compile-time). The address of the method is determined dynamically at run-time, as the true type of the object may not be known to the compiler.

### Polymorphism

Polymorphism is one of the essential features of an object-oriented language; this is the mechanism of decoupling the behavior from the message.

- The same message sent to different types of objects results in:
  - execution of behavior that is specific to the object and,
  - Possibly different behavior than that of other objects receiving the same message.
- Example: the message **draw()** sent to an object of type *Square* and an object of type *Circle* will result in different behaviors for each object.

There are many **forms of Polymorphism** in object-oriented languages, such as:

- **True Polymorphism:** Same method signature defined for different classes with different behaviors (i.e. **draw()** for the Classes *Circle* and *Square*)
- **Parametric Polymorphism:** This is the use of the same method name within a class, but with a different signature (different parameters).
- **Overloading:** This usually refers to operators (such as **+**, **-**, **\***, etc) when they can be applied to several types such as **int**, **floats**, **strings**, etc.
- **Overriding:** This refers to the feature of subclasses that replace the behavior of a parent class with new or modified behavior.

#### 1.8.2. Java Advantages, Scope and feature

Java is an object-oriented programming language and it was intended to serve as a new way to manage software complexity. Java refers to a number of computer software products and specification from Sun Microsystems that together provide a system for developing application software and deploying it in a



cross - platform environment. Java is used in variety of computing platforms from embedded devices and mobile phones on the low end, to enterprise servers and supercomputers on the high end. Java is nearly everywhere in mobile phones, web servers and enterprise applications while less common on desktop computers. Java applets are often used to provide improved functionality while browsing the World Wide Web.

### **Advantages of Java**

It is an **open source**, so users do not have to struggle with heavy license fees every year.

- Platform independent
- Java API's can easily be accessed by developers
- Java perform supports garbage collection, so memory management is automatic.
- Java always allocates objects on the stack.
- Java embrace the concept of exception specifications.
- Multi - platform support language and for support web-services.
- Using Java we can develop dynamic web applications.
- It allows you to create modular programs and reusable codes

### **C++ versus Java**

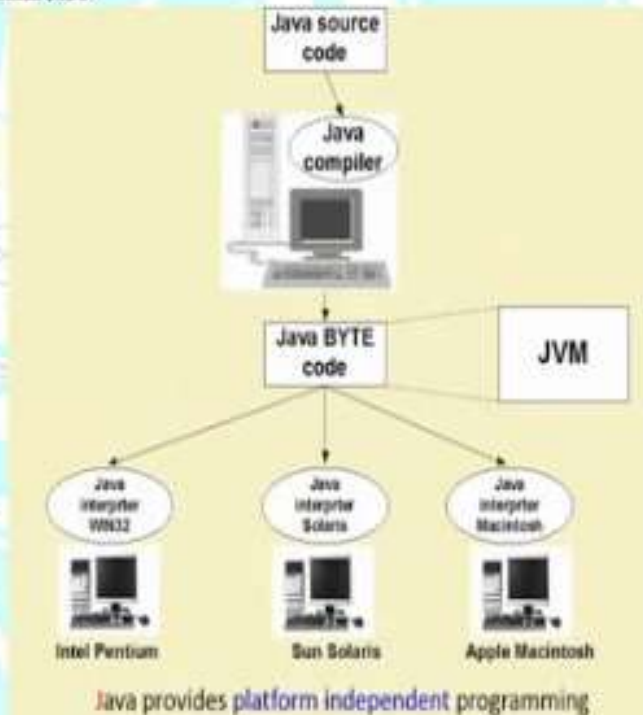
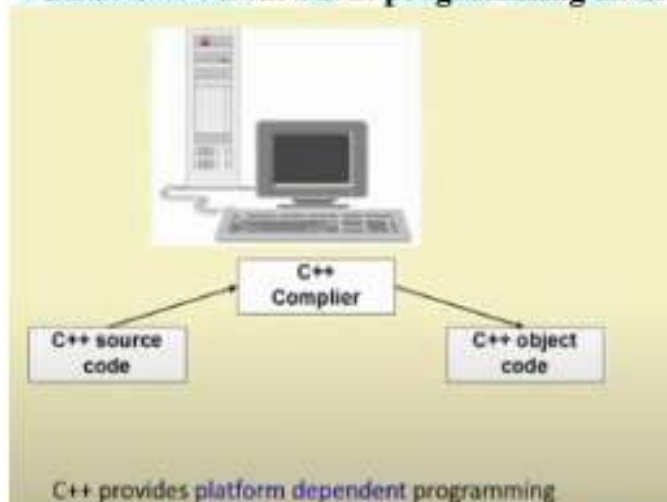
#### **Area of application**

- C++ is best suitable for developing large software
  - Library management system
  - Employee management system
  - passenger reservation system
  - etc
- Java is best suitable for developing communication /Internet application software
  - Network protocols
  - Internet programs
  - Web pages
  - Web browsers

#### **C++ versus Java: programming features**

Features		in C++	in Java
Data abstraction and encapsulation		✓	✓
Polymorphism		✓	✓
Binding	Static	✓	✓
	Dynamic	✓	✓
Inheritance	Single Inheritance	✓	✓
	Multiple Inheritance	✓	×
Operator overloading		✓	×
Template classes		✓	×
Global variables		✓	×
Header files		✓	×
Pointers		✓	×
Interface and packages		×	✓
API (Application Programming Interface)		×	✓

### 1.8.3. C++ versus Java: programming environment



### 1.8.4. Java tools available for java programming

- Java Software Developer's Kit(SDK)/JDK
- Editors
  - Net beans



- Eclipse
- Notepad ++

### There are seven programs in SDK

- *javac* – the Java Compiler
- *java* – the Java Interpreter
- *javadoc* – generates documentation in HTML
- *appletviewer* – the Java Interpreter to execute Java applets
- *jdb* – the Java Debugger to find and fix bugs in Java programs
- *javap* – the Java Disassembler to displays the accessible functions and data in a compiled class; it also displays the meaning of byte codes
- *javah* – to create interface between Java and C routines

### Packages in Java

#### API (Application Programming Interface) in Java SDK

- The API enables Java programmers to develop varieties of applets and applications
- It contains **nine** packages
  - *java.applet* – for applet programming
  - *java.awt* – the Abstract Windowing Toolkit for designing GUI like Button, Checkbox, Choice, Menu, Pannel, etc.
  - *java.io* – file input/output handling
  - *java.lang* – provides useful classes like to handle Object, Thread, Exception, String, System, Math, Float, Integer, etc.
  - *java.lang* – provides useful classes like to handle Object, Thread, Exception, String, System, Math, Float, Integer etc.
  - *java.net* – classes for network programming; supports TCP/IP networking protocols
  - *java.util* – it contains miscellaneous classes like Vector, Stack, List, Date, Dictionary, Hash etc.
  - *javax.swing* – for designing graphical user interface (GUI)
  - *java.sql* – for database connectivity (JDBC)

#### 1.8.5. Identifiers in Java

Identifiers are programmer given names. They identify *classes, methods, variables, objects, packages*.

#### Naming convention:

- Identifier must begin with Letters, Underscore characters ( `_` ) or any currency symbol

- Remaining characters
  - Letters
  - Digits
- As long as you like!! (until you are tired of typing)
- The first letter cannot be a digit
- **Java is case sensitive language**

#### Identifiers' naming conventions

- **Class names:** starts with cap letter and should be inter-cap e.g., *StudentGrade*
- **Variable names:** start with lower case and should be inter-cap e.g., *varTable*
- **Method names:** start with lower case and should be inter-cap e.g., *calTotal*
- **Constants:** often written in all capital and use underscore if you are using more than one word.

#### Declaration and assignment statements

```
int a, b = 0;
a = 123;
b = 45;
int c = a + b;
System.out.print("The sum is" + c);
```

#### 1.8.6. Variables

There are three kinds of variables in Java.

1. Instance variables: variables that hold data for an instance of a class.
2. Class variables: variables that hold data that will be shared among all instances of a class.
3. Local variables: variables that pertain only to a block of code.

#### Scope and Lifetime of variables

- scope is the region of a program within which the variable can be referred to by its simple name
- scope also determines when the system creates and destroys memory for the variable
  - **A block defines a scope.** Each time you create a block of code, you are creating a new, nested scope.
  - Objects declared in the outer block will be visible to code within the inner block down from the declaration. (The reverse is not true)
  - **Variables are created** when their scope is entered, and destroyed when their scope is left. This means that a variable will not hold its value once it has gone out of scope.



- Variable declared within a block will lose its value when the block is left. Thus, the **lifetime** of a variable is confined to its scope.

You can't declare a variable in an inner block to have the same name as the one up in an outer block.

But it is possible to declare a variable down outside the inner block using the same name.

### 1.8.7. Expression

- Combine literals, variables, and operators to form expressions
- Expression is segments of code that perform computations and return values.
- Expressions can contain: **Example:**
  - a number literal, 3.14
  - a variable, count
  - a method call, Math.sin(90)
  - an operator between two expressions (binary operator),  
phase + Math.sin(90)
  - an operator applied to one expression (unary operator),  
-discount
  - expressions in parentheses.  
(3.14-amplitude)

### Statements

- Roughly equivalent to sentences in natural languages
- Forms a complete unit of execution.
- Terminating the expression with a semicolon (;)
- Three kinds of statements in java
  - expression statements
  - declaration statements
  - control flow statements
- expression statements
  - Null statements
  - Assignment expressions
  - Any use of ++ or --
  - Method calls
  - Object creation expressions

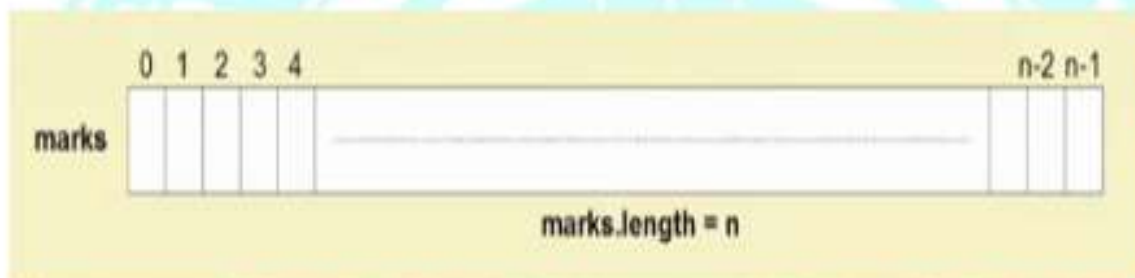
### Example

- aValue = 8933.234; //assignment statement
- aValue++; //increment statement

- `System.out.println(aValue);` //method call statement
- `Integer integerObject = new Integer(4);` //object creation
- A declaration statement declares a variable
  - `double aValue = 8933.234;` // declaration statement
- A *control flow statement* regulates the order in which statements get executed  
E.g - if, for

### 1.8.8. Arrays in Java

- An array is a finite ordered and collection of homogenous data elements
- Following are the three tasks to manipulate an array in Java
  - Declaration of an array
  - Allocate memory to it
  - Loading values into the array



### Creating an array

Define and allocate memory together

```
<type> <arrayName> [ ] = new <type> [<size>];
```

Example:

```
int x [ ] = new int [100];
```

<arrayName>;

```
int x[ ];
```

```
int [ ] x;
```

### Allocation of memory for an array

```
<arrayName> = new <type> [<size>];
```

Example:

```
x = new int [100 ];
```



## ■ Initialization of Array

```
<arrayName> [<subscript> ] = <value>;
```

Example:

```
x [5] = 100;
```

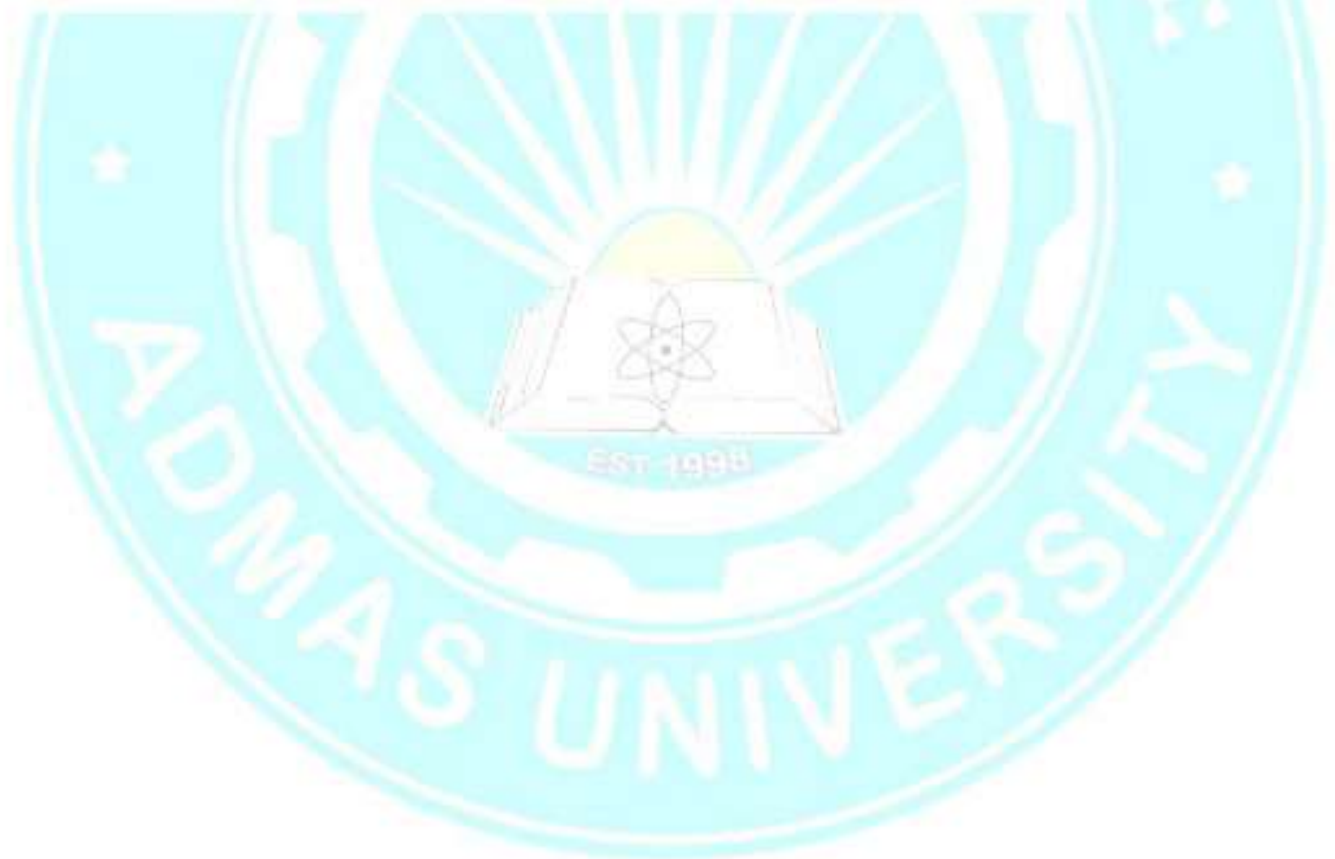
```
for (int i = 0; i < 100; i++)  
    x[i] = <value>;
```

## ■ Initialization of Array :an alternative way

```
<type> <arrayName> [ ] = { <list of values> };
```

Example:

```
int x [ ] = {12, 3, 9, 15};
```



### 7.8.1 Processing elements in an array

- **Insertion**
  - Insertion at any location
  - Insertion at front
  - Insertion at end
  - Insertion in sorted order
- **Deletion**
  - Deletion of a particular element
  - Deletion of an element at a particular location
  - Deletion of the element at front
  - Deletion of the element at end
- **Searching and Traversal**
  - Finding the smallest and largest element
  - Printing all elements or some specific element
- **Sorting**
  - In ascending order, descending order, lexicographical order etc.

### Creating 2D array

Declare and Allocate

Example:

```
int myArray [ ] [ ];  
myArray = new int [3] [4];
```

OR

```
int myArray [ ] [ ] = new int [3] [4];
```



## Loading 2D array

### Initializing 2D Array: an example

1	2	3
4	5	6

```
int myArray [2] [3] = {1, 2, 3, 4, 5, 6};
```

OR

```
int myArray [ ] [ ] = ( {1, 2, 3}, {4, 5, 6} );
```

### 1.8.9. Java Applets

Java programs are available in two flavors

#### 1. Applet

- A Java applet is a program that appears embedded in a web document and applet come into effect when the browser the web page.
- Applet is a small program/ window run on web browser.

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

#### Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac OS etc.

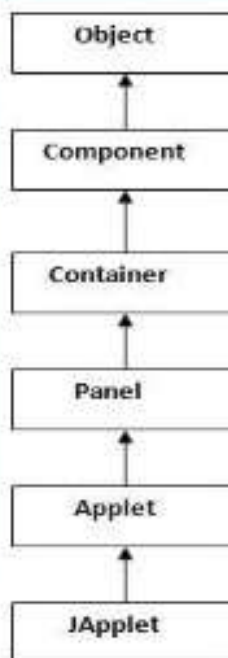
#### Drawback of Applet

- Plugin is required at client browser to execute applet.

### Do You Know

- Who is responsible to manage the life cycle of an applet ?
- How to perform animation in applet ?
- How to paint like paint brush in applet ?
- How to display digital clock in applet ?
- How to display analog clock in applet ?
- How to communicate two applets ?

### Hierarchy of Applet





As displayed in the above diagram, Applet class extends Panel. Panel class extends Container which is the subclass of Component.

### Lifecycle methods for Applet:

The `java.applet.Applet` class 4 life cycle methods and `java.awt.Component` class provides 1 life cycle methods for an applet.

### `java.applet.Applet` class

For creating any applet `java.applet.Applet` class must be inherited. It provides 4 life cycle methods of applet.

### Application

- It is similar to all kinds of programs like in C, C++, Pascal etc. to solve a problem.

### Applet program writing

```
import java.applet.*;  
import java.awt.*;  
Public Class HelloWorld extends Applet {  
    public void paint(Graphics g){  
        g.drwaString("Hello World !",150 , 150);  
    }  
}
```

#### 1.8.10. Thread in Java

- **Process and Thread** are two basic units of Java program execution.
- **Process:** A process is a self contained execution environment and it can be seen as a program or application.
- **Thread:** it can be called lightweight process
  - Thread requires less to create and exists in the process
  - Thread shares the process resources

#### Multithreading program

- Multithreading in java is a process of executing multiple processes simultaneously.
- A program divided into two or more subprograms, which can be implemented at the same time in parallel.
- Multiprocessing and Multithreading, both are used to achieve multitasking.
- Java multithreading is mostly used in games, animation etc

#### Creating thread

- Threads are implemented in the form of objects
- The `run()` and `start()` are two built in methods which helps to thread implementation.
- The *run() method* and the *start() method* are heart and soul of any thread.
  - It makes up the entire body of a thread.
- The *run() method* can be initiated with the help of *start() method*

Thread is class found in **java.lang** package.

#### 1.8.11. AWT and SWING

Java is one of the most in-demand programming languages for developing a variety of applications. The popularity of Java can be attributed to its versatility as it can be used to design customized applications that are light and fast and serve a variety of purposes ranging from *web services to android applications*. Java is *fast, reliable, and secure*. There are multiple ways to develop GUI-based *applications* in java, out of which the most popular ones are *AWT* and *Swing*.

Before getting into the differences, let us first understand what each of them is.

#### 1. AWT

AWT stands for Abstract Window Toolkit. It is a platform-dependent API to develop GUI (Graphical User Interface) or window-based applications in Java. It was developed by heavily *Sun Microsystems* In 1995. It is heavy-weight in use because it is generated by the system's host operating system. It contains a large number of classes and methods, which are used for creating and managing GUI.



The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc. The AWT tutorial will help the user to understand Java GUI programming in simple and easy steps.

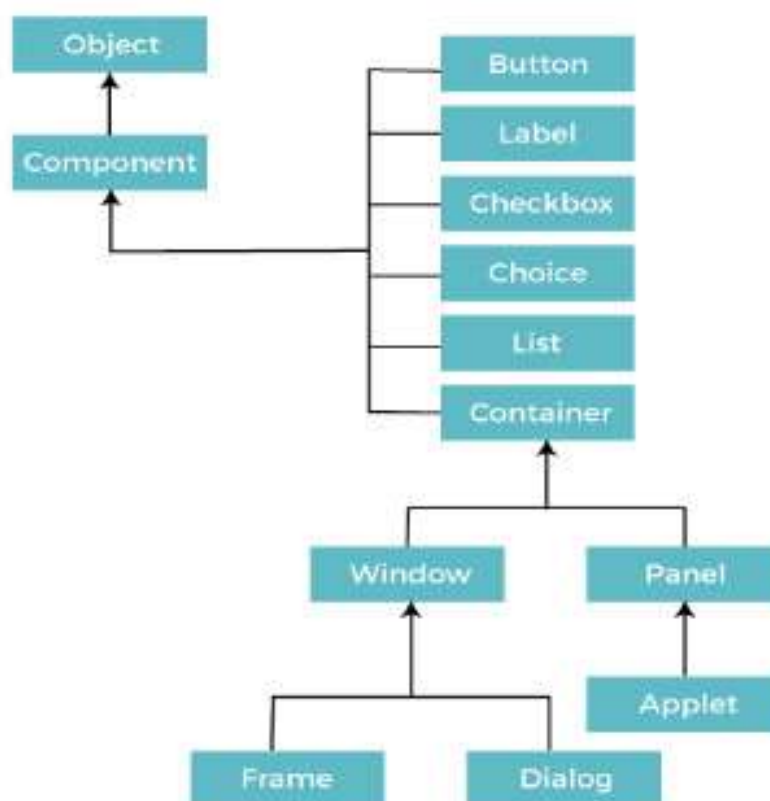
### Why AWT is platform independent?

Java AWT calls the native platform (operating systems) subroutine for creating API components like TextField, CheckBox, button, etc.

For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components. In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

### Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



## Components

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

## Container

The Container is a component in AWT that can contain another components like [buttons](#), textfields, labels etc. The classes that extends Container class are known as container such as **Frame**, **Dialog** and **Panel**.

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

**Note:** A container itself is a component (see the above diagram), therefore we can add a container inside container.

### Types of containers:

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

### Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

### Panel

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

### Frame

The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. **Frame** is most widely used container while developing an AWT application.

## Useful Methods of Component Class



Method	Description
<code>public void add(Component c)</code>	Inserts a component on this component.
<code>public void setSize(int width,int height)</code>	Sets the size (width and height) of the component.
<code>public void setLayout(LayoutManager m)</code>	Defines the layout manager for the component.
<code>public void setVisible(boolean status)</code>	Changes the visibility of the component, by default false.

### Java AWT Example

To create simple AWT example, you need a frame. There are two ways to create a GUI using Frame in AWT.

1. By extending Frame class (**inheritance**)
2. By creating the object of Frame class (**association**)

### AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

#### AWTExample1.java

```
// importing Java AWT class
import java.awt.*;

// extending Frame class to our class AWTExample1
public class AWTExample1 extends Frame {

    // initializing using constructor
    AWTExample1() {

        // creating a button

        Button b = new Button("Click Me!!!");

        // setting button position on screen

        b.setBounds(30,100,80,30);

        // adding button into frame

        add(b);
```

```

// frame size 300 width and 300 height
setSize(300,300);

// setting the title of Frame
setTitle("This is our basic AWT example");

// no layout manager
setLayout(null);

// now frame will be visible, by default it is not visible
setVisible(true);
}

// main method
public static void main(String args[]) {
    // creating instance of Frame class
    AWTEExample1 f = new AWTEExample1();
}
}

```

The `setBounds(int x-axis, int y-axis, int width, int height)` method is used in the above example that sets the position of the awt button.

Output



### AWT Example by Association

Let's see a simple example of AWT where we are creating instance of Frame class. Here, we are creating a TextField, Label and Button component on the Frame.

**AWTEExample2.java**



```
// importing Java AWT class
import java.awt.*;

// class AWTEExample2 directly creates instance of Frame class
class AWTEExample2 {

    // initializing using constructor
    AWTEExample2() {

        // creating a Frame
        Frame f = new Frame();

        // creating a Label
        Label l = new Label("Employee id:");

        // creating a Button
        Button b = new Button("Submit");
```



```
// creating a TextField
TextField t = new TextField();

// setting position of above components in the frame
l.setBounds(20, 80, 80, 30);
t.setBounds(20, 100, 80, 30);
b.setBounds(100, 100, 80, 30);

// adding components into frame
f.add(b);
f.add(l);
f.add(t);

// frame size 300 width and 300 height
f.setSize(400,300);

// setting the title of frame
f.setTitle("Employee info");
```





```
// no layout
f.setLayout(null);

// setting visibility of frame
f.setVisible(true);
}

// main method
public static void main(String args[]) {

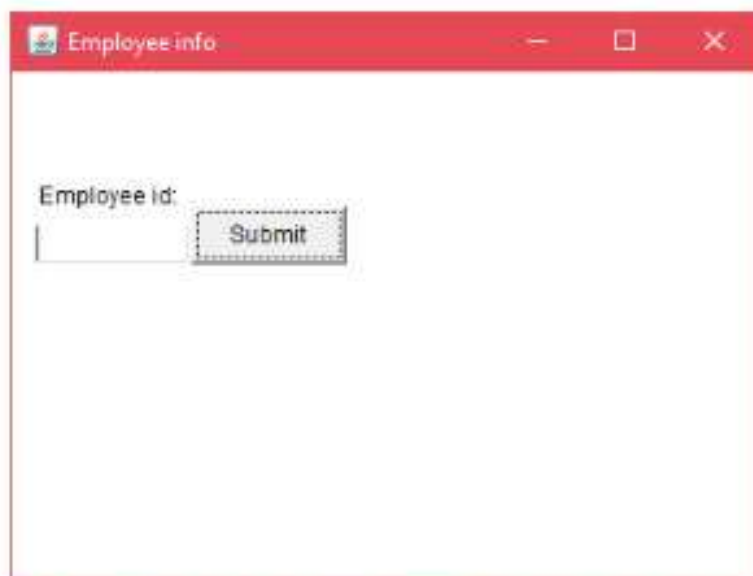
// creating instance of Frame class
AWTExample2 awt_obj = new AWTExample2();

}

}
```



## Output:



## 2. Swing:

Swing is a lightweight Java graphical user interface (GUI) that is used to create various applications. Swing has platform-independent components. It enables the user to create buttons and scroll bars. Swing includes packages for creating desktop applications in Java. Swing components are written in Java language. It is a part of Java Foundation Classes (JFC).

### SWING

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.



## Difference between AWT and SWING

No.	Java AWT	Java Swing
1)	AWT components are <b>platform-dependent</b> .	Java swing components are <b>platform-independent</b> .
2)	AWT components are <b>heavyweight</b> .	Swing components are <b>lightweight</b> .
3)	AWT <b>doesn't support pluggable look and feel</b> .	Swing <b>supports pluggable look and feel</b> .
4)	AWT provides <b>less components</b> than Swing.	Swing provides <b>more powerful components</b> such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT <b>doesn't follows MVC</b> (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing <b>follows MVC</b> .

### What is JFC?

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

### Do You Know

- How to create runnable jar file in java?
- How to display image on a button in swing?
- How to change the component color by choosing a color from Color Chooser?
- How to display the digital watch in swing tutorial?
- How to create a notepad in swing?
- How to create puzzle game and pic puzzle game in swing?
- How to create tic tac toe game in swing?

## Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.

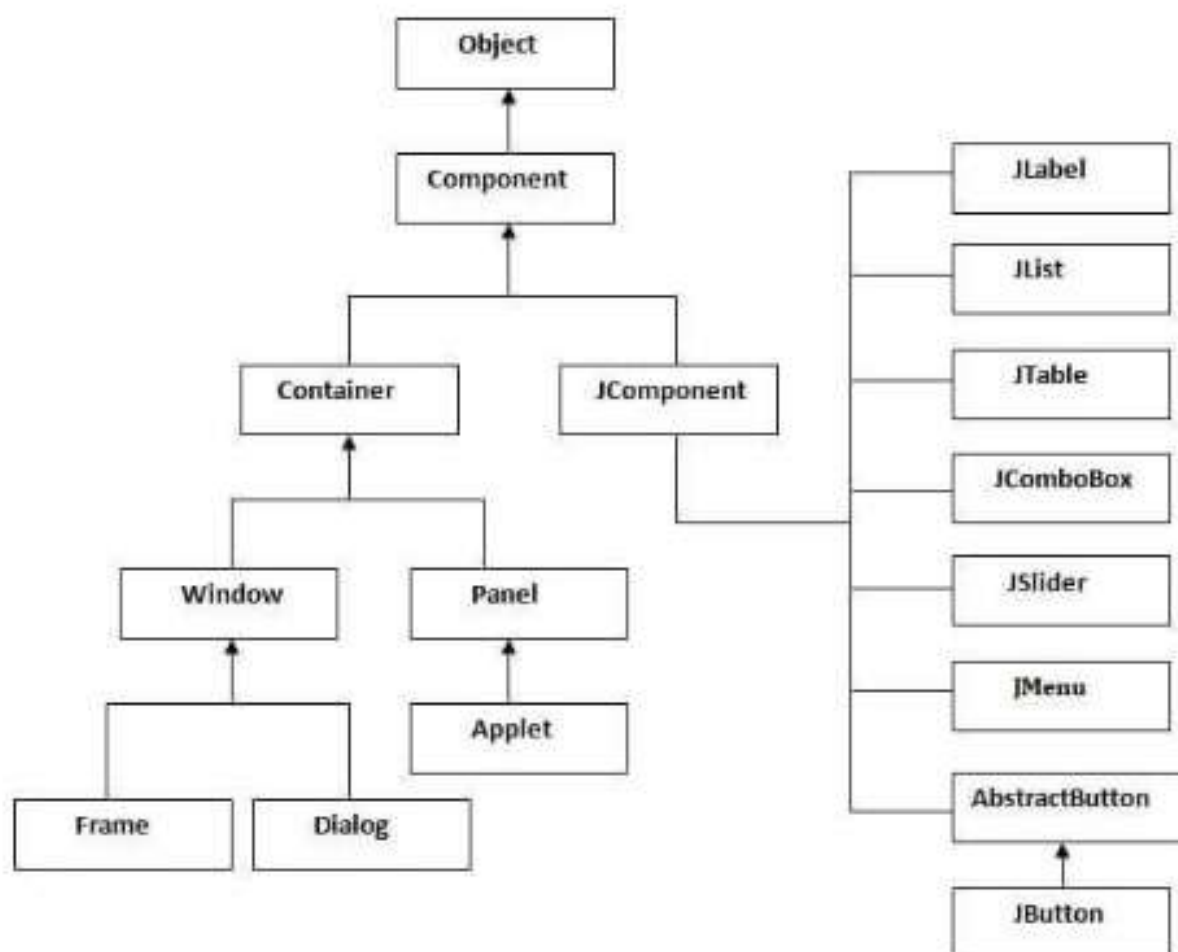


Figure: Hierarchy of Java Swing classes

### Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

### Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)



- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

### Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

*File: FirstSwingExample.java*

```
import javax.swing.*;

public class FirstSwingExample {
    public static void main(String[] args) {
        JFrame f=new JFrame();//creating instance of JFrame

        JButton b=new JButton("click");//creating instance of JButton
        b.setBounds(130,100,100, 40);//x axis, y axis, width, height

        f.add(b);//adding button in JFrame

        f.setSize(400,500);//400 width and 500 height
        f.setLayout(null);//using no layout managers
        f.setVisible(true);//making the frame visible
    }
}
```

### Output



### Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

*File: Simple.java*

```

import javax.swing.*;

public class Simple {
    JFrame f;
    Simple(){
        f=new JFrame();//creating instance of JFrame

        JButton b=new JButton("click");//creating instance of JButton
        b.setBounds(130,100,100, 40);

        f.add(b);//adding button in JFrame

        f.setSize(400,500);//400 width and 500 height
        f.setLayout(null);//using no layout managers
        f.setVisible(true);//making the frame visible
    }

    public static void main(String[] args) {
        new Simple();
    }
}

```

The `setBounds(int xaxis, int yaxis, int width, int height)` is used in the above example that sets the position of the button.

### Simple example of Swing by inheritance

We can also inherit the `JFrame` class, so there is no need to create the instance of `JFrame` class explicitly.

*File: Simple2.java*



```
import javax.swing.*;
public class Simple2 extends JFrame{//inheriting JFrame
JFrame f;
Simple2(){
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);

add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args) {
new Simple2();
}}
```

### Java JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

#### JButton class declaration

Let's see the declaration for javax.swing.JButton class.

```
public class JButton extends AbstractButton implements Accessible
```

#### Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

#### Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the <a href="#">action listener</a> to this object.

### Java JButton Example

```

import javax.swing.*;
public class ButtonExample {
    public static void main(String[] args) {
        JFrame f=new JFrame("Button Example");
        JButton b=new JButton("Click Here");
        b.setBounds(50,100,95,30);
        f.add(b);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}

```

Output





## Java JButton Example with ActionListener

```
import java.awt.event.*;
import javax.swing.*;
public class ButtonExample {
    public static void main(String[] args) {
        JFrame f=new JFrame("Button Example");
        final JTextField tf=new JTextField();
        tf.setBounds(50,50, 150,20);
        JButton b=new JButton("Click Here");
        b.setBounds(50,100,95,30);
        b.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                tf.setText("Welcome to Javatpoint.");
            }
        });
        f.add(b);f.add(tf);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

## Output



#### 1.8.12. What is Servlet?

Basically, a java program that runs on the server. It is used for developing dynamic web applications. Servlet technology is used to create web application (resides at server-side and generates dynamic web pages). Servlet technology is robust and scalable because of java language. There are many interfaces and classes in servlet API such as *Servlet*, *GenericServlet*, *HttpServlet*, *HttpRequest*, *HttpResponse* etc. Servlets are useful in many business-oriented websites such as Amazon and eBay



## Summary multiple choice questions (Computer Programming)

1. \_\_\_\_\_ refers to the process of locating and removing the errors in a program
  - A. Analyzing
  - B. Correcting
  - C. Debugging
  - D. Executing
2. Who invented C++?
  - A. Dennis Ritchie
  - B. Ken Thompson
  - C. Brian Kernighan
  - D. Bjarne Stroustrup
3. What is C++?
  - A. C++ is an object-oriented programming language
  - B. C++ is a procedural programming language
  - C. C++ supports both procedural and object-oriented programming language
  - D. C++ is a functional programming language
4. Which of the following is the correct syntax of including a user defined header files in C++?
  - A. #include [userdefined]
  - B. #include "userdefined"
  - C. #include <userdefined.h>
  - D. #include <userdefined>
5. Which of the following is used for comments in C++?
  - A. /\* comment \*/
  - B. // comment \*/
  - C. // comment
  - D. both // comment or /\* comment \*/
6. The feature that allows the same operations to be carried out differently depending on the object is \_\_
  - A. Polymorphism
  - B. Polygamy
  - C. Inheritane
  - D. Multitasking

7. Which of the following user-defined header file extension used in c++?

- A. hg
- B. cpp
- C. h
- D. hf

8. Variable declaration defines

- A. only the name of the variable
- B. an address in the memory
- C. the type of data the variable can hold
- D. both a) and c)

9. Which of the following is correct identifier in C++?

- A. VAR\_1234
- B. \$var\_name
- C. 7VARNAME
- D. 7var\_name

10. Which of the following correctly declares an array in C++?

- A. array{10};
- B. array array[10];
- C. int array;
- D. int array[10];

11. Which is more effective while calling the C++ functions?

- A. call by object
- B. call by pointer
- C. call by value
- D. call by reference

12. Who invented Java Programming?

- A. Guido van Rossum
- B. James Gosling
- C. Dennis Ritchie
- D. Bjarne Structure

13. Methods can be added to objects

- A. to move objects
- B. to define behavior
- C. to rotate objects
- D. to turn objects



14. Which statement is true about Java?

- A. Java is a sequence-dependent programming language
- B. Java is a code dependent programming language
- C. Java is a platform-dependent programming language
- D. Java is a platform-independent programming language

15. Which component is used to compile, debug and execute the java programs?

- A. JRE
- B. JIT
- C. JDK
- D. JVM

16. Which of the following cannot be part of a function declaration?

- A. Function name
- B. Function return type
- C. Function parameter
- D. Function definition

17. Which one of the following is not a Java feature?

- A. Object-oriented
- B. Use of pointers
- C. Portable
- D. Dynamic and Extensible

18. Which of these cannot be used for a variable name in Java?

- A. identifier & keyword
- B. identifier
- C. keyword
- D. none of the mentioned

19. What will be the output of the following Java code?

```
1. class increment {  
2.     public static void main(String args[])  
3.     {  
4.         int g = 3;  
5.         System.out.print(++g * 8);  
6.     }  
7. }
```

- A. 32
- B. 33
- C. 24
- D. 25

20. You have declared an integer pointer called point You have also declared an integer called number. Which statement is the correct format?
- A. point = number;
  - B. point = \*number;
  - C. point = &number;
  - D. point = +number;
21. Which of the following is not an Object-Oriented programming concept in Java?
- A. Polymorphism
  - B. Inheritance
  - C. Compilation
  - D. Encapsulation
22. What is not the use of "this" keyword in Java?
- A. Referring to the instance variable when a local variable has the same name
  - B. Passing itself to the method of the same class
  - C. Passing itself to another method
  - D. Calling another constructor in constructor chaining
23. Which of the following is a type of polymorphism in Java Programming?
- A. Multiple polymorphism
  - B. Compile time polymorphism
  - C. Multilevel polymorphism
  - D. Execution time polymorphism
24. What is Truncation in Java?
- A. Floating-point value assigned to a Floating type
  - B. Floating-point value assigned to an integer type
  - C. Integer value assigned to floating type
  - D. Integer value assigned to floating type
25. What will be the output of the following Java program?

```
1.  class Output
2.  {
3.      public static void main(String args[])
4.      {
5.          int arr[] = {1, 2, 3, 4, 5};
6.          for ( int i = 0; i < arr.length - 2; ++i)
7.              System.out.println(arr[i] + " ");
8.      }
9.  }
```



- A. 1 2 3 4 5
- B. 1 2 3 4
- C. 1 2
- D. 1 2 3

26. What is the extension of compiled java classes?

- A. .txt
- B. .js
- C. .class
- D. .java

27. What will be the output of the following Java program?

```
1.  class Output
2.  {
3.      public static void main(String args[])
4.      {
5.          double x = 2.0;
6.          double y = 3.0;
7.          double z = Math.pow( x, y );
8.          System.out.print(z);
9.      }
10. }
```

- A. 9.0
- B. 8.0
- C. 4.0
- D. 2.0

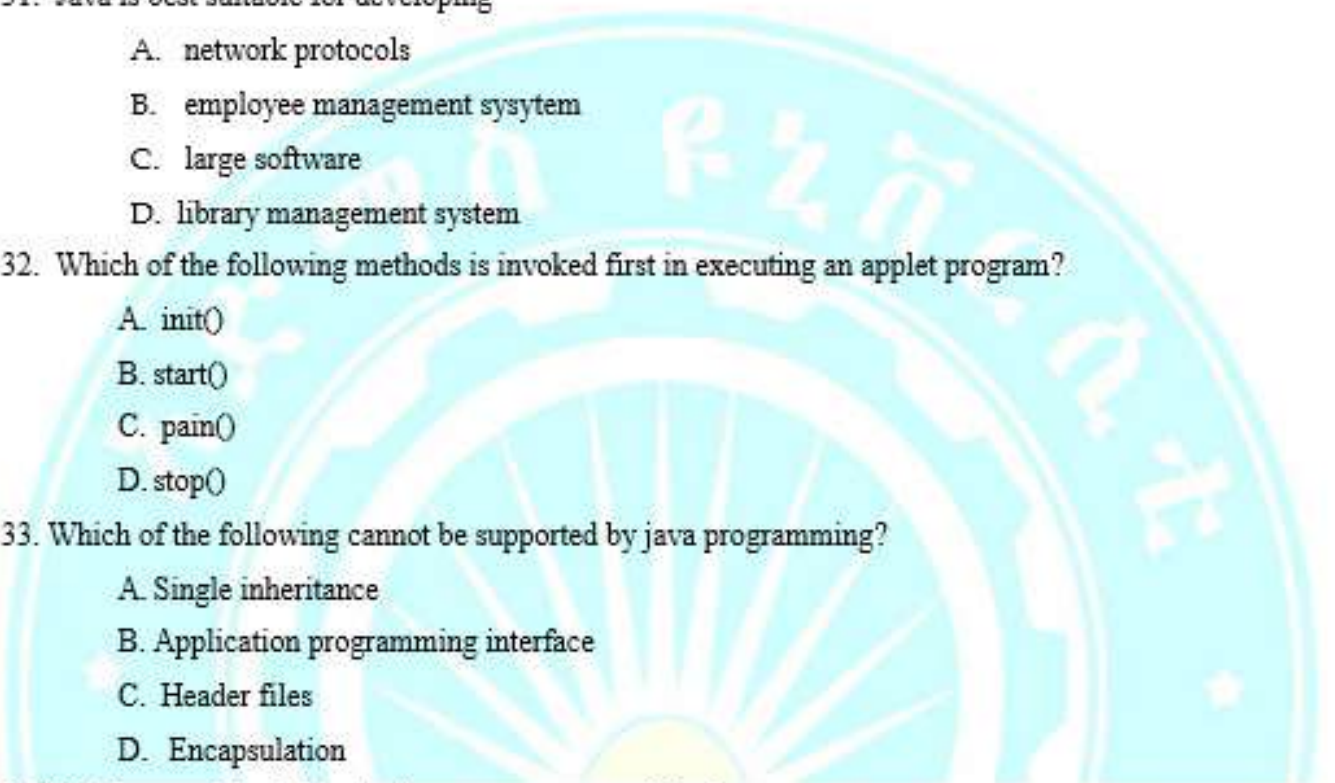
28. Which of these statements is incorrect about Thread?

- A. start () method is used to begin execution of the thread
- B. run() method is used to begin execution of a thread before start() method in special cases
- C. A thread can be formed by implementing Runnable interface only
- D. A thread can be formed by a class that extends Thread class

29. Which of these keywords are used for the block to be examined for exceptions?

- A. check
- B. throw
- C. catch
- D. try

30. An applet program can

- 
- A. draw pictures on web page  
B. play sounds, videos  
C. create new window and draw in it  
D. all of the above
31. Java is best suitable for developing
- A. network protocols  
B. employee management system  
C. large software  
D. library management system
32. Which of the following methods is invoked first in executing an applet program?
- A. init()  
B. start()  
C. paint()  
D. stop()
33. Which of the following cannot be supported by java programming?
- A. Single inheritance  
B. Application programming interface  
C. Header files  
D. Encapsulation
34. Which one of the following is not an access modifier?
- A. Protected  
B. Void  
C. Public  
D. Private
35. Which class provides system independent server-side implementation?
- A. Server  
B. ServerReader  
C. Socket  
D. ServerSocket
36. Which of the following is true about servlets?
- A. Servlets can use the full functionality of the Java class libraries  
B. Servlets execute within the address space of web server, platform independent and uses the functionality of java class libraries



- C. Servlets execute within the address space of web server
- D. Servlets are platform-independent because they are written in java

37. Which one acts as communication endpoint between applications?

- A. IP Address
- B. Port number
- C. Socket
- D. Uniform Resource Locator

38. Give the abbreviation of AWT?

- A. Applet Windowing Toolkit
- B. Abstract Windowing Toolkit
- C. Absolute Windowing Toolkit
- D. None of the above

39. Which of the following names must start always in upper case?

- A. Object name
- B. Class name
- C. Variable name
- D. Method name

40. The output of the following Java program will be

```
class Test {  
    private static int x;  
    public static void main(String args[]) {  
        System.out.println(fun());  
    }  
    static int fun() {  
        return ++x;  
    }  
}
```

B. 2

D. 3

41. Give the abbreviation of AWT?

- A. Applet Windowing Toolkit
- B. Abstract Windowing Toolkit
- C. Absolute Windowing Toolkit
- D. None of the above

42. Which of these class is used to encapsulate IP address and DNS??

- A. DatagramPacket
- B. URL
- C. InetAddress
- D. ContentHandler

43. What is multithreaded programming?

- A. It's a process in which two different processes run simultaneously
  - B. It's a process in which two or more parts of same process run simultaneously
  - C. It's a process in which many different processes are able to access same information
  - D. It's a process in which a single process can access information from many sources
44. Which of the following protocol follows connection less service?
- A. TCP
  - B. TCP/IP
  - C. UDP
  - D. HTTP
45. Which of the following hosts Java web applications?
- A. Tomcat
  - B. Eclipse
  - C. NetBeans
  - D. JDK
46. Thread priority in Java is?
- A. Integer
  - B. Float
  - C. Double
  - D. Long
47. Which of the following is not true?
- A. Passing by value, the argument is read-only
  - B. Passing by reference, the parameter is a local variable
  - C. The argument passed by value may be a constant or a variable
  - D. The argument passed by reference must be a variable
48. Inheritance is the principle that
- A. Classes with the same name must be derived from one another
  - B. Knowledge of a general category can be applied to more specific objects
  - C. C++ functions may be used only if they have logical predecessors
  - D. One function name may invoke different methods
49. A function that is called automatically each time an object is created is a(n)
- A. Constructor
  - B. Contractor
  - C. Builder
  - D. Architect
50. An expression contains relational, assignment and arithmetic operators. In the absence of parentheses, the order of evaluation will be



- A. Assignment, relational, arithmetic
- B. Arithmetic, relational, assignment
- C. Relational, arithmetic, assignment
- D. Assignment, arithmetic, relational

### Answer for Computer Programming questions

1.C	6.A	11.D	16.D	21.C	26.C	31.A	36.B	41.C	46.C
2.D	7.C	12.B	17.B	22.B	27.B	32.A	37.C	42.B	47.B
3.C	8.D	13.B	18.C	23.B	28.B	33.C	38.B	43.C	48.A
4.D	9.A	14.D	19.A	24.B	29.D	34.B	39.C	44.A	49.B
5.D	10.D	15.C	20.C	25.D	30.D	35.D	40.B	45.A	50.C

## Part II: Data Structure and Algorithms, Theory of Algorithms

### Module objective

At the end of this module, students will be able to:

- Introduce the most common data structures like stack, queue, linked list
- Give alternate methods of data organization and representation
- Enable students use the concepts related to Data Structures and Algorithms to solve real world problems
- Practice Recursion, Sorting, and searching on the different data structures
- Implement the data structures with a chosen programming language

### 2.1. Data Structures and Algorithms Analysis

#### 2.1.1. Introduction to Data Structures and Algorithms Analysis

A program is written in order to solve a problem. A solution to a problem actually consists of two things:

- A way to organize the data
- Sequence of steps to solve the problem

The way data are organized in a computer memory is said to be Data Structure and the sequence of computational steps to solve a problem is said to be an algorithm. Therefore, a program is nothing but data structures plus algorithms.

#### Introduction to Data Structures

Given a problem, the first step to solve the problem is obtaining one's own abstract view, or *model*, of the problem. This process of modeling is called *abstraction*. The model defines an abstract view to the problem. This implies that the model focuses only on problem related stuff and that a programmer tries to define the *properties* of the problem.

These properties include

- The *data* which are affected and
- The *operations* that are involved in the problem.

With abstraction you create a well-defined entity that can be properly handled. These entities define the *data structure* of the program. An entity with the properties just described is called an *abstract data type* (ADT).

#### Abstract Data Types

An ADT consists of an abstract data structure and operations. Put in other terms, an ADT is an abstraction of a data structure.



The ADT specifies:

1. What can be stored in the Abstract Data Type?
2. What operations can be done on/by the Abstract Data Type?

For example, if we are going to model employees of an organization:

- This ADT stores employees with their relevant attributes and discarding irrelevant attributes.
- This ADT supports hiring, firing, retiring, ... operations.

A data structure is a language construct that the programmer has defined in order to implement an abstract data type. There are lots of formalized and standard Abstract data types such as Stacks, Queues, Trees, etc.

Do all characteristics need to be modeled?

Not at all

- It depends on the scope of the model
- It depends on the reason for developing the model

### **Abstraction**

Abstraction is a process of classifying characteristics as relevant and irrelevant for the particular purpose at hand and ignoring the irrelevant ones. Applying abstraction correctly is the essence of successful programming

### **Algorithms**

An algorithm is a well-defined computational procedure that takes some value or a set of values as input and produces some value or a set of values as output. Data structures model the static part of the world. Properties of algorithm includes: Finiteness, Definiteness, Sequence, Feasibility, Correctness, Language Independence, Completeness, Effectiveness, Efficiency, Generality, and Input/Output.

### **Algorithm Analysis Concepts**

Algorithm analysis refers to the process of determining the amount of computing time and storage space required by different algorithms. In other words, it's a process of predicting the resource requirement of algorithms in a given environment.

### **Complexity Analysis**

Complexity Analysis is the systematic study of the cost of computation, measured either in time units or in operations performed, or in the amount of storage space required.

The goal is to have a meaningful measure that permits comparison of algorithms independent of operating platform.

There are two things to consider:

- **Time Complexity:** Determine the approximate number of operations required to solve a problem of size  $n$ .
- **Space Complexity:** Determine the approximate memory required to solve a problem of size  $n$ .

### Measures of Times

In order to determine the running time of an algorithm it is possible to define three functions  $T_{best}(n)$ ,  $T_{avg}(n)$  and  $T_{worst}(n)$  as the best, the average and the worst case running time of the algorithm respectively.

Average Case ( $T_{avg}$ ): The amount of time the algorithm takes on an "average" set of inputs.

Worst Case ( $T_{worst}$ ): The amount of time the algorithm takes on the worst possible set of inputs.

Best Case ( $T_{best}$ ): The amount of time the algorithm takes on the smallest possible set of inputs.

We are interested in the worst-case time, since it provides a bound for all input – this is called the “Big-Oh” estimate.

### Asymptotic Analysis

Asymptotic analysis is concerned with how the running time of an algorithm increases with the size of the input in the limit, as the size of the input increases without bound.

There are five notations used to describe a running time function. These are:

- Big-Oh Notation ( $O$ )
- Big-Omega Notation ( $\Omega$ )
- Theta Notation ( $\Theta$ )
- Little-o Notation ( $o$ )
- Little-Omega Notation ( $\omega$ )



#### 2.1.2. Simple Sorting and Searching Algorithms

##### Searching

Searching is a process of looking for a specific element in a list of items or determining that the item is not in the list. There are two simple searching algorithms:

- Sequential Search, and
- Binary Search

##### Linear Search (Sequential Search)



### Pseudocode

Loop through the array starting at the first element until the value of target matches one of the array elements.

If a match is not found, return -1.

Time is proportional to the size of input ( $n$ ) and we call this time complexity  $O(n)$ .

### Binary Search

This searching algorithm works only on an ordered list.

The basic idea is:

- Locate midpoint of array to search
- Determine if target is in lower half or upper half of an array.
  - If in lower half, make this half the array to search
  - If in the upper half, make this half the array to search
- Loop back to step 1 until the size of the array to search is one, and this element does not match, in which case return -1.

The computational time for this algorithm is proportional to  $\log_2 n$ . Therefore the time complexity is  $O(\log n)$ .

### Sorting Algorithms

Sorting is one of the most important operations performed by computers. Sorting is a process of reordering a list of items in either increasing or decreasing order. The following are simple sorting algorithms used to sort small-sized lists.

- Insertion Sort
- Selection Sort
- Bubble Sort

## Insertion Sort

The insertion sort works just like its name suggests - it inserts each item into its proper place in the final list. The simplest implementation of this requires two list structures - the source list and the list into which sorted items are inserted. To save memory, most implementations use an in-place sort that works by moving the current item past the already sorted items and repeatedly swapping it with the preceding item until it is in place.

It's the most instinctive type of sorting algorithm. The approach is the same approach that you use for sorting a set of cards in your hand. While playing cards, you pick up a card, start at the beginning of your hand and find the place to insert the new card, insert it and move all the others up one place.

### Basic Idea:

Find the location for an element and move all others up, and insert the element.

The process involved in insertion sort is as follows:

1. The left most value can be said to be sorted relative to itself. Thus, we don't need to do anything.
2. Check to see if the second value is smaller than the first one. If it is, swap these two values. The first two values are now relatively sorted.
3. Next, we need to insert the third value in to the relatively sorted portion so that after insertion, the portion will still be relatively sorted.
4. Remove the third value first. Slide the second value to make room for insertion. Insert the value in the appropriate position.
5. Now the first three are relatively sorted.
6. Do the same for the remaining items in the list.

## Selection Sort

### Basic Idea:

- Loop through the array from  $i=0$  to  $n-1$ .
- Select the smallest element in the array from  $i$  to  $n$
- Swap this value with value at position  $i$ .



## Bubble Sort

Bubble sort is the simplest algorithm to implement and the slowest algorithm on very large inputs.

### Basic Idea:

- Loop through array from  $i=0$  to  $n$  and swap adjacent elements if they are out of order.

## Data Structures

### Structures

Structures are aggregate data types built using elements of primitive data types.

Structure are defined using the struct keyword:

```
E.g. struct Time{  
    int hour;  
    int minute;  
    int second;  
};
```

The struct keyword creates a new user defined data type that is used to declare variables of an aggregate data type.

Structure variables are declared like variables of other types.

**Syntax:** struct <structure tag> <variable name>;

```
E.g. struct Time timeObject,  
      struct Time *timeptr;
```

### 2.1.3. Accessing Members of Structure Variables

*The Dot operator (.)*: to access data members of structure variables.

*The Arrow operator (->)*: to access data members of pointer variables pointing to the structure.

E.g. Print member hour of timeObject and timeptr.

```
cout<< timeObject.hour; or  
cout<<timeptr->hour;
```

**TIP:** timeptr->hour is the same as (\*timeptr).hour.

The parentheses is required since (\*) has lower precedence than (.).

### 2.1.4. Singly Linked Lists

Linked lists are the most basic self-referential structures. Linked lists allow you to have a chain of structs with related data.

## Array vs. Linked lists

Arrays are *simple* and *fast* but *we must* specify their size at construction time. This has its own drawbacks. If you construct an array with space for  $n$ , tomorrow you may need  $n+1$ . Here comes a need for a more flexible system.

## Advantages of Linked Lists

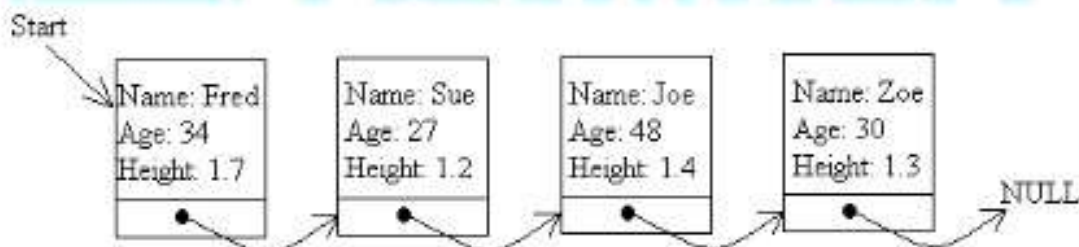
Flexible space use by dynamically allocating space for each element as needed. This implies that one need not know the size of the list in advance. Memory is efficiently utilized.

A linked list is made up of a chain of nodes. Each **node** contains:

- the data item, and
- a pointer to the next node

## Creating Linked Lists in C++

A linked list is a data structure that is built from structures and pointers. It forms a chain of "nodes" with pointers representing the links of the chain and holding the entire thing together. A linked list can be represented by a diagram like this one:



This linked list has four nodes in it, each with a link to the next node in the series. The last node has a link to the special value NULL, which any pointer (whatever its type) can point to, to show that it is the last link in the chain. There is also another special pointer, called Start (also called head), which points to the first link in the chain so that we can keep track of it.

## Defining the data structure for a linked list

The key part of a linked list is a structure, which holds the data for each node (the name, address, age or whatever for the items in the list), and, most importantly, a pointer to the next node. Here we have given the structure of a typical node:

```
struct node
{
    char name[20];    // Name of up to 20 letters
    int age;
    float height;     // In metres
    // pointer to next node
```



```

        node *nxt;// Pointer to next node
    };
    struct node *start_ptr = NULL;

```

The important part of the structure is the line before the closing curly brackets. This gives a pointer to the next node in the list. This is the only case in C++ where you are allowed to refer to a data type (in this case `node`) before you have even finished defining it!

We have also declared a pointer called `start_ptr` that will permanently point to the start of the list. To start with, there are no nodes in the list, which is why `start_ptr` is set to `NULL`.

### Adding a node to the list

The first problem that we face is how to add a node to the list. For simplicity's sake, we will assume that it has to be added to the end of the list, although it could be added anywhere in the list (a problem we will deal with later on).

Firstly, we declare the space for a pointer item and assign a temporary pointer to it. This is done using the `new` statement as follows:

```
temp = new node;
```



We can refer to the new node as `*temp`, i.e. "the node that `temp` points to". When the fields of this structure are referred to, brackets can be put round the `*temp` part, as otherwise the compiler will think we are trying to refer to the fields of the pointer. Alternatively, we can use the arrow pointer notation. That's what we shall do here.

Having declared the node, we ask the user to fill in the details of the person, i.e. the name, age, address or whatever:

```

cout << "Please enter the name of the person: ";
cin >> temp->name;
cout << "Please enter the age of the person : ";
cin >> temp->age;
cout << "Please enter the height of the person : ";

```

```

cin >> temp->height;
temp->nxt = NULL;

```

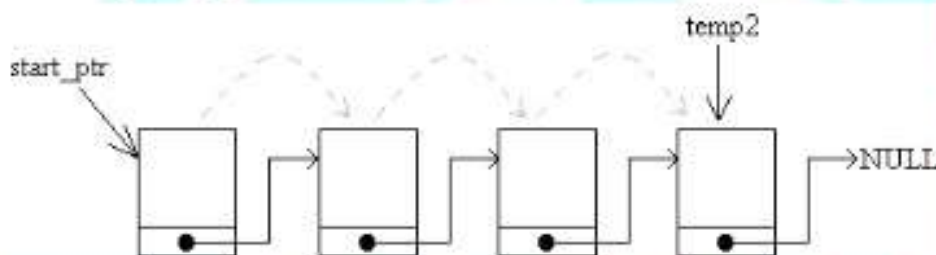
The last line sets the pointer from this node to the next to NULL, indicating that this node, when it is inserted in the list, will be the last node. Having set up the information, we have to decide what to do with the pointers. Of course, if the list is empty to start with, there's no problem - just set the Start pointer to point to this node (i.e. set it to the same value as temp):

```

if (start_ptr == NULL)
    start_ptr = temp;

```

It is harder if there are already nodes in the list. In this case, the secret is to declare a second pointer, **temp2**, to step through the list until it finds the last node.



```

temp2 = start_ptr;
// We know this is not NULL - list not empty!
while (temp2->nxt != NULL) {
    temp2 = temp2->nxt; // Move to next link in chain
}

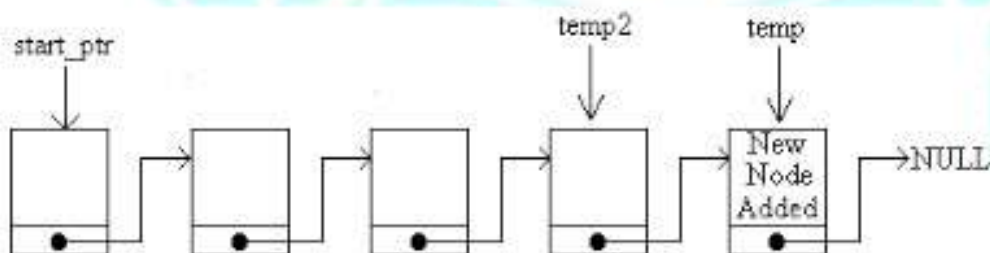
```

The loop will terminate when **temp2** points to the last node in the chain, and it knows when this happened because the **nxt** pointer in that node will point to NULL. When it has found it, it sets the pointer from that last node to point to the node we have just declared:

```

temp2->nxt = temp;

```



The link **temp2->nxt** in this diagram is the link joining the last two nodes. The full code for adding a node at the end of the list is shown below, in its own little function:

```

void add_node_at_end ()
{
    node *temp, *temp2; // Temporary pointers

```



```

// Reserve space for new node and fill it with data
temp = new node;
cout << "Please enter the name of the person: ";
cin >> temp->name;
cout << "Please enter the age of the person : ";
cin >> temp->age;
cout << "Please enter the height of the person : ";
cin >> temp->height;
temp->nxt = NULL;

// Set up link to this node
if (start_ptr == NULL)
    start_ptr = temp;
else
    { temp2 = start_ptr;
// We know this is not NULL - list not empty!
    while (temp2->nxt != NULL)
        { temp2 = temp2->nxt;
// Move to next link in chain
        }
    temp2->nxt = temp;
    }
}

```

### Displaying the list of nodes

Having added one or more nodes, we need to display the list of nodes on the screen. This is comparatively easy to do. Here is the method:

1. Set a temporary pointer to point to the same thing as the start pointer.
2. If the pointer points to NULL, display the message "End of list" and stop.
3. Otherwise, display the details of the node pointed to by the start pointer.
4. Make the temporary pointer point to the same thing as the **nxt** pointer of the node it is currently indicating.
5. Jump back to step 2.

The temporary pointer moves along the list, displaying the details of the nodes it comes across. At each stage, it can get hold of the next node in the list by using the `nxt` pointer of the node it is currently pointing to. Here is the C++ code that does the job:

```
temp = start_ptr;
do
{
    if (temp == NULL)
        cout << "End of list" << endl;
    else{
        // Display details for what temp points to
        cout << "Name : " << temp->name << endl;
        cout << "Age : " << temp->age << endl;
        cout << "Height : " << temp->height << endl;
        cout << endl;          // Blank line

        // Move to next node (if present)
        temp = temp->nxt;
    }
} while (temp != NULL);
```

Check through this code, matching it to the method listed above. It helps if you draw a diagram on paper of a linked list and work through the code using the diagram.

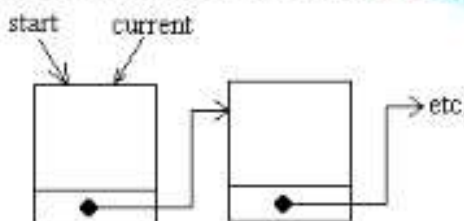
### Navigating through the list

One thing you may need to do is to navigate through the list, with a pointer that moves backwards and forwards through the list, like an index pointer in an array. This is certainly necessary when you want to insert or delete a node from somewhere inside the list, as you will need to specify the position.

We will call the mobile pointer `current`. First of all, it is declared, and set to the same value as the `start_ptr` pointer:

```
node *current;
current = start_ptr;
```

Notice that you don't need to set `current` equal to the *address* of the start pointer, as they are both pointers. The statement above makes them both point to the same thing:





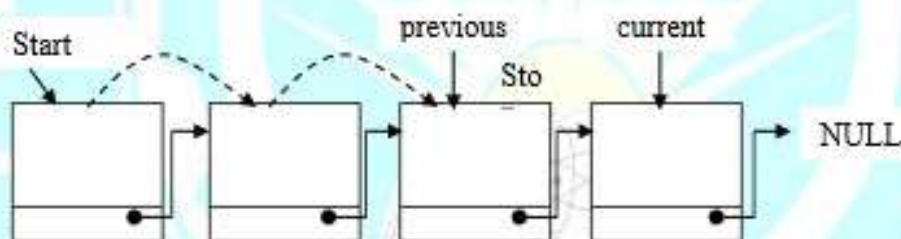
It's easy to get the current pointer to point to the next node in the list (i.e. move from left to right along the list). If you want to move current along one node, use the `nxt` field of the node that it is pointing to at the moment:

```
current = current->nxt;
```

In fact, we had better check that it isn't pointing to the last item in the list. If it is, then there is no next node to move to:

```
if (current->nxt == NULL)
    cout << "You are at the end of the list." << endl;
else
    current = current->nxt;
```

Moving the current pointer back one step is a little harder. This is because we have no way of moving back a step automatically from the current node. The only way to find the node before the current one is to start at the beginning, work our way through and stop when we find the node before the one we are considering at the moment. We can tell when this happens, as the `nxt` pointer from that node will point to exactly the same place in memory as the current pointer (i.e. the current node).



First of all, we had better check to see if the current node is also first the one. If it is, then there is no "previous" node to point to. If not, check through all the nodes in turn until we detect that we are just behind the current one (Like a pantomime - "behind you!")

```
if (current == start_ptr)
    cout << "You are at the start of the list" << endl;
else
{
    node *previous;    // Declare the pointer
    previous = start_ptr;
    while (previous->nxt != current)
    {
        previous = previous->nxt;
    }
    current = previous;
}
```

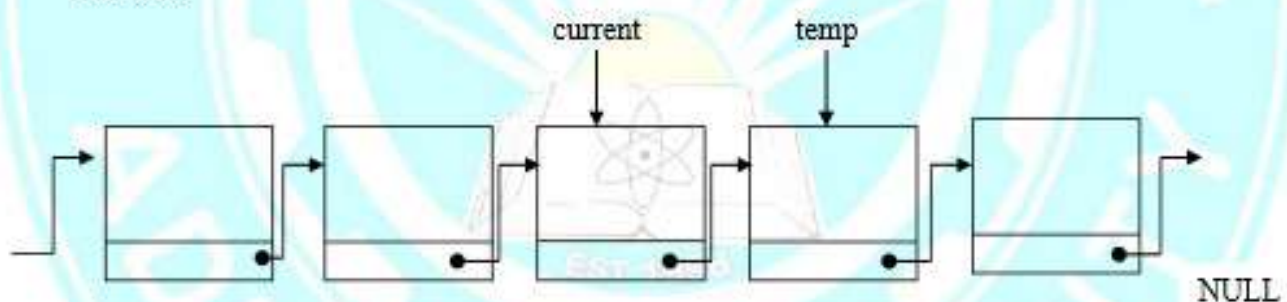
The else clause translates as follows: Declare a temporary pointer (for use in this else clause only). Set it equal to the start pointer. All the time that it is not pointing to the node before the current node, move it along the line. Once the previous node has been found, the current pointer is set to that node - i.e. it moves back along the list.

Now that you have the facility to move back and forth, you need to do something with it. Firstly, let's see if we can alter the details for that particular node in the list:

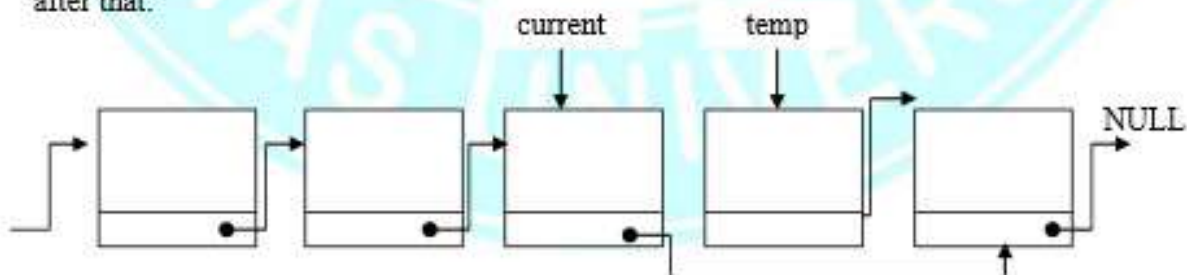
```
cout << "Please enter the new name of the person: ";
cin >> current->name;
cout << "Please enter the new age of the person : ";
cin >> current->age;
cout << "Please enter the new height of the person : ";
cin >> current->height;
```

The next easiest thing to do is to delete a node from the list directly after the current position. We have to use a temporary pointer to point to the node to be deleted. Once this node has been "anchored", the pointers to the remaining nodes can be readjusted before the node on death row is deleted. Here is the sequence of actions:

1. Firstly, the temporary pointer is assigned to the node after the current one. This is the node to be deleted:



2. Now the pointer from the current node is made to leap-frog the next node and point to the one after that:



3. The last step is to delete the node pointed to by temp.

Here is the code for deleting the node. It includes a test at the start to test whether the current node is the last one in the list:



```

if (current->nxt == NULL)
    cout << "There is no node after current" << endl;
else
    { node *temp;
      temp = current->nxt;
      current->nxt = temp->nxt;    // Could be NULL
      delete temp;
    }

```

Here is the code to **add** a node after the current one. This is done similarly, but we haven't illustrated it with diagrams:

```

if (current->nxt == NULL)
    add_node_at_end();
else
    { node *temp;
      new temp;
      get_details(temp);
      // Make the new node point to the same thing as
      // the current node
      temp->nxt = current->nxt;
      // Make the current node point to the new link
      // in the chain
      current->nxt = temp;
    }

```

We have assumed that the function `add_node_at_end()` is the routine for adding the node to the end of the list that we created near the top of this section. This routine is called if the current pointer is the last one in the list so the new one would be added on to the end.

Similarly, the routine `get_temp(temp)` is a routine that reads in the details for the new node similar to the one defined just above.

### Deleting a node from the list

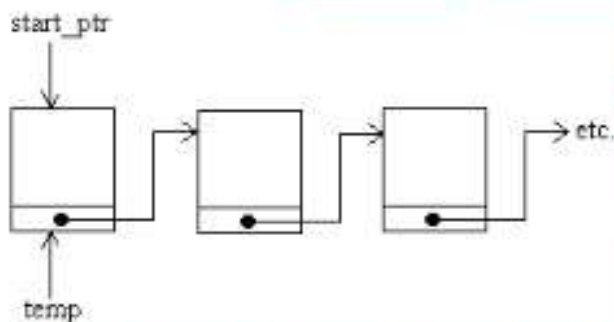
When it comes to deleting nodes, we have three choices: Delete a node from the start of the list, delete one from the end of the list, or delete one from somewhere in the middle. For simplicity, we shall just deal with deleting one from the start or from the end.

When a node is deleted, the space that it took up should be reclaimed. Otherwise the computer will eventually run out of memory space. This is done with the **delete** instruction:

```
delete temp; // Release the memory pointed to by temp
```

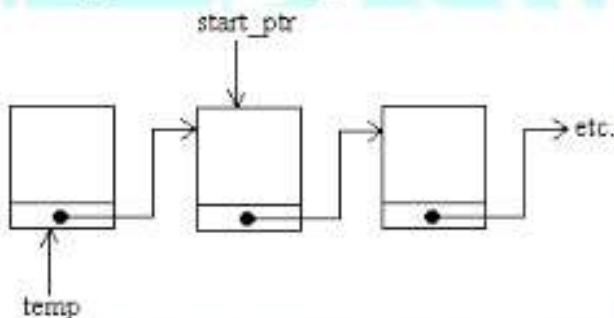
However, we can't just delete the nodes willy-nilly as it would break the chain. We need to reassign the pointers and then delete the node at the last moment. Here is how we go about deleting the first node in the linked list:

```
temp = start_ptr; // Make the temporary pointer  
                  // identical to the start pointer
```

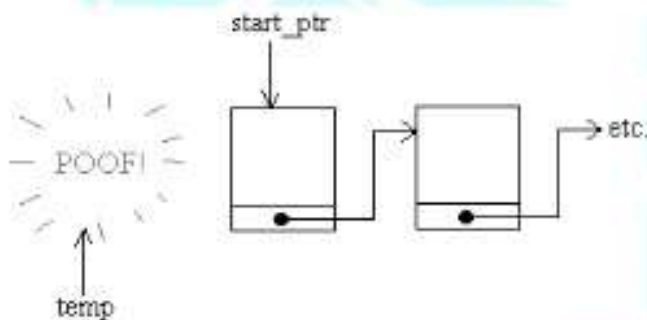


Now that the first node has been safely tagged (so that we can refer to it even when the start pointer has been reassigned), we can move the start pointer to the next node in the chain:

```
start_ptr = start_ptr->nxt; // Second node in chain.
```



```
delete temp; // Wipe out original start node
```



Here is the function that deletes a node from the start:

```
void delete_start_node()  
{ node *temp;  
  temp = start_ptr;
```



```

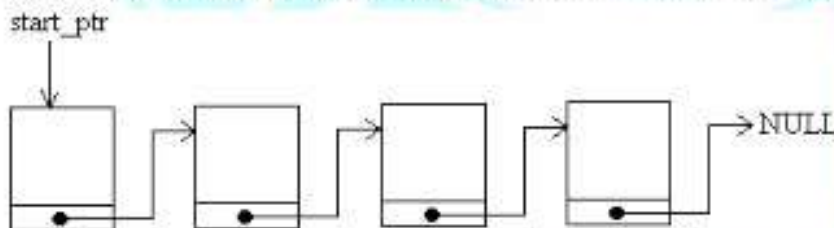
    start_ptr = start_ptr->nxt;
    delete temp;
}

```

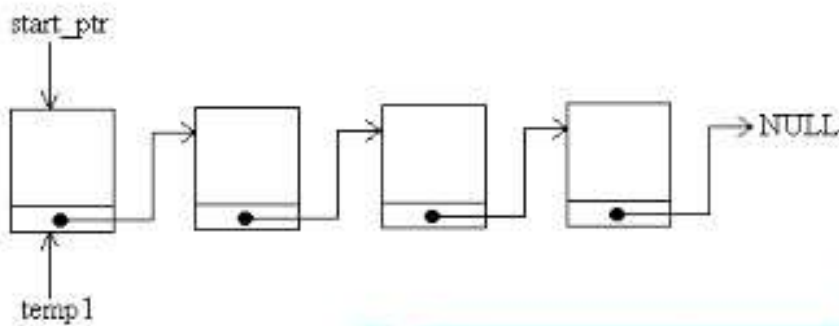
Deleting a node from the end of the list is harder, as the temporary pointer must find where the end of the list is by hopping along from the start. This is done using code that is almost identical to that used to insert a node at the end of the list. It is necessary to maintain two temporary pointers, `temp1` and `temp2`. The pointer `temp1` will point to the last node in the list and `temp2` will point to the previous node. We have to keep track of both as it is necessary to delete the last node and immediately afterwards, to set the `nxt` pointer of the previous node to NULL (it is now the new last node).

1. Look at the start pointer. If it is NULL, then the list is empty, so print out a "No nodes to delete" message.
2. Make `temp1` point to whatever the start pointer is pointing to.
3. If the `nxt` pointer of what `temp1` indicates is NULL, then we've found the last node of the list, so jump to step 7.
4. Make another pointer, `temp2`, point to the current node in the list.
5. Make `temp1` point to the next item in the list.
6. Go to step 3.
7. If you get this far, then the temporary pointer, `temp1`, should point to the last item in the list and the other temporary pointer, `temp2`, should point to the last-but-one item.
8. Delete the node pointed to by `temp1`.
9. Mark the `nxt` pointer of the node pointed to by `temp2` as NULL - it is the new last node.

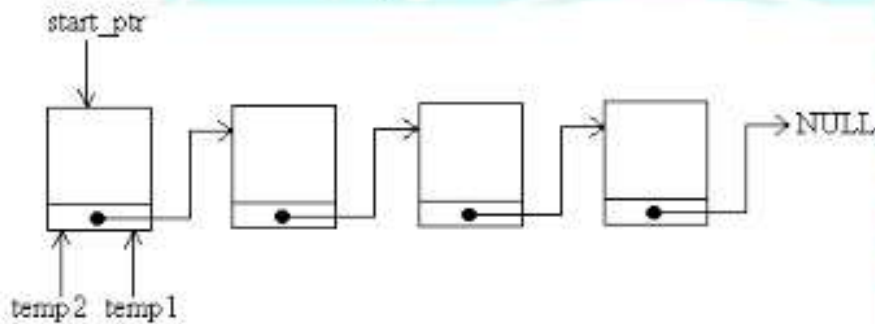
Let's try it with a rough drawing. This is always a good idea when you are trying to understand an abstract data type. Suppose we want to delete the last node from this list:



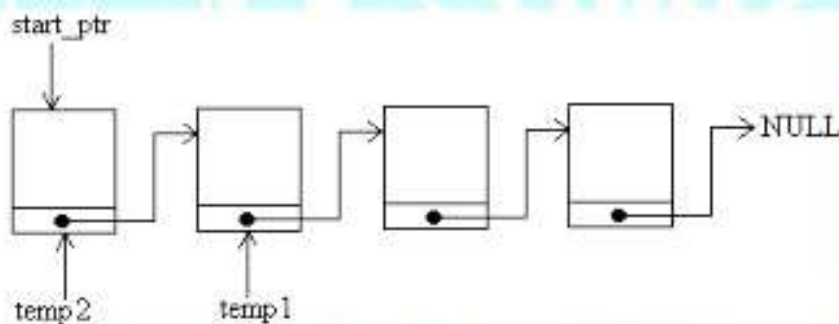
Firstly, the start pointer doesn't point to NULL, so we don't have to display a "Empty list, wise guy!" message. Let's get straight on with step2 - set the pointer `temp1` to the same as the start pointer:



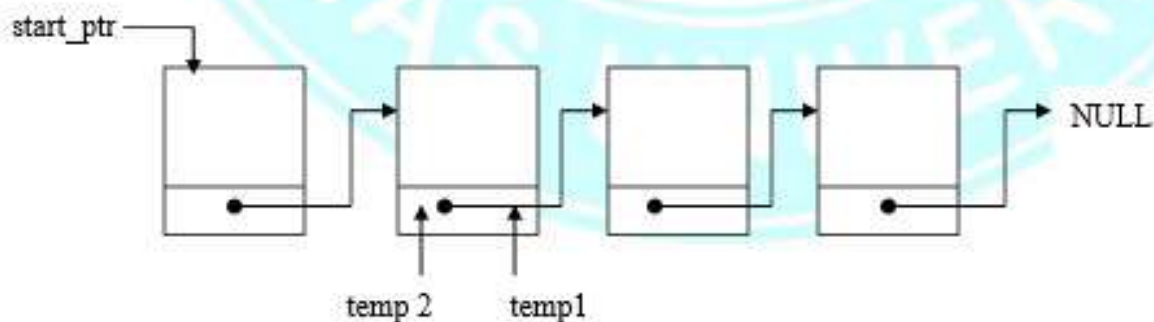
The **next** pointer from this node isn't NULL, so we haven't found the end node. Instead, we set the pointer **temp2** to the same node as **temp1**



and then move **temp1** to the next node in the list:

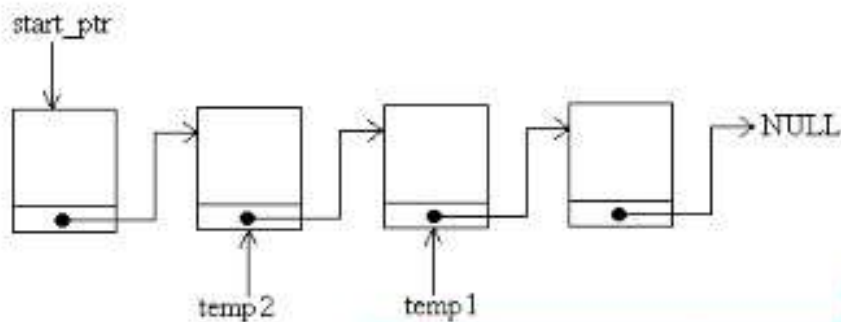


Going back to step 3, we see that **temp1** still doesn't point to the last node in the list, so we make **temp2** point to what **temp1** points to

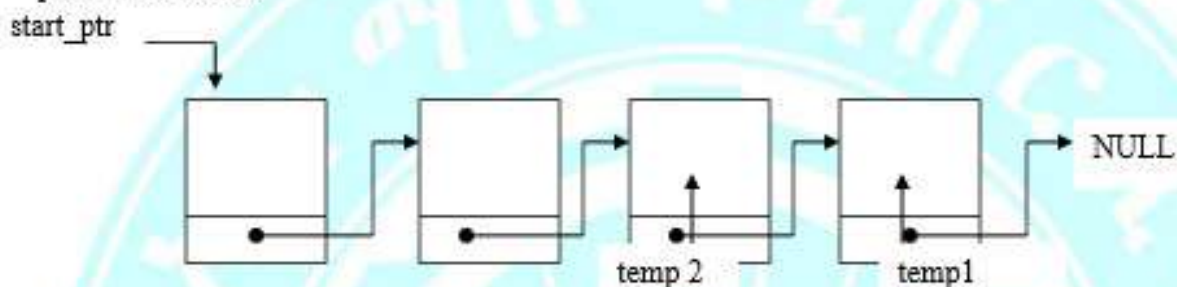


and **temp1** is made to point to the next node along:

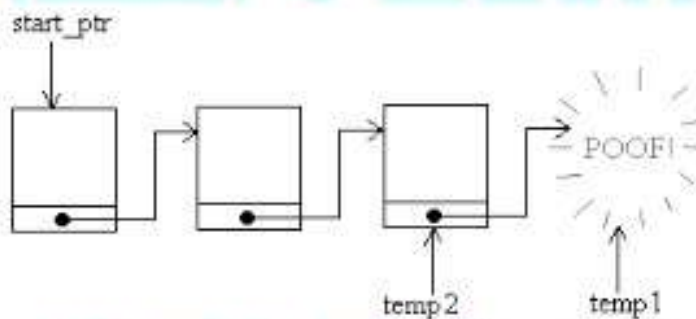




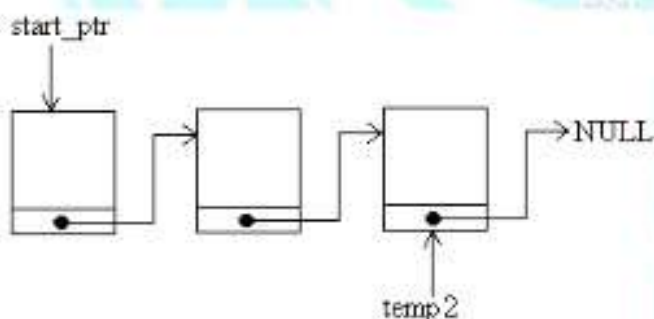
Eventually, this goes on until `temp1` really is pointing to the last node in the list, with `temp2` pointing to the penultimate node:



Now we have reached step 8. The next thing to do is to delete the node pointed to by `temp1`



and set the `next` pointer of what `temp2` indicates to `NULL`:



We suppose you want some code for all that! All right then ....

```
void delete_end_node()
{ node *temp1, *temp2;
  if (start_ptr == NULL)
    cout << "The list is empty!" << endl;
  else
```

```

        { temp1 = start_ptr;
          while (temp1->nxt != NULL)
            { temp2 = temp1;
              temp1 = temp1->nxt;
            }
          delete temp1;
          temp2->nxt = NULL;
        }
    }
}

```

The code seems a lot shorter than the explanation!

Now, the sharp-witted amongst you will have spotted a problem. If the list only contains one node, the code above will malfunction. This is because the function goes as far as the `temp1 = start_ptr` statement, but never gets as far as setting up `temp2`. The code above has to be adapted so that if the first node is also the last (has a `NULL` `nxt` pointer), then it is deleted and the `start_ptr` pointer is assigned to `NULL`. In this case, there is no need for the pointer `temp2`:

```

void delete_end_node()
{ node *temp1, *temp2;
  if (start_ptr == NULL)
    cout << "The list is empty!" << endl;
  else
    { temp1 = start_ptr;
      if (temp1->nxt == NULL) // This part is new!
        { delete temp1;
          start_ptr = NULL;
        }
      else
        { while (temp1->nxt != NULL)
            { temp2 = temp1;
              temp1 = temp1->nxt;
            }
          delete temp1;
          temp2->nxt = NULL;
        }
    }
}
}
}

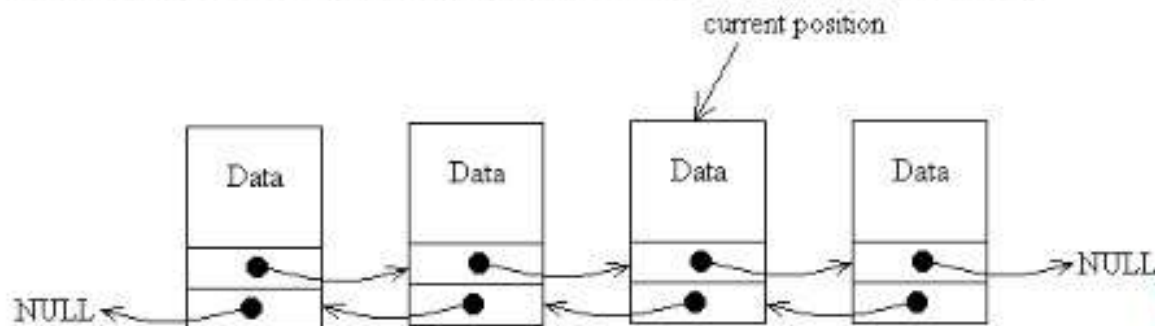
```



### 2.1.5. Doubly Linked Lists

That sounds even harder than a linked list! Well, if you've mastered how to do singly linked lists, then it shouldn't be much of a leap to doubly linked lists

A doubly linked list is one where there are links from each node in both directions:



You will notice that each node in the list has two pointers, one to the next node and one to the previous one - again, the ends of the list are defined by NULL pointers. Also there is no pointer to the start of the list. Instead, there is simply a pointer to some position in the list that can be moved left or right.

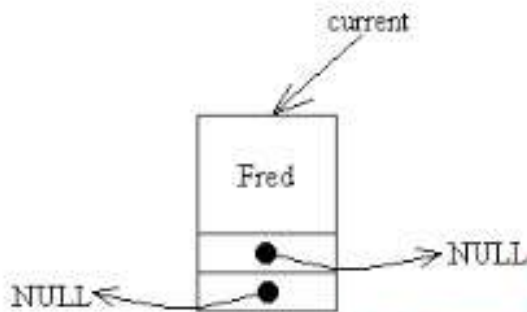
The reason we needed a start pointer in the ordinary linked list is because, having moved on from one node to another, we can't easily move back, so without the start pointer, we would lose track of all the nodes in the list that we have already passed. With the doubly linked list, we can move the current pointer backwards and forwards at will.

#### Creating Doubly Linked Lists

The nodes for a doubly linked list would be defined as follows:

```
struct node{
    char name[20];
    node *nxt;    // Pointer to next node
    node *prv;    // Pointer to previous node
};
node *current;
current = new node;
current->name = "Fred";
current->nxt = NULL;
current->prv = NULL;
```

We have also included some code to declare the first node and set its pointers to NULL. It gives the following situation:



We still need to consider the directions 'forward' and 'backward', so in this case, we will need to define functions to add a node to the start of the list (left-most position) and the end of the list (right-most position).

#### Adding a Node to a Doubly Linked List

```
void add_node_at_start (string new_name)
{ // Declare a temporary pointer and move it to the start
  node *temp = current;
  while (temp->prv != NULL)
    temp = temp->prv;
  // Declare a new node and link it in
  node *temp2;
  temp2 = new node;
  temp2->name = new_name; // Store the new name in the
  node
  temp2->prv = NULL;      // This is the new start of the
  list
  temp2->nxt = temp;      // Links to current list
  temp->prv = temp2;
}

void add_node_at_end ()
{ // Declare a temporary pointer and move it to the end
  node *temp = current;
  while (temp->nxt != NULL)
    temp = temp->nxt;
  // Declare a new node and link it in
  node *temp2;
  temp2 = new node;
```

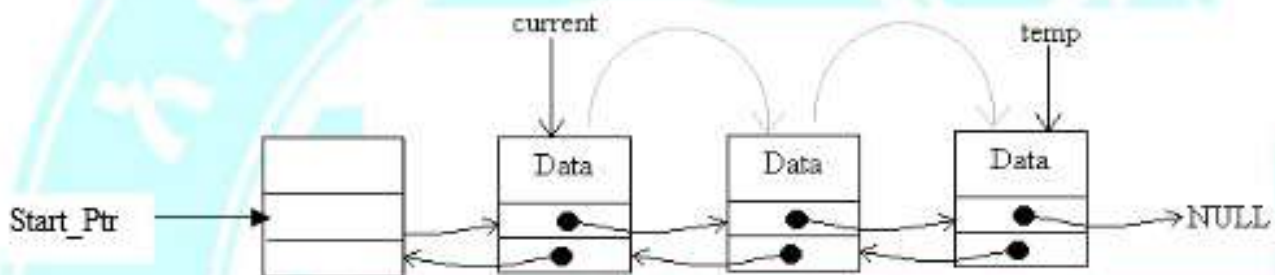


```

    temp2->name = new_name; // Store the new name in the
node
    temp2->nxt = NULL;      // This is the new start of the
list
    temp2->prv = temp;      // Links to current list
    temp->nxt = temp2;
}

```

Here, the new name is passed to the appropriate function as a parameter. We'll go through the function for adding a node to the right-most end of the list. The method is similar for adding a node at the other end. Firstly, a temporary pointer is set up and is made to march along the list until it points to last node in the list.



After that, a new node is declared, and the name is copied into it. The `nxt` pointer of this new node is set to `NULL` to indicate that this node will be the new end of the list.

The `prv` pointer of the new node is linked into the last node of the existing list.

The `nxt` pointer of the current end of the list is set to the new node.

### Stacks

A simple data structure, in which insertion and deletion occur at the same end, is termed (called) a stack. It is a LIFO (Last In First Out) structure.

The operations of insertion and deletion are called **PUSH** and **POP**.

**Push** - push (put) item onto stack

**Pop** - pop (get) item from stack

### Our Purpose:

To develop a stack implementation that does not tie us to a particular data type or to a particular implementation.

### Implementation:

Stacks can be implemented both as an array (contiguous list) and as a linked list. We want a set of operations that will work with either type of implementation: i.e. the method of implementation is hidden and can be changed without affecting the programs that use them.

### The Basic Operations:

**Push()**

```
{
```

```

    if there is room {
        put an item on the top of the stack
    }
    else
        give an error message
    }
}

Pop()
{
    if stack not empty {
        return the value of the top item
        remove the top item from the stack
    }
    else {
        give an error message
    }
}

CreateStack()
{
    remove existing items from the stack
    initialise the stack to empty
}

```

#### 2.1.6. Array Implementation of Stacks: The PUSH operation

Here, as you might have noticed, addition of an element is known as the PUSH operation. So, if an array is given to you, which is supposed to act as a STACK, you know that it has to be a STATIC Stack; meaning, data will overflow if you cross the upper limit of the array. So, keep this in mind.

##### Algorithm:

**Step-1:** Increment the Stack TOP by 1. Check whether it is always less than the Upper Limit of the stack.

If it is less than the Upper Limit go to step-2 else report -"Stack Overflow"

**Step-2:** Put the new element at the position pointed by the TOP

##### Array Implementation of Stacks: the POP operation

POP is the synonym for delete when it comes to Stack. So, if you're taking an array as the stack, remember that you'll return an error message, "Stack underflow", if an attempt is made to Pop an item from an empty Stack. OK.

##### Algorithm



**Step-1:** If the Stack is empty then give the alert "Stack underflow" and quit; or else go to step-2

**Step-2:** a) Hold the value for the element pointed by the TOP

b) Put a NULL value instead

c) Decrement the TOP by 1

### **Linked List Implementation of Stacks: the PUSH operation**

#### **Algorithm**

**Step-1:** If the Stack is empty go to step-2 or else go to step-3

**Step-2:** Create the new element and make your "stack" and "top" pointers point to it and quit.

**Step-3:** Create the new element and make the last (top most) element of the stack to point to it

**Step-4:** Make that new element your TOP most element by making the "top" pointer point to it.

### **Linked List Implementation of Stacks: the POP Operation**

#### **Algorithm:**

**Step-1:** If the Stack is empty then give an alert message "Stack Underflow" and quit; or else proceed

**Step-2:** If there is only one element left go to step-3 or else step-4

**Step-3:** Free that element and make the "stack", "top" and "bottom" pointers point to NULL and quit

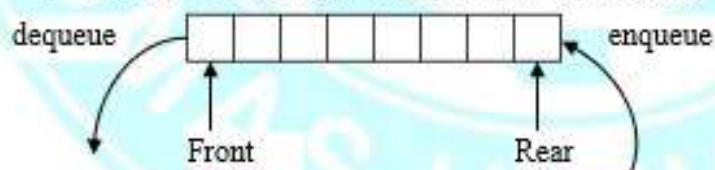
**Step-4:** Make "target" point to just one element before the TOP; free the TOP most element; make "target" as your TOP most element

#### **Applications of Stacks**

- Evaluation of Algebraic Expressions
- Reverse Polish Notation or postfix
- Function Calls

#### **Queue**

- a data structure that has access to its data at the front and rear.
- operates on FIFO (Fast In First Out) basis.
- uses two pointers/indices to keep track of information/data.
- has two basic operations:
  - enqueue - inserting data at the rear of the queue
  - dequeue - removing data at the front of the queue



### **Simple array implementation of enqueue and dequeue operations**

#### **Analysis:**

Consider the following structure: `int Num[MAX_SIZE];`

We need to have two integer variables that tell:

- the index of the front element
- the index of the rear element

We also need an integer variable that tells:

- the total number of data in the queue

```
int FRONT = -1, REAR = -1;
```

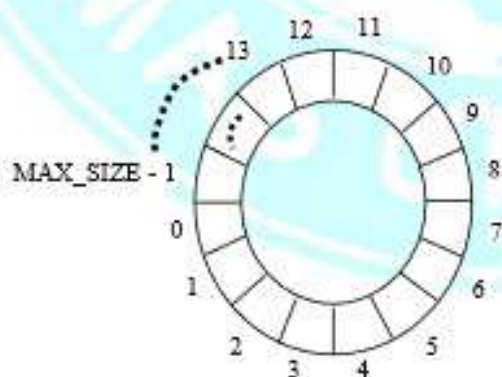
```
int QUEUE_SIZE = 0;
```

- To enqueue data to the queue
  - check if there is space in the queue  
 $REAR < MAX\_SIZE - 1$  ?
    - Yes:
      - Increment REAR
      - Store the data in Num[REAR]
      - Increment QUEUE\_SIZE
      - FRONT == -1 ?
        - Yes: - Increment FRONT
    - No: - Queue Overflow
- To dequeue data from the queue
  - check if there is data in the queue  
 $QUEUE\_SIZE > 0$  ?
    - Yes:
      - Copy the data in Num[FRONT]
      - Increment FRONT
      - Decrement QUEUE\_SIZE
    - No: - Queue Underflow

### Circular array implementation of enqueue and dequeue operations

A problem with simple arrays is we run out of space even if the queue never reaches the size of the array. Thus, simulated circular arrays (in which freed spaces are re-used to store data) can be used to solve this problem.

The circular array implementation of a queue with MAX\_SIZE can be simulated as follows:



### Analysis:

Consider the following structure: `int Num[MAX_SIZE];`

We need to have two integer variables that tell:

- the index of the front element



- the index of the rear element

We also need an integer variable that tells:

- the total number of data in the queue

int FRONT = -1, REAR = -1;

int QUEUE\_SIZE = 0;

- To enqueue data to the queue
  - o check if there is space in the queue  
 $QUEUE\_SIZE < MAX\_SIZE$  ?  
 Yes: { - Increment REAR  
            $REAR == MAX\_SIZE$  ?  
               Yes:  $REAR = 0$   
 - Store the data in Num[REAR]  
 - Increment QUEUE\_SIZE  
        $FRONT == -1$  ?  
           Yes: - Increment FRONT  
 No: - Queue Overflow
- To dequeue data from the queue
  - o check if there is data in the queue  
 $QUEUE\_SIZE > 0$  ?  
 Yes: { - Copy the data in Num[FRONT]  
           - Increment FRONT  
                $FRONT == MAX\_SIZE$  ?  
                   Yes:  $FRONT = 0$   
 - Decrement QUEUE\_SIZE  
 No: - Queue Underflow

### Priority Queue

- is a queue where each data has an associated key that is provided at the time of insertion.
- Dequeue operation deletes data having highest priority in the list
- One of the previously used dequeue or enqueue operations has to be modified

### Demerging Queues

- is the process of creating two or more queues from a single queue.
- used to give priority for some groups of data

### Merging Queues

- is the process of creating a priority queue from two or more queues.

- The ordinary dequeue implementation can be used to delete data in the newly created priority queue.

### Application of Queues

- Print server- maintains a queue of print jobs
- Disk Driver- maintains a queue of disk input/output requests
- Task scheduler in multiprocessing system- maintains priority queues of processes
- Telephone calls in a busy environment –maintains a queue of telephone calls
- Simulation of waiting line- maintains a queue of persons

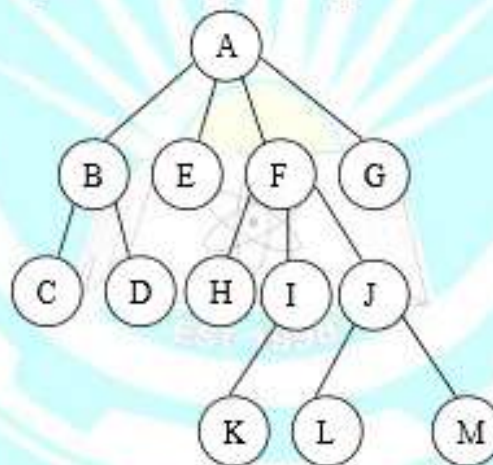
### Trees

A tree is a set of nodes and edges that connect pairs of nodes that connect pairs of nodes. It is an abstract model of a hierarchical structure. Rooted tree has the following structure:

- One node distinguished as root.
- Every node C except the root is connected from exactly other node P. P is C's parent, and C is one of C's children.
- There is a unique path from the root to the each node.
- The number of edges in a path is the length of the path.

### Tree Terminologies

Consider the following tree.



Root: a node with out a parent. → A

Internal node: a node with at least one child. → A, B, F, I, J

External (leaf) node: a node without a child. → C, D, E, H, K, L, M, G

Ancestors of a node: parent, grandparent, grand-grandparent, etc of a node.

Ancestors of K → A, F, I

Descendants of a node: children, grandchildren, grand-grandchildren etc of a node.

Descendants of F → H, I, J, K, L, M

Depth of a node: number of ancestors or length of the path from the root to the node.

Depth of H → 2



Height of a tree: depth of the deepest node.  $\rightarrow 3$

Subtree: a tree consisting of a node and its descendants.

Binary tree: a tree in which each node has at most two children called left child and right child.

Full binary tree: a binary tree where each node has either 0 or 2 children.

Balanced binary tree: a binary tree where each node except the leaf nodes has left and right children and all the leaves are at the same level.

Complete binary tree: a binary tree in which the length from the root to any leaf node is either  $h$  or  $h-1$  where  $h$  is the height of the tree. The deepest level should also be filled from left to right.

Binary search tree (ordered binary tree): a binary tree that may be empty, but if it is not empty it satisfies the following.

- Every node has a key and no two elements have the same key.
- The keys in the right subtree are larger than the keys in the root.
- The keys in the left subtree are smaller than the keys in the root.
- The left and the right subtrees are also binary search trees.

### 2.1.7. Operations on Binary Search Tree

#### Insertion

When a node is inserted the definition of binary search tree should be preserved. Suppose there is a binary search tree whose root node is pointed by `RootNodePtr` and we want to insert a node (that stores a value) pointed by `InsNodePtr`.

Case 1: There is no data in the tree (i.e. `RootNodePtr` is NULL)

- The node pointed by `InsNodePtr` should be made the root node.

Case 2: There is data

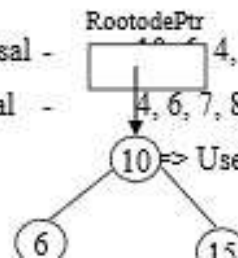
- Search the appropriate position.
- Insert the node in that position.

#### Traversing

Binary search tree can be traversed in three ways.

- Pre order traversal - traversing binary tree in the order of parent, left and right.
- Inorder traversal - traversing binary tree in the order of left, parent and right.
- Postorder traversal - traversing binary tree in the order of *left, right and parent*.

Example:

Preorder traversal -  4, 8, 7, 15, 14, 12, 11, 13, 18, 16, 17, 19

Inorder traversal - 4, 6, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19

10  $\Rightarrow$  Used to display nodes in ascending order.



Postorder traversal- 4, 7, 8, 6, 11, 13, 12, 14, 17, 16, 19, 18, 15, 10

### Application of binary tree traversal

- Store values on leaf nodes and operators on internal nodes

Preorder traversal - used to generate mathematical expression in prefix notation.

Inorder traversal - used to generate mathematical expression in infix notation.

Postorder traversal - used to generate mathematical expression in postfix notation.

### Function calls:

Preorder(RootNodePtr);

Inorder(RootNodePtr);

Postorder(RootNodePtr);

### Deletion

To delete a node (whose Num value is N) from binary search tree (whose root node is pointed by RootNodePtr), four cases should be considered. When a node is deleted the definition of binary search tree should be preserved.

Consider the following binary search tree.

Case 1: Deleting a leaf node (a node having no child)

Case 2: Deleting a node having only one child,

Approach 1: Deletion by merging – one of the following is done

- If the deleted node is the left child of its parent and the deleted node has only the left child, the left child of the deleted node is made the left child of the parent of the deleted node.
- If the deleted node is the left child of its parent and the deleted node has only the right child, the right child of the deleted node is made **the left child** of the parent of the deleted node.
- If the deleted node is the right child of its parent and the node to be deleted has only the left child, the left child of the deleted node is made the right child of the parent of the deleted node.
- If the deleted node is the right child of its parent and the deleted node has only the right child, the right child of the deleted node is made **the right child** of the parent of the deleted node.

Approach 2: Deletion by copying- the following is done

- Copy the node containing the largest element in the left (or the smallest element in the right) to the node containing the element to be deleted
- Delete the copied node



Case 3: Deleting a node having two children, e.g. 6

Approach 1: Deletion by merging – one of the following is done

- If the deleted node is the left child of its parent, one of the following is done
  - The left child of the deleted node is made the left child of the parent of the deleted node, and
  - The right child of the deleted node is made the right child of the node containing largest element in the left of the deleted node
  - OR
  - The right child of the deleted node is made the left child of the parent of the deleted node, and
  - The left child of the deleted node is made the left child of the node containing smallest element in the right of the deleted node
- If the deleted node is the right child of its parent, one of the following is done
  - The left child of the deleted node is made the right child of the parent of the deleted node, and
  - The right child of the deleted node is made the right child of the node containing largest element in the left of the deleted node
  - OR
  - The right child of the deleted node is made the right child of the parent of the deleted node, and
  - The left child of the deleted node is made the left child of the node containing smallest element in the right of the deleted node

Approach 2: Deletion by copying- the following is done

- Copy the node containing the largest element in the left (or the smallest element in the right) to the node containing the element to be deleted
- Delete the copied node

Case 4: Deleting the root node, 10

Approach 1: Deletion by merging- one of the following is done

- If the tree has only one node the root node pointer is made to point to nothing (NULL)
- If the root node has left child
  - the root node pointer is made to point to the left child

- o the right child of the root node is made the right child of the node containing the largest element in the left of the root node
- If root node has right child
  - o the root node pointer is made to point to the right child
  - o the left child of the root node is made the left child of the node containing the smallest element in the right of the root node

Approach 2: Deletion by copying- the following is done

- Copy the node containing the largest element in the left (or the smallest element in the right) to the node containing the element to be deleted
- Delete the copied node

### Advanced Sorting and Searching Algorithms

#### Shell Sort

Shell sort is an improvement of insertion sort. It is developed by Donald Shell in 1959. Insertion sort works best when the array elements are sorted in a reasonable order. Thus, shell sort first creates this reasonable order.

Algorithm:

1. Choose gap  $g_k$  between elements to be partly ordered.
2. Generate a sequence (called increment sequence)  $g_k, g_{k-1}, \dots, g_2, g_1$  where for each sequence  $g_k, A[j] \leq A[j+g_k]$  for  $0 \leq j \leq n-1-g_k$  and  $k \geq i \geq 1$

It is advisable to choose  $g_k = n/2$  and  $g_{k-1} = g_k/2$  for  $k \geq i \geq 1$ . After each sequence  $g_{k-1}$  is done and the list is said to be  $g_i$ -sorted. Shell sorting is done when the list is  $1$ -sorted (which is sorted using insertion sort) and  $A[j] \leq A[j+1]$  for  $0 \leq j \leq n-2$ . Time complexity is  $O(n^{3/2})$ .

Example: Sort the following list using shell sort algorithm.

5	8	2	4	1	3	9	7	6	0
---	---	---	---	---	---	---	---	---	---

Choose  $g_3 = 5$  ( $n/2$  where  $n$  is the number of elements = 10)

Sort (5, 3)

Sort (8, 9)

Sort (2, 7)

Sort (4, 6)

Sort (1, 0)

→ 5- sorted list

3	8	2	4	1	5	9	7	6	0
3	8	2	4	1	5	9	7	6	0
3	8	2	4	1	5	9	7	6	0
3	8	2	4	1	5	9	7	6	0
3	8	2	4	0	5	9	7	6	1
3	8	2	4	0	5	9	7	6	1



Choose  $g_2=3$

Sort (3, 4, 9, 1)

Sort (8, 0, 7)

Sort (2, 5, 6)

→ 3- sorted list

1	8	2	3	0	5	4	7	6	9
1	0	2	3	7	5	4	8	6	9
1	0	2	3	7	5	4	8	6	9
1	0	2	3	7	5	4	8	6	9

Choose  $g_1=1$  (the same as insertion sort algorithm)

Sort (1, 0, 2, 3, 7, 5, 4, 8, 6, 9)

→ 1- sorted (shell sorted) list

0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9

### Quick Sort

Quick sort is the fastest known algorithm. It uses divide and conquer strategy and in the worst case its complexity is  $O(n^2)$ . But its expected complexity is  $O(n \log n)$ .

#### Algorithm:

1. Choose a pivot value (mostly the first element is taken as the pivot value)
2. Position the pivot element and partition the list so that:
  - the left part has items less than or equal to the pivot value
  - the right part has items greater than or equal to the pivot value
3. Recursively sort the left part
4. Recursively sort the right part

The following algorithm can be used to position a pivot value and create partition.

```
Left=0;
Right=n-1; // n is the total number of elements in the list
PivotPos=Left;
while(Left<Right)
{
    if(PivotPos==Left)
    {
        if(Data[Left]>Data[Right])
        {
            swap(data[Left], Data[Right]);
            PivotPos=Right;
            Left++;
        }
        else
            Right--;
    }
    else
        // ... (rest of the algorithm code)
}
```

```
{  
    if(Data[Left]>Data[Right])  
    {  
        swap(data[Left], Data[Right]);  
        PivotPos=Left;  
        Right--;  
    }  
    else  
        Left++;  
}  
}
```





Example: Sort the following list using quick sort algorithm.

5 8 2 4 1 3 9 7 6 0

5 8 2 4 1 3 9 7 6 0

Left Pivot Right

0 8 2 4 1 3 9 7 6 5

Left Right Pivot

0 5 2 4 1 3 9 7 6 8

Left Pivot Right

0 5 2 4 1 3 9 7 6 8

Left Pivot Right

0 5 2 4 1 3 9 7 6 8

Left Pivot Right

0 5 2 4 1 3 9 7 6 8

Left Pivot Right

0 5 2 4 1 3 9 7 6 8

Left Pivot Right

0 5 2 4 1 3 9 7 6 8

Left Pivot Right

0 3 2 4 1 5 9 7 6 8

Left Right Pivot

0 3 2 4 1 5 9 7 6 8

Left Right Pivot

0 3 2 4 1 5 9 7 6 8

Left Right Pivot

0 3 2 4 1 5 9 7 6 8

Left Pivot Right Left Pivot Right

0 3 2 4 1 5 8 7 6 9

Left Pivot Right Left Right Pivot

0 3 2 4 1 5 8 7 6 9

Left Pivot Right Left Right Pivot

0 3 2 4 1 5 8 7 6 9

Left Pivot Right Left Right Pivot

0 3 2 4 1 5 6 7 8 9

Left Pivot Right Left Right Pivot

0 1 2 4 3 5 6 7 8 9

Left Right Pivot Left Right Pivot

0 1 2 4 3 5 6 7 8 9

Left Right Pivot

0 1 2 3 4 5 6 7 8 9

Left Right Pivot

0 1 2 3 4 5 6 7 8 9

Left Right Pivot

0 1 2 3 4 5 6 7 8 9

Heap sort operates by first converting the list in to a heap tree. Heap tree is a binary tree in which each node has a value greater than both its children (if any). It uses a process called "adjust" to accomplish its task (building a heap tree) whenever a value is larger than its parent. The time complexity of heap sort is  $O(n \log n)$ .

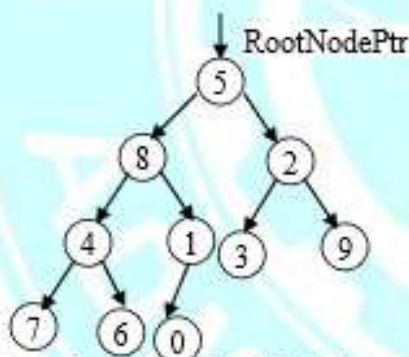
**Algorithm:**

1. Construct a binary tree
  - The root node corresponds to  $\text{Data}[0]$ .
  - If we consider the index associated with a particular node to be  $i$ , then the left child of this node corresponds to the element with index  $2*i+1$  and the right child corresponds to the element with index  $2*i+2$ . If any or both of these elements do not exist in the array, then the corresponding child node does not exist either.
2. Construct the heap tree from initial binary tree using "adjust" process.
3. Sort by swapping the root value with the lowest, right most value and deleting the lowest, right most value and inserting the deleted value in the array in it proper position.

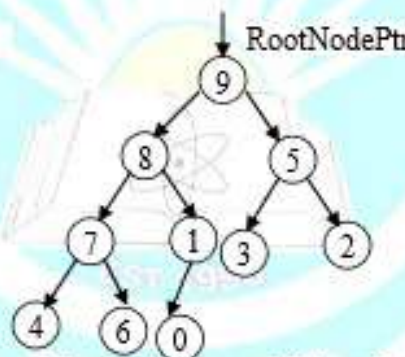
Example: Sort the following list using heap sort algorithm.

5	8	2	4	1	3	9	7	6	0
---	---	---	---	---	---	---	---	---	---

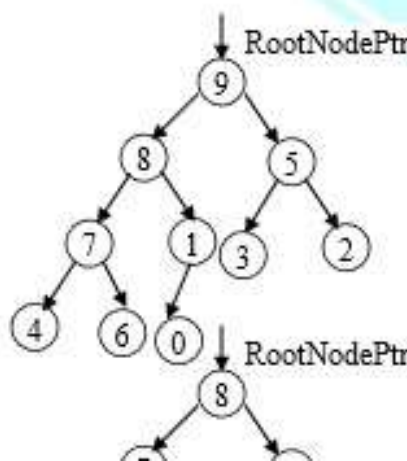
Construct the initial binary tree



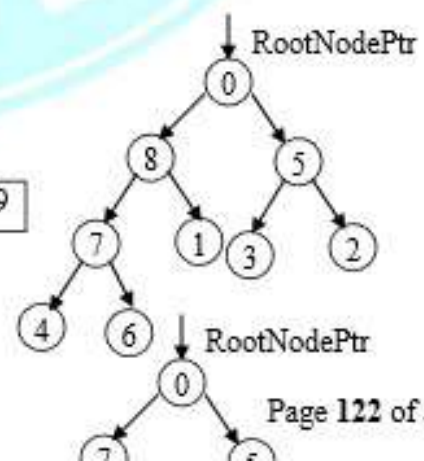
Construct the heap tree



Swap the root node with the lowest, right most node and delete the lowest, right most value; insert the deleted value in the array in its proper position; adjust the heap tree; and repeat this process until the tree is empty.

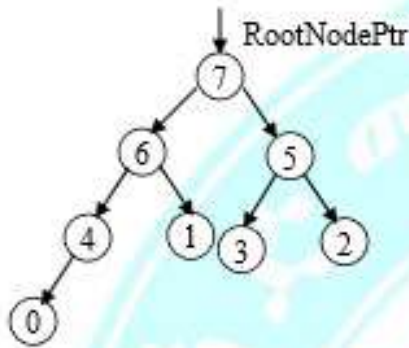


									9
--	--	--	--	--	--	--	--	--	---

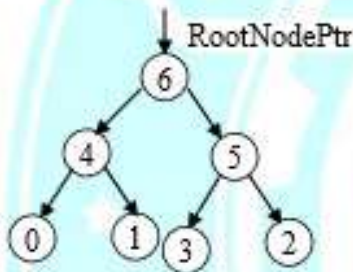
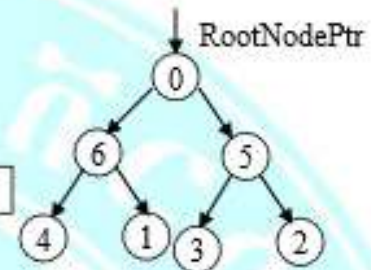




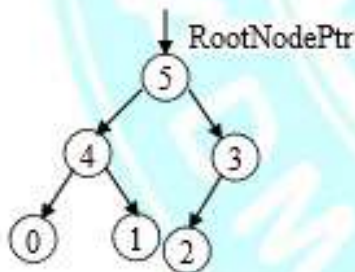
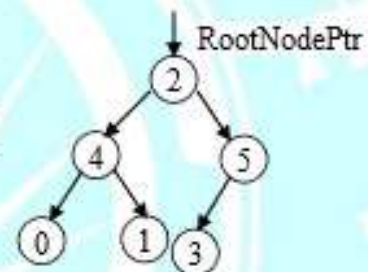
								8	9
--	--	--	--	--	--	--	--	---	---



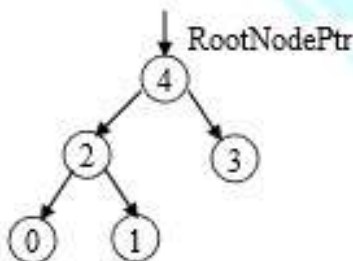
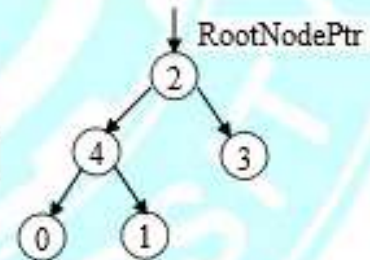
								7	8	9
--	--	--	--	--	--	--	--	---	---	---



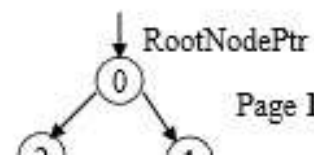
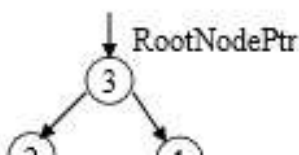
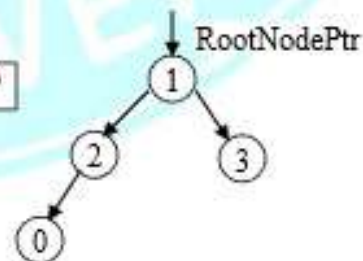
							6	7	8	9
--	--	--	--	--	--	--	---	---	---	---



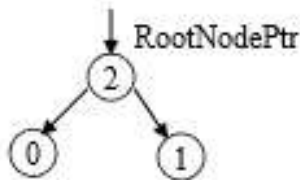
							5	6	7	8	9
--	--	--	--	--	--	--	---	---	---	---	---



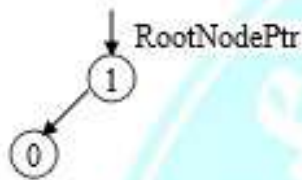
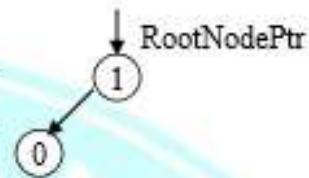
							4	5	6	7	8	9
--	--	--	--	--	--	--	---	---	---	---	---	---



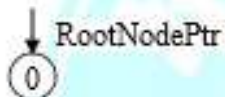
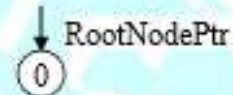
			3	4	5	6	7	8	9
--	--	--	---	---	---	---	---	---	---



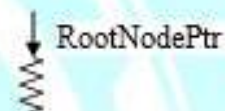
		2	3	4	5	6	7	8	9
--	--	---	---	---	---	---	---	---	---



	1	2	3	4	5	6	7	8	9
--	---	---	---	---	---	---	---	---	---



0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



### Merge Sort

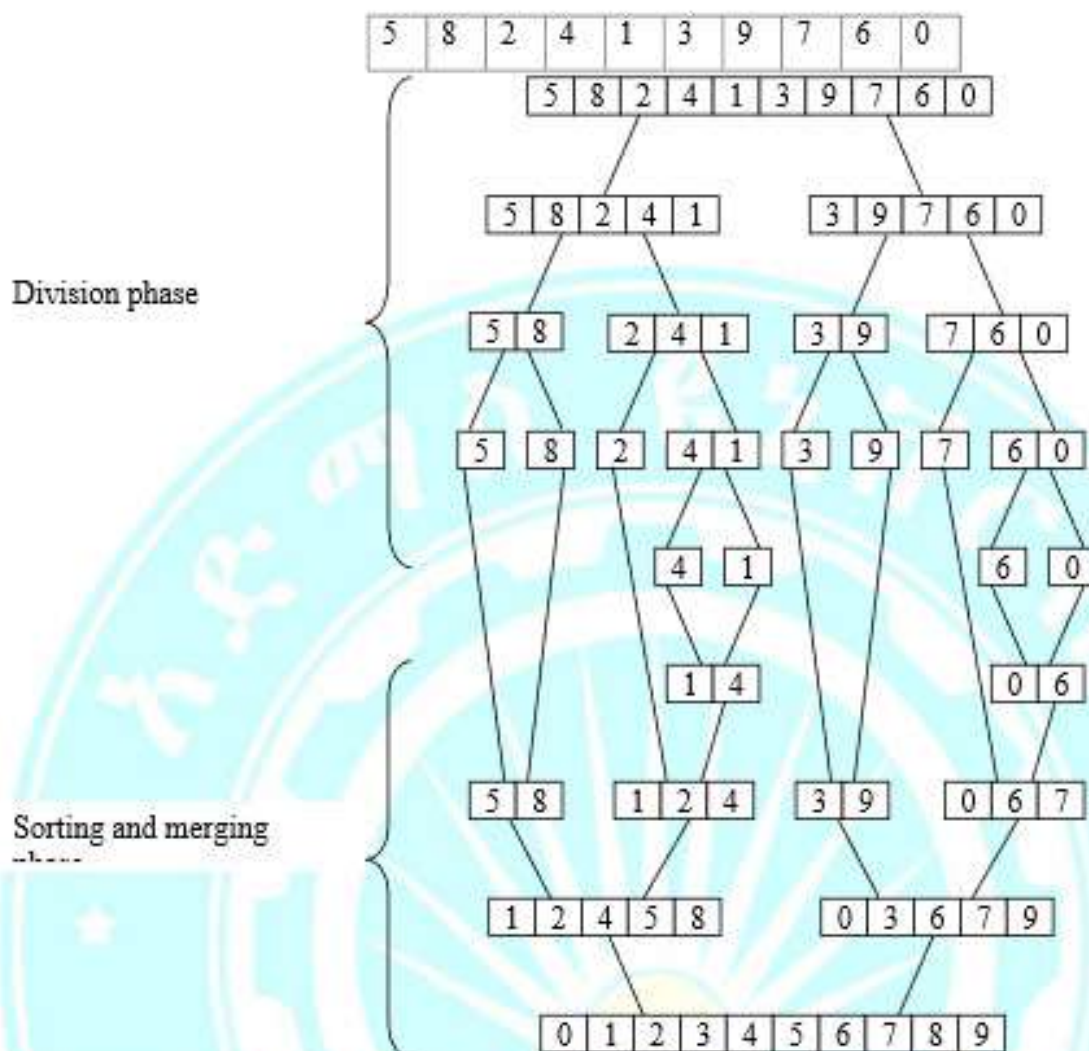
Like quick sort, merge sort uses divide and conquer strategy and its time complexity is  $O(n \log n)$ .

#### Algorithm:

1. Divide the array in to two halves.
2. Recursively sort the first  $n/2$  items.
3. Recursively sort the last  $n/2$  items.
4. Merge sorted items (using an auxiliary array).

Example: Sort the following list using merge sort algorithm.





## 2.2. Design and Analysis of Algorithm

The following is a list of several popular design approaches:

### 2.2.1. Divide and Conquer Approach

It is a top-down approach. The algorithms which follow the divide & conquer techniques involve three steps:

- Divide the original problem into a set of subproblems.
- Solve every subproblem individually, recursively.
- Combine the solution of the subproblems (top level) into a solution of the whole original problem.

The following computer algorithms are based on the divide and conquer approach

- a. maximum and minimum problems

- b. binary search
- c. merge sort and quick sort
- d. tower of Hanoi

### **Advantages of Divide and Conquer**

- Divide and Conquer tend to successfully solve one of the biggest problems, such as the Tower of Hanoi, a mathematical puzzle. It is challenging to solve complicated problems for which you have no basic idea, but with the help of the divide and conquer approach, it has lessened the effort as it works on dividing the main problem into two halves and then solve them recursively. This algorithm is much faster than other algorithms.
- It efficiently uses cache memory without occupying much space because it solves simple subproblems within the cache memory instead of accessing the slower main memory.
- It is more proficient than that of its counterpart Brute Force technique.
- Since these algorithms inhibit parallelism, it does not involve any modification and is handled by systems incorporating parallel processing.

### **Disadvantages of Divide and Conquer**

- Since most of its algorithms are designed by incorporating recursion, so it necessitates high memory management.
- An explicit stack may overuse the space.
- It may even crash the system if the recursion is performed rigorously greater than the stack present in the CPU.

### **2.2.2. Greedy Technique**

Greedy method is used to solve the optimization problem. An optimization problem is one in which we are given a set of input values, which are required either to be maximized or minimized (known as objective), i.e. some constraints or conditions.

- Greedy Algorithm always makes the choice (greedy criteria) looks best at the moment, to optimize a given objective.
- The greedy algorithm doesn't always guarantee the optimal solution however it generally produces a solution that is very close in value to the optimal.

### **Applications of Greedy Algorithm**



- It is used in finding the shortest path.
- It is used to find the minimum spanning tree using the prim's algorithm or the Kruskal's algorithm.
- It is used in a job sequencing with a deadline.
- This algorithm is also used to solve the fractional knapsack problem.

### **2.2.3. Dynamic Programming**

Dynamic Programming is a bottom-up approach we solve all possible small problems and then combine them to obtain solutions for bigger problems. Dynamic programming is a technique that breaks the problems into sub-problems, and saves the result for future purposes so that we do not need to compute the result again.

This is particularly helpful when the number of copying subproblems is exponentially large. Dynamic Programming is frequently related to Optimization Problems.

#### **How does the dynamic programming approach work?**

The following are the steps that the dynamic programming follows:

- It breaks down the complex problem into simpler subproblems.
- It finds the optimal solution to these sub-problems.
- It stores the results of subproblems (memorization). The process of storing the results of subproblems is known as memorization.
- It reuses them so that same sub-problem is calculated more than once.
- Finally, calculate the result of the complex problem.

### **2.2.4. Backtracking Algorithm**

Backtracking Algorithm tries each possibility until they find the right one. It is a depth-first search of the set of possible solution. During the search, if an alternative doesn't work, then backtrack to the choice point, the place which presented different alternatives, and tries the next alternative.

#### **Applications of Backtracking**

- N-queen problem
- Sum of subset problem
- Graph coloring
- Hamilton cycle

**Dijkstra algorithm** is a single-source shortest path algorithm. Here, single-source means that only one source is given, and we have to find the shortest path from the source to all the nodes

**The minimum spanning tree** is a spanning tree whose sum of the edges is minimum. Kruskal and Prim are the two methods.

### 2.2.5. Kruskal's Algorithm

An algorithm to construct a Minimum Spanning Tree for a connected weighted graph. It is a Greedy Algorithm. The Greedy Choice is to put the smallest weight edge that does not because a cycle in the MST constructed so far.

### Prim's Algorithm

It is a greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices:

- Contain vertices already included in MST.
- Contain vertices not yet included.

At every step, it considers all the edges and picks the minimum weight edge. After picking the edge, it moves the other endpoint of edge to set containing MST.





- 
1. In a stack, if a user tries to remove an element from an empty stack it is called \_\_\_\_\_.
    - a) Underflow
    - b) Empty collection
    - c) Overflow
    - d) Garbage Collection
  2. Which of the following is not the application of stack?
    - a) A parentheses balancing program
    - b) Tracking of local variables at run time
    - c) Compiler Syntax Analyzer
    - d) Data Transfer between two asynchronous process
  3. Consider the usual algorithm for determining whether a sequence of parentheses is balanced. The maximum number of parentheses that appear on the stack AT ANY ONE TIME when the algorithm analyzes:  $((O(OX(O)))?)$ 
    - a) 1
    - b) 2
    - c) 3
    - d) 4 or more
  4. What is the value of the postfix expression  $6\ 3\ 2\ 4\ +\ -\ *?$ 
    - a) 1
    - b) 40
    - c) 74
    - d) -18
  5. What are the advantages of arrays?
    - a) Objects of mixed data types can be stored
    - b) Elements in an array cannot be sorted
    - c) Index of first element of an array is 1
    - d) Easier to store elements of same data type
  6. The postfix form of the expression  $(A + B) * (C * D - E) * F / G$  is?
    - a)  $AB + CD * E - FG /**$
    - b)  $AB + CD * E - F **G /$
    - c)  $AB + CD * E - *F *G /$
    - d)  $AB + CDE * - *F *G /$
  7. What data structure would you most likely see in non-recursive implementation of a recursive algorithm?

- a) Linked List  
b) Stack  
c) Queue  
d) Tree
8. If the elements "A", "B", "C" and "D" are placed in a stack and are deleted one at a time, what is the order of removal?  
a) ABCD  
b) DCBA  
c) DCAB  
d) ABDC
9. A linear list of elements in which deletion can be done from one end (front) and insertion can take place only at the other end (rear) is known as \_\_\_\_\_  
a) Queue  
b) Stack  
c) Tree  
d) Linked list
10. What would be the asymptotic time complexity to add a node at the end of singly linked list, if the pointer is initially pointing to the head of the list?  
a)  $O(1)$   
b)  $O(n)$   
c)  $\theta(n)$   
d)  $\theta(1)$
11. What would be the asymptotic time complexity to find an element in the linked list?  
a)  $O(1)$   
b)  $O(n)$   
c)  $O(n^2)$   
d)  $O(n^4)$
12. What is the worst case time complexity of inserting a node in a doubly linked list?  
a)  $O(n \log n)$   
b)  $O(\log n)$   
c)  $O(n)$   
d)  $O(1)$
13. The pre-order and in-order are traversals of a binary tree are T M L N P O Q and L M N T O P Q. Which of following is post-order traversal of the tree?



- a) LNMOQPT  
b) NMOPOLT  
c) LMNOPQT  
d) OPLMNQT
14. The number of edges from the node to the deepest leaf is called \_\_\_\_\_ of the tree.  
a) Height  
b) Depth  
c) Length  
d) Width
15. What is a full binary tree?  
a) Each node has exactly zero or two children  
b) Each node has exactly two children  
c) All the leaves are at the same level  
d) Each node has exactly one or two children
16. In a full binary tree if number of internal nodes is  $I$ , then number of nodes  $N$  are?  
a)  $N = 2 * I$   
b)  $N = I + 1$   
c)  $N = I - 1$   
d)  $N = 2 * I + 1$
17. Dijkstra's algorithm is used to solve \_\_\_\_\_ problems  
a) Network lock  
b) Single source shortest path  
c) All pair shortest path  
d) Sorting
18. Which of the following is used for solving the N Queens Problem?  
a) Greedy algorithm  
b) Dynamic programming  
c) Backtracking  
d) Sorting
19. Which of the following is a Divide and Conquer algorithm?  
a) Bubble sort  
b) Selection sort

- c) Heap sort
- d) Merge sort

**Answer**

- 1. A
- 2. D
- 3. C
- 4. D
- 5. D
- 6. C
- 7. B
- 8. B
- 9. A
- 10. C
- 11. B
- 12. C
- 13. A
- 14. A
- 15. A
- 16. D
- 17. B
- 18. C
- 19. D





## Part III: Database Systems

### 3.1. Introduction

#### 3.1.1. Data, Information and Information System

##### Data

Data are the raw (unorganized) facts that are stored and so not helpful in making decisions.

- ✚ Raw facts are seldom immediately useful to a decision maker. (by themselves are useless)
- ✚ What the decision maker really needs is information, which is defined as data processed and presented in a meaningful form.

We live in the Information Age. Information used to make organizations more productive and competitive. So in order to make data useful we have to process it and the processed data is called Information.

##### Information

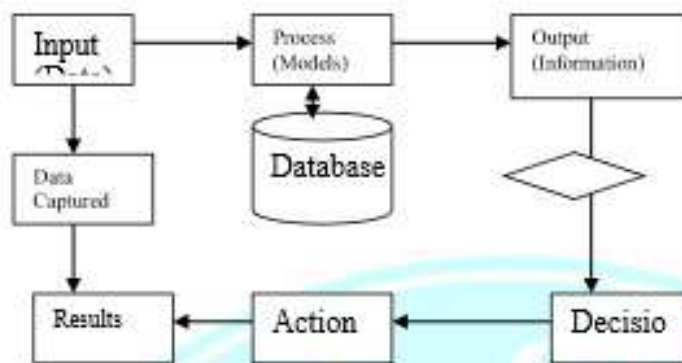
Information is nothing but a refined data.

**Burch and Grudnitski:** "Information is data that have been put into a meaningful and useful context and communicated to a recipient who uses it to make decisions."

- ✚ For effective decisions, decision makers need accurate information. To obtain this accurate information, data must be organized scientifically. Thus, the more accurate, relevant and timely the information is, the better the informed people will be when making decisions.
- ✚ *Information* consists of facts and items of knowledge. It can be anything that has meaning to people.
  - A list of names and addresses
  - The contents of a letter
  - The words of a song
  - Processing Information
  - A map

##### ✚ *Information Processing Tasks*

- **Information processing** is the organisation, manipulation and distribution of information.
- When data is collected and processed a series of operations is carried out on it this is known as the **data processing cycle**
  - Data collection
  - Input
  - Process
  - Output



## Information systems

A system is a set of components which works as a unit, this includes the hardware involved, the software, the people and the methods used.

### 1.1.1 Forms and resources of Data

#### Types of Data

1. **Quantitative:** information that can be added or subtracted.  
Eg. Daily expenses or incomes
2. **Qualitative:**  
Eg. DOB, sex, educational background, profession
3. Textual (Descriptive)
4. Audio
5. Video

### 3.1.2. Levels of Data

Basically there are three types of data. Namely:

- a. Objects in the real world
- b. Structures of a database (logical model)
- c. Actual data

#### Objects in the real world

At this level we talk about the organization for which the database is designed, we identify entities (identify objects).

- i. Entities are real objects, like students, Employees, items
- ii. Attributes are identified: they are descriptions/properties of entity.

Eg. Attributes of a student (entity): Name, sex, Age, Address, Id. No, class (Batch)



iii. Relationships between objects must be defined.

Eg. Entity(Student)    Student  $\xleftarrow{\text{enroll}} \xrightarrow{\text{enroll}}$  Course

### Logical Model

- ✚ Is information about the logical structure of the database?
- ✚ It describes the property or characteristic of other data.
- ✚ It is stored in the data dictionary of the DBMS.

Eg.	Data type	width	Rules
Name	char	30	No abb, No blank
Age	int	2	$18 < \text{Age} < 50$
Sex	char	1	Only 'M' or 'F'
Tel	char	10	--
Id. No	char	6	No Blank

The logical model should allow users to keep track of the real changes of entities, attributes and relationships.

### Actual Data

It consists of data or occurrences of a real object in the database. For each entity in the real world there is an occurrence of a corresponding record in the database.

Eg. Entity can be represented using a table or file (object)

XX	YY	ZZ

Fields/attributes

- ✚ Rows (records) are the actual representation of the real object.
- ✚ Records are simply instances or actual occurrences of the real object.

### 3.1.3. What is a Database System?

A database system is basically a computerized record keeping system; that is, it is a computerized system whose overall purpose is to maintain information and to make that information available on demand. The information concerned can be anything that is deemed to be of significance to the individual or organization the system is intended to serve. That is, needed to assist in the general process of running the business of that individual or organization.

### What is a Database?

A database can be taken as an organized collection of related data. It consists of some collection of persistent data that is used by application systems of some given enterprise (any reasonably self-contained commercial, scientific, technical, or other organization). Enterprises nowadays often maintain two distinct databases, one containing operational data and one containing decision support data. The decision support database frequently consists of *summary information* (e.g., totals, averages), where that summary information in turn is extracted from the operational database on a periodic basis, say, once a day or once a week.



The advantages of a database system over the traditional, paper-based methods of record keeping will include the following:

**Compactness:** no need for possibly voluminous paper files.

**Speed:** the machine can retrieve and change data faster than a human can. In particular, ad hoc, spur-of-the-moment queries ("Do we have more red screws than blue ones?") can be answered quickly without any need for time consuming manual or visual searches.

**Accuracy:** timely, accurate and up-to-date information is available on demand at any time.

The foregoing benefits apply with even more force in a multi-user environment where the database is likely to be much larger and much more complex than in the single user case. In a multi-user environment the database system provides the enterprise with centralized control of its data. The centralized approach has the advantages:

- **Redundancy can be reduced**

In non database or non centralized systems each application or department keeps its own private files. The files may hold common data elements that exist as part of the enterprises data. This will lead to considerable redundancy in stored data, with resultant waste in storage space. For example, a personnel application and an education records application might both own a file that includes department information for employees. Note that, this is not to say we should eliminate all redundancies. Sometimes there are sound reasons for maintaining several copies of the same data.

- **Inconsistency can (to some extent) avoided**

If there exist a number of files which store similar data elements among other sorts of data then when a change is made to a particular data (among the common ones) this change need to be done throughout the system where there is such data stored. This is not, often, the case. Some of the data might be updated and others left as they are which results in an inconsistent information about the same phenomena.

- **Standards can be enforced**

Standardizing data representation is particularly desirable as an aid to data interchange or migration of data between systems. Likewise, data naming and documentation standards are also very desirable as they facilitate data sharing and understandability.

- **Security restrictions can be applied**

Since the data is stored in one place/area all accesses to the data can be regulated by the system through some defined rules built into the system. The system ensures that the only means of access to



the database is through proper channels. Different rules can be established for each type of access (retrieve, insert, delete, etc.) to each of information to the database.

- **Integrity can be maintained**

The problem of integrity is the problem of ensuring the data in the database is accurate. Inconsistency between two entries that represent the same “fact” is an example of lack of integrity. It is more serious in a multi-user environment where one user may enter bad data and other users may go on working on the updated data as if it were a correct one.

- **Conflicting requirements can be balanced**

Knowing the overall requirements of the enterprise the Database Administrator (DBA) can structure the system so as to provide an overall service that is “best for the enterprise”. For example, a representation can be chosen for the data in storage that gives fast access for the most important applications(possibly at the cost of poorer performance for certain other applications).

#### **3.1.4. Components of a Database System**

A database system involves four major components, namely, data, hardware, software and users. A brief discussion will follow on each of these components.

##### **Data**

The actual data stored in the database system may be stored as a single database or distributed in many distinct files and treated as one. Is the system a single-user or multi-user one? How are we going to achieve the utmost possible performance concerning the data storage and maintenance? What other benefits or drawbacks do we expect as the result of placement or structure of the database? These and similar issues might be concerned with the way the data stored in the system.

##### **Hardware**

This portion of the system consists of secondary storage media (disks, tapes and optical media) that are used to hold the stored data and associated device controllers (hard disk controller, etc.); and the processor(s) and associated main memory that are used to support the execution of the database system software.

##### **Software**

This is the software, Database Management System (DBMS) that is responsible for the overall management of communications between the user and the database. It is found between the data and the users, which, in other words, means the data is entirely covered/shielded by the DBMS software. The



DBMS provides facilities for operating on the database. This is the most important software component in the overall system.

### **Users**

These are those people who are engaged on processing different types of operations on the database system and they can be categorized as:

**Application programmers** who are responsible for writing application programs that use the database using some programming language such as COBOL, Pascal, or a programming language built-in to the DBMS.

**End users** are those that interact with the system from online workstations or terminals that use an application program developed by application programmers or those that query the system through an interface provided by the DBMS.

**Database Administrator** a person that is responsible for all technical operations or details of the database system.

### **Data Independence**

Can be defined as the ability (immunity) of applications to change storage structure and access technique without modifying the main application. In older systems, the way in which the data is organized in secondary storage, and the technique for accessing it, are both dictated by the requirements of the application under consideration, and moreover that *knowledge of that data organization and that access technique is built into the application logic and code*. In such type of systems it is impossible to change the storage structure (how the data is physically stored) or access technique (how it is accessed) without affecting the application. The applications mentioned are simply programs that are designed to specific tasks where every knowledge of the data structure and the access mechanism is also defined within itself. In database systems, it would be extremely undesirable to allow applications to be data dependent. Major reasons are:

Different applications will need different views of the same data. Suppose, we have an employee data stored with (employee\_id, employee\_name, employee\_salary, and employee\_address, etc. data items), one user may need only to use the employee\_name and employee\_salary data items whereas another user require only the employee\_name and employee\_address data items. For data dependent applications, such needs will entail the change of the main application with creation of two different copies of the same application, as it would be applied by both users.

The Database Administrator (DBA) must have the freedom to change the storage structure or access technique in response to changing requirements, without having to modify existing applications. For



example, new kinds of data might be added to the database, new standards might be adopted; new types of storage devices might become available, and so on.

### **The Database Administrator**

The Database Administrator (DBA) is the person who provides the necessary technical support for a database system of an enterprise. The DBA is responsible for the overall control of the system at a technical level. The functions of the DBA include the following.

#### **Defining the conceptual schema**

Will directly participate or help on the process of identifying the content of the database, i.e., what information is to be held in the database and create the corresponding conceptual schema using the conceptual DDL.

#### **Defining the internal schema**

The DBA must also decide how the data is to be represented in the stored database and then create the corresponding storage structure definition (the internal schema) using the internal DDL (including associated mapping between the internal and conceptual schema).

#### **Liaising with users**

By communicating with users the DBA will ensure that the data they require is available, and to write (or help users write) the necessary external schemas using the applicable external DDL. Other functions include consulting on application design, providing technical education, assisting with problem determination and resolution, and similar system related professional services.

#### **Defining security and integrity rules**

Since security and integrity rules are part of the conceptual schema, the conceptual DDL should include facilities for specifying such rules.

#### **Defining backup and recovery procedures**

In the event of damage to any portion of a database, caused by human error or failure in the hardware or operating system, it is essential to be able to repair the data concerned with the minimum of delay and with as little effect as possible on the rest of the system. The DBA should define and implement appropriate backup and recovery scheme.



### **Monitoring performance and responding to changing requirements**

Periodic performance analysis should be done by the DBA and based on the results obtained propose for improved systems and or do the modifications on the existing data definitions.

#### **3.1.5. The Database Management System**

The Database Management System (DBMS) is the software that handles all access to the database. The major functions of a typical DBMS include the following:

##### **Data definition**

The DBMS must include language processor components (must have a data sub language with features for data definition and data manipulation activities) for each of the various data definition languages (DDLs). That is, it must be able to accept data definitions (external schemas, the conceptual schema, the internal schema and all associated mappings) in source form and convert them to the appropriate object form.

##### **Data manipulation**

The DBMS must include a data manipulation language (DML) processor component that would enable the DBMS must to handle requests from the user to retrieve, update or delete existing data in the database, or add new data to the database.

##### **Data security and integrity**

The DBMS must monitor user requests and reject any attempts to violate the security and integrity rules defined by the DBA.

##### **Data recovery and concurrency**

The DBMS or else some other related software component, usually called the *transaction manager* must enforce certain recovery and concurrency controls.

##### **Data dictionary**

The DBMS must provide a data dictionary function. The data dictionary contains definitions of objects in the system such as tables and table relationships and rules defined on objects.

#### **3.1.6. Database Systems Architecture**

There may be several types of architectures of database systems. However, the following architecture (ANSI/SPARC) is applicable to most modern database systems. The architecture is consists of the three levels: internal level, conceptual level and external level.



### 3.1.6.1. The External Level

The external level is the one closest to the users, i.e., it is the one concerned with the way the data is viewed by individual users.

- An external view is the content of the database as seen by some particular user (i.e., to that user the database is similar to the view he is working/accessing).
- Each external view is defined by a means of an **external schema**, which consists basically of definitions of each of the various external record types in that external view. The external schema is written using the external DDL portion of the user's data sublanguage.

### 3.1.6.2. The Conceptual Level

The conceptual level is found in between the other two. It is a representation of the entire information content of the database including the relations with one another and security and integrity rules, etc.

- It is the view of the data as it really is or by its entirety rather than as users are forced to see it by the constraints of (for example) the particular language or hardware they might be using.
- The conceptual view is defined by means of the **conceptual schema**, which is written using another DDL, the conceptual DDL of the data sublanguage in use. If data independence is to be achieved, then those conceptual DDL must not involve any considerations of storage structure or access technique. Thus there must be no reference in the conceptual schema to stored field representations, stored record sequence, indexing, hashing addressing, pointers or any other storage and access details.
- The conceptual schema includes a great many additional features, such as the security and integrity rules.

### 3.1.6.3. The Internal Level

- Is the one closest to the physical storage, i.e., it is concerned with the way the data is physically stored.
- Is a low-level representation of the entire database?
- The internal view is described by means of the **internal schema**, which not only defines the various stored record types but also specifies what indexes exist, how stored fields are represented, what physical sequence the stored records are in, and so on. The internal schema is written using yet another DDL-the internal DDL.

There will be many distinct external views, each consisting of a more or less abstract representation of some portion of the total database, and there will be precisely one conceptual view, consisting of a similarly abstract representation of the database in its entirety. Note that most users will not be interested in the total database, but only in some restricted portion of it. Likewise, there will be precisely one



internal view, representing the total database as physically stored. The following example will clarify the levels to some extent.

At the conceptual level, the database contains information concerning an entity type called *employee*. Each individual *employee* occurrence has an *employee\_number* (six characters), a *department\_number* (four characters), and a *salary* (five decimal digits).

At the internal level, employees are represented by a stored record type called *stored\_emp*, twenty bytes long. *Stored\_emp* contains four stored fields: a six byte prefix (presumably containing control information such as flags or pointers), and three data fields corresponding to the three properties of employees. In addition, *stored\_emp* records are indexed on the empno field by an index called *empix*, whose definition is not shown.

The Pascal user has an external view of the database in which employee is represented by a Pascal record containing two fields (department numbers are of no interest to this user and therefore been omitted from the view). The record type is defined according to the syntax and declaration rules in Pascal.

Similarly, the COBOL user has an external view in which each employee is represented by a COBOL record containing two fields (this time salary is not needed by this user and omitted). The record type is defined according to COBOL rules.

External view(Pascal)		External view(Cobol)
TYPE		01 EMP.
CHAR6 = STRING[6];		02 EMPNO PIC
EMPP = RECORD	X(6).	03 DEPTNO PIC
EMPNO:CHAR6;		
SALARY:REAL	X(4).	
END;		

#### Conceptual view

##### Employee

Employee\_number character (6)  
Department\_number character (4)  
Salary Numeric (5)

#### Internal view

STORED\_EMP      LENGTH=20  
PREFIX          TYPE=BYTE (6), OFFSET=0



EMP# TYPE=BYTE (6), OFFSET=6, INDEX=EMPX  
DEPT# TYPE=BYTE (4), OFFSET=12  
PAY TYPE=FULLWORD, OFFSET=16

Notice that the corresponding objects can have different names at each level. The employee number is referred to as empno in the Pascal view, as emp# in the internal view and as employee\_number in the conceptual view. In general, to define the correspondence between the conceptual view and the internal view; and the conceptual view and the external view we need an operation called **mapping**. The mappings are important, for example, fields can have different data types, field and record names can be changed, several conceptual fields can be combined into a single external field, and so on.

### 3.1.7. Database Models

A database model is a conceptual description of how the database works. It describes how the data elements are stored in the database and how the data is presented to the user and programmer for access; and the relationship between different items in the database. The DBMSs available today can be grouped into four types of models; these are the file management system, hierarchical database system, network database system, and the relational model.

Historically, the relational model was the first real description of database model based on computer science theory whereas the other models were defined later to describe databases that had already been in use for several years. The actual evolution of these databases followed the path from file management system to hierarchical database system to network database system to relational model.

### 3.1.8. The File Management System (FMS)

The FMS is the easiest data model to understand, and it is the only one that also describes how the data is actually stored on an external storage (disk, tape, etc.). In the FMS model, each field or data item is stored sequentially on disk in one large file. In order to find a particular item, the DBMS has to search the entire file from the beginning. It can also keep a *pointer* (a logical or physical indicator on the disk) to the last data item retrieved, so that searches for more occurrences of the same data type don't have to begin at the start of the file. The greatest disadvantage of the FMS is that it doesn't indicate the relationship between the various data items, other than the sequence they are stored. The search mechanism is also very slow as it employs sequential search technique.

Genet	30	Female	Ahmed	41	Male
-------	----	--------	-------	----	------

A record of two people with name, age and sex data stored sequentially, in the FMS.

The FMS was the first method used to store data in a computerized database, and its only advantage is its simplicity. Today, about the only DBMS products that are built on this model are the low end, "flat-file" databases, such as Borlands Reflex.



### 3.1.9. The Hierarchical Database System (HDS)

In this model, the data is organized in a tree structure that originates from a root, and each class of data resides at different levels along a particular branch off the root. The data structure at each class level is called a *node*. There is always a single root node which is usually 'owned' by the system or DBMS. Each of the pointers in the root then will point to (child) nodes there by depicting a parent-child sort of relationship.

Searches are done by traversing the tree up and down with known search algorithms and modules supplied by the DBMS or may, for special cases, be designed by the application programmer. The initial structure of the database must be defined by the application programmer when the database is created. From this point on, the parent-children structure can't be changed without redesigning the whole structure.

### 3.1.10. The Network Database System

The network is a conceptual description of databases where many-to-many (multiple parent-child) relationships exist. To make this model easier to understand, the relationships between the different data items are commonly referred to as sets to distinguish them from the strictly parent-child relationships defined by the HDS. The network model uses pointers to map the relationships between the different data items. The flexibility of the NDS model in showing many-to-many relationships is its greatest strength, though the flexibility comes at a price (the interrelationships between the different data sets become extremely complex and difficult to map). Like the HDS, NDSs can very quickly be searched, especially through the use of index pointers that lead directly to the first item in a set being searched. The NDS suffers from the same structural problem as the HDS; the initial design of the database is arbitrary, and once it's setup, any changes to the different sets require the programmer to create an entirely new structure. The dual problems of duplicated data and inflexible structure led to the development of a database model that minimizes both problems by making relationships between the different data items the foundation for how the database is structured.

### 3.1.11. The Relational Model (RM)

The relational model is *a way of looking at data*- that is, it is a prescription for a way of representing data (namely, by means of tables), and a prescription for a way of manipulating such data (by means of operators). More precisely, the relational model is concerned with three aspects of data: **data structure (objects)**, **data integrity**, and **data manipulation (operators)**.

The primary purpose behind the relational model is the preservation of data integrity. To be considered truly relational, a DBMS must completely prevent access to the data by any means other than queries



handled by the DBMS itself. While the relational model does not specify how the data is stored on the disk, the preservation of data integrity implies that the data must be stored in a format that prevents it from being accessed from outside the DBMS that created it. The relational model also requires that the data be accessed through programs that don't rely on the position of the data in the database. This is in direct contrast to the other database models, where the program has to follow a series of pointers to the data it wants. A program querying a relational database simply asks for the data it wants, and it's up to the DBMS to do the necessary searches and provide the answer. Searches can be sped up by creating an index on one or more columns in a table; however, the DBMS controls and uses the index. The user has only to ask the DBMS to create the index, and it will be maintained and used automatically from that point on.

The Relational model has a number of advantages over the other models. The most important is its complete flexibility in describing the relationships between the various data items. Once the tables are created and relationships defined then users can query the database on any of the individual columns in a table or on the relationships between the different tables. Changing the structure of the database objects is as simple as adding or deleting columns in a table. Creating a new table, deleting old tables, etc. are also very simple. The major decision/task that the designer of a relational database has to make concerns the definitions of the tables and their relationships in the database.

### **3.1.12. The Relational Database Systems**

The relational database systems are database systems that use/apply the relational data model. We have described in brief about this model in previous sections. The following sections will discuss in further detail the model and its building blocks the relational objects.

A relational system is a system in which the data is perceived by the user as tables; and the user uses certain operators to work on the data. The relational model is divided in three parts, having to do with *objects*, *integrity* and *operators*.

#### **Relational objects**

The relational data objects are: domains and relations.

#### **Domains**

In this model, the smallest semantic units of data are called **scalars**. Scalars cannot be decomposed (break apart) further without losing meaning. For example, a sex field may have the values F or M. But, an address column may be made to hold a woreda, kebele and house number information. In this case, the address data cannot be called a scalar as it can be further decomposed to three data units each of which can represent a separate entity.



A domain is a pool of values from which specific attributes of specific relations draw their actual values.

A domain also can be defined as a set (the same type) of scalar values. For example,

Domain of cities (Addis Ababa, Jimma, Diredawa, Lekempt, Assosa)

Domain of courses (English, Amharic, Database Management, Geography)

The domain concept is not supported by many RDBMS (Relational DBMSes). Instead a built-in data types are used. Domains can be used for refusing invalid entries and facilitating comparisons.

The data definition for domains takes the form:

**Create domain Dname definition**

To change an existing domain definition we can use:

**Alter domain Dname definition**

Domains can be removed from the system using:

**Destroy domain dname**

## Relations

A relation is made up of attributes (columns) and tuples (rows). The terms relation, tuple, and attribute are used as a 'substitute' for the more formal terms table, row, and column or table, record and field. In the discussions to follow, for the sake of simplicity, we will avoid the relational terms and in place of them use their equivalents.

A table is composed of a predefined number of columns and changeable number of rows. Each column in a table draws its values from a domain defined earlier.

The table can be viewed as a mathematical set of the table header and the table body parts.

The table header consists of a set of <column names: their respected domain names>. Where as the table body contains a set of pairs (< column name:value>). For example, if we have the following table:

Name	age	sex
Abebe	23	M
Kebede	45	M
e		
Zahra	26	F
Ahmed	53	M

The table definition requires a definition of each column and its associated domain. The domain for this table may be defined as:

**Create domain Dname character(20)**

**Create domain Dage Numeric(2)**

**Create domain Dsex character (1)**



And the table header definition will look like:

**{{(Name,Dname),(age,Dage),(Sex,Dsex)}}**

The body part of the table, will look like:

{{(<Name:'Abebe'>,<Age:23>,<Sex:'M'>),( <Name:'Kebede'>,<Age:45>,<Sex:'M'>),( <Name:'Zahra'>,<Age:26>,<Sex:'F'>),( <Name:'Ahmed'>,<Age:53>,<Sex:'M'>)}}

The data definition for relations takes the form:

Create base relation *relation\_name*  
(*column\_definition\_commalist*)  
*candidate-key-definition-list*  
*foreign-key-definition\_list*

## Properties of Relations

Relations or tables possess certain important properties and these are:

1. There are no duplicate rows (tuples) allowed in a table; in other words, there must not be two identical rows in a table. This is very important property of the relational model, for if duplicate rows are allowed in a table, then there would be no way for a program to uniquely reference a certain row in a table, thus creating an inherent problem for programming.
2. All value in a column are atomic or consist of scalar values or never a collection of several values.
3. Rows in a table are not ordered, i.e., within a table there is no inherent ordering of rows (top to bottom).
4. Columns in a table are not ordered i.e., within a table there is no inherent ordering of columns (left to right).

The last two properties mean that the operations in tables in the relational model should not depend on a specific ordering of columns or rows. In short, all rows and fields are equal in the sense that none of them has to exist in the context of others and that none is higher or lower than others in the overall data structure.

## Kinds of Relations (tables)

The following types of tables can be defined in a relational system.

### Base relations/tables

These tables are created by some data definition language command. Data in base tables do not come from any source internal to a database; instead, data must be entered into base tables manually or through

some batch data transfer process. Data in these tables are stored “permanently” in relation to the database itself.

## **Views**

A view is a named- derived table (relation) that is represented within the system purely by its definition in terms of other base tables or views. Views can be treated just like real tables.

## **Snapshots**

A snapshot is a named-derived relation like a view but unlike a view it stores its own data rather than the definition. Snapshots, also, can be treated as a base table. The snapshot can be taken as saved form of a result of some query that produces a table.

## **Query result**

This is the final output table resulting from the specified query. It may or may not be saved or have persistent existence.

## **Temporary tables**

Are tables, usually, created by the DBMS and destroyed by it at some appropriate time. This can include intermediate tables that are created when some large operation is underway and removed when the operation is finalized.

## **The Catalog**

The catalog or data dictionary contains detailed information regarding the various objects in the relational database. These are the tables, indexes, user information, data integrity and security rules, and so on. The catalog itself is made up of tables that can be manipulated as any other table in the system. In most cases, they might be kept hidden by the DBMS with the possibility of manipulating or working on them.

### **3.1.13. Relational Data Integrity**

Integrity rules are certain rules or checking mechanisms, when applied, guide the system to prohibit the entry of invalid or an acceptable (to the case in consideration) data or operations that would result in such types of data. The integrity rules may be defined at different levels: Column (field) level, row (record) level or table level or at the database level.

For example, some of the rules needed for a supplier-parts database system are shown below.

- Supplier id numbers must be of the form, Snnnn.
- Part numbers must be of the form, Pnnnn.
- Part colors must be Red, Green and White only.



- Part weights must be greater than zero.

The database definition needs to be extended to include certain integrity rules, the purpose of which is to inform the DBMS of certain constraints of the real world (such as the constraint that part weight can not be negative), so that it can prevent such undesired values from entering/occurring in the system. The DBMS may need to monitor all INSERT and UPDATE operations and reject any operation that attempts to enter the invalid entries (a negative weight).

The relational model provides two general integrity features that can be applicable to any relational database system. These are a) Candidate keys and b) Foreign keys.

### Candidate keys

Tables can contain multiple rows of data, and each row of a table in a relational system must be uniquely identified by some column or combinations of columns in that table. All columns (or combinations of columns) in a table with unique values are referred to as *Candidate keys*. Among the candidate keys found in a table one can be selected as the *primary key* of that table and all other candidate keys (other than the primary key are referred to as *alternate keys*). Keys can be simple or composite. A *simple key* is made up of one column, whereas a *composite key* is made up of two or more columns. If one or more columns of a composite key satisfy the definition of a *candidate key*, then that composite key will not be considered as a candidate key.

In most DBMS, indexes are used to implement candidate keys. Hence, the *unique indexes* found in such systems are not similar to candidate keys. Note that a system that does not have a candidate key can display strange behaviors in some circumstances.

### The Entity Integrity Rule

Specifies that no component of a candidate key or the primary key is allowed to have nulls (need to have some values.).

### Choosing among the Candidate keys

Since there may be multiple candidate keys in a table, you must make a decision as to which candidate key is to be the primary key. There is no general rule that can be applied here; you have to use your own judgement in many cases. Some rules of thumb are:

- Choose the column(s) least likely to change.
- Choose as few columns as possible.
- Choose columns that are familiar to users, if possible.

### Foreign keys

The power of a relational database system lies in the fact that rows (or records) in one table can be matched to records in other tables through the use of keys; therefore, primary keys would be largely

useless if not used for cross-referencing between tables. Primary keys are reference through *foreign keys*. A foreign key is a column in a table used to reference a primary key in another table. Take, as an example, the following tables that hold data about a company's employees and all departments in the company.

### Employees

Emp_id	Emp_name	Dep_id	Salary
A1	Abebe	D1	456.90
A2	Almaz	D2	600.00
B1	Belay	D1	677.00
A3	Ahmed	D2	600.88
G1	Genet	D3	500.00

### Departments

Dep_id	Dep_name	Budget
D1	Administration	33456.90
D2	Planning	33600.00
D3	Sales	33677.00
D4	Purchase	66600.00
D5	Construction	56600.00

Note that the Dep\_id column appears in both the employees and departments tables. In the departments table, the Dep\_id is primary key; in the employees table, however, this field is used as a foreign key. You must make sure that both foreign keys and their corresponding primary keys share a common meaning and draw their values from the same domain. Any column including, the primary key, can be a foreign key and can, also, be *simple* or *composite*. In the above example, the departments table is referred to as the *referencing* and the employees table is referred to as the *referenced/target* table.



## The Referential Integrity Rule

Along with the foreign key concept, the relational model includes the referential integrity rule. The rule says, the database must not contain any unmatched foreign key values. The term 'unmatched foreign key value' here means a foreign key value for which there does not exist a matching value of the relevant primary key in the target table. The cases to be considered are:

1. What should happen on attempt to delete the target of a foreign key reference? For example, an attempt to delete a department for which there exists at least one employee working. In general, there are two possibilities:

**Restricted** The delete operation is restricted to the case where there are no such matching records (reject otherwise).

**Cascades** The delete operation cascades to deleting all records with matching values in the referencing table (employee).

2. What should happen on attempt to update a primary key that is the target of a foreign key reference? For example, an attempt to update a department id for which there exists at least one employee. In general, there are two possibilities:

**Restricted** The update operation is restricted to the case where there are no such matching records (reject otherwise).

**Cascades** The update operation cascades to update the foreign key in those matching records in the referencing table (employees).

For each foreign key in the design, the database designer should specify, not only the columns that constitute the foreign key and the tables, but also the foreign key rule to apply when the situations occur.

### 3.1.14. The Structured Query Language (SQL)

Ideally, a database language should allow a user to create the database and relation structures; it should allow a user to perform basic data management tasks, such as the insertion, modification and deletion of data from the relations; and it should allow a user to perform both simple and complex queries to transform the raw data into information. In addition, a database language must perform these tasks with minimal user effort, and its command structure and syntax must be relatively easy to learn. Finally, it must be portable: that is, it must conform to some recognized standard so that we can use the same command structure and syntax when we move from one DBMS to another.

SQL is an example of a **transform-oriented language**, or a language designed to use relations (tables) to transform inputs into required outputs. As a language SQL has two components:

A Data Definition Language (DDL) for defining the database structure, and

A Data Manipulation Language (DML) for retrieving and updating data.



SQL contains only these definitional and manipulative commands; it does not contain flow control commands. In other words, there are no IF..THEN..ELSE, GO TO, DO ... WHILE or other commands to provide a flow of control.

## Data Definition

The SQL data definition language allows us to create or destroy database objects (schemas, domains, tables, views and indexes). The main SQL data definition language statements are:

CREATE		DROP
SCHEMA		SCHEMA
CREATE	ALTER	DROP
DOMAIN	DOMAIN	DOMAIN
CREATE	ALTER TABLE	DROP TABLE
TABLE		
CREATE VIEW		DROP VIEW

The SCHEMA is a collection of database objects that are in some way related to one another. (All objects in a database are described in one schema or another). The objects in a schema can be tables, views, domains, character sets, assertions (rules), etc. At present CREATE and DROP SCHEMA are not yet widely implemented. In some implementations, the following statement is used instead of SCHEMA.

CREATE DATABASE database\_name

So, this might be the first step before starting to create any object.

## Data definition for Domains

CREATE DOMAIN domain\_name [AS] data\_type  
DEFAULT default\_value [CHECK (search\_condition)]

A domain is given a name, domain\_name, a data type (CHAR, VARCHAR, INTEGER, NUMERIC, FLOAT, DATE, etc.)

### Examples:

CREATE DOMAIN gender AS CHARACTER (1) CHECK (VALUE IN ('M','F'))

Creates a domain called gender that consists of a single character with either the value 'M' or 'F'. When defining the Sex column we can use this domain.

DROP DOMAIN domain\_name [RESTRICT|CASCADE]

Removes domains defined in the system. In the CASCADE, any table column that is based on the domain is automatically changed to the underlying data type. RESTRICT means don't remove domain if it is used in any column definition.

DROP DOMAIN gender



## Data definition for Tables (CREATE TABLE)

Once the database structure is created we can create table structures as follows. The syntax for creating a table is:

```
CREATE TABLE table_name  
(column_name data_type [NOT NULL|NULL])  
[DEFAULT default_value][CHECK (search_condition)] ... {for each column and then after all columns  
defined}  
[PRIMARY KEY (column(s))][UNIQUE (column(s))]  
[FOREIGN KEY (column(s)) REFERENCES target_table_name [list_of_candidate_key_columns]  
[ON UPDATE referential_integrity_action]  
[ON DELETE referential_integrity_action]]  
[CHECK (search_condition)]
```

The above definition of the create table structure is similar to the one described in data definition for relational objects. Accordingly, the following examples illustrate the different ways of using this command.

### Examples

1. Create a table with fields name, age and sex.

```
CREATE TABLE newtable (name char(30) NOT NULL, age integer NOT NULL CHECK age>0, sex char(1))
```

2. Create a table with fields: name, age, sex and city, and make the default value of city 'Addis Ababa' (expecting many records will hold this value) and a rule for male members to be of age>30.

```
CREATE TABLE temp (name char(30) NOT NULL, age integer NOT NULL CHECK age>0, sex char(1) CHECK VALUE  
IN ('M','F') PRIMARY KEY (name) CHECK sex='M' AND age>30)
```

3. The following table defines a foreign key rule for the employee table whose target is the department table.

```
CREATE TABLE department (depid char(5) NOT NULL, depname char(40), depid char(5), budget float(14,2)
```

```
PRIMARY KEY (depid) UNIQUE (depname))
```

```
CREATE TABLE employee (empid char(5) NOT NULL, empname char(40), depid char(5), salary float(10,2)
```

```
PRIMARY KEY (empid) FOREIGN KEY (depid) REFERENCES department ON UPDATE CASCADE ON DELETE  
CASCADE)
```

## Changing a Table definition (ALTER TABLE)

The ALTER TABLE command in SQL is used to change the structure/definition of an existing table. It is used mainly to:

- Add a new column to a table
- Drop a column from a table
- Add a new table constraint

Drop a table constraint

Set a default for a column

Drop the default for a column

The basic format of the statement is:

ALTER TABLE table\_name

[ADD|ALTER [COLUMN] column\_name data\_type [NOT NULL][DEFAULT dvalue] [CHECK (condition)]]

[ALTER [COLUMN] column\_name data\_type]

[DROP [COLUMN] column\_name [RESTRICT|CASCADE]]

[ADD [CONSTRAINT [constraint\_name]] table\_constraint\_definition]

[DROP CONSTRAINT constraint\_name [RESTRICT|CASCADE]]

[ALTER [COLUMN] SET DEFAULT default\_value]

[ALTER [COLUMN] DROP DEFAULT]

Where the parameters are as defined in the CREATE TABLE. The table\_constraint\_definition is one of the clauses: PRIMARY KEY, UNIQUE, FOREIGN KEY OR CHECK.

#### Examples

1. Change the field width of the empid column inside the employee table.

ALTER TABLE employee ALTER COLUMN empid character (20)

2. Add a new default value of 'AA' for the empid field inside the employee table.

ALTER TABLE employee ALTER COLUMN empid SET DEFAULT 'AA'

3. Tell the system that empid is the primary key inside the employee table.

ALTER TABLE employee ADD PRIMARY KEY (empid)

#### Removing Tables (DROP TABLE)

To remove a table we use the DROP TABLE command as follows:

DROP TABLE table\_name

#### Data Manipulation Language

This section looks at the available SQL DML statements, namely:

SELECT	To query data in the database.
INSERT	To insert data into a table.
UPDATE	To update data in a table.
DELETE	To delete data from a table.

#### SELECT statement



The purpose of the SELECT statement is to retrieve and display data from one or more database tables. It is the most frequently used SQL command. The general form of the SELECT statement is:

```
SELECT [DISTINCT|ALL] {*|[[Column_expression [AS new_name]] [, .....]}
FROM table_expression
[WHERE condition]
[GROUP BY column_list] [HAVING condition]
[ORDER BY column_list]
```

*Column\_expression* represents a column or an expression. The sequence of processing in a SELECT statement is:

<b>FROM</b>	Specifies the table or tables used.
<b>WHERE</b>	Filters the rows subject to some condition.
<b>GROUP BY</b>	Form groups of rows the same column value.
<b>HAVING</b>	Filters the groups subject to some condition.
<b>SELECT</b>	Specifies which columns are to appear in the output.
<b>ORDER BY</b>	Specifies the order of the output.

The above order of the clauses in the processing of SELECT *cannot* be changed. The only two mandatory clauses are the first two: SELECT and FROM and the remainder are optional. The result of a query on a table is another table. The following examples show the different usage of this statement.

Consider the following table for the examples.

Employee

EmpId	Empname	Depid	Salary
A8	Abebe Kebede	1	1050.00
B4	Girma Belew	1	500.45
A5	Ahmed Mohammed	2	675.99
B6	Zahara Hagos	3	710.00

Department

Depid	Department	Dept_tel
1	Finance	1
2	Planning	2

## I. Column Selection

### 1. Retrieve all rows and all columns

```
SELECT empid, empname, depid, salary
FROM employee
```

Since many SQL retrievals require all columns of a table, there is a quick way of expressing 'all columns', using an asterisk (\*) in place of the column names.

```
SELECT *
```

## FROM EMPLOYEE

The result table for the above two SELECT statements is:

EmpId	Empname	Depid	Salary
A8	Abebe Kebede	1	1050.00
B4	Girma Belew	1	500.45
A5	Ahmed Mohammed	2	675.99
B6	Zahara Hagos	3	710.00
C4	Tarik Sisay	2	835.00
C5	Tadesse Belay	1	980.00

2. Retrieve specific columns, in all rows.

```
SELECT empname, salary  
FROM employee
```

This shows all employees with their name and salary.

Empname	Salary
Abebe Kebede	1050.00
Girma Belew	500.45
Ahmed Mohammed	675.99
Zahara Hagos	710.00
Tarik Sisay	835.00
Tadesse Belay	980.00

3. Using distinct (includes all those values that are unique in a particular column (by eliminating duplicate values))

```
SELECT DISTINCT depid  
FROM employee
```

Depid
1
2
3

4. Using a calculated field or an expression



```
SELECT empname, 'Yearly salary =' AS title, salary * 12 AS yearly
FROM employee
```

This command displays the following table. Note the *title* and *Yearly* column names in the following table heading.

Empname	title	Yearly
Abebe Kebede	Yearly salary =	12600.0 0
Girma Belew	Yearly salary =	6005.40
Ahmed Mohammed	Yearly salary =	8111.88
Zahara Hagos	Yearly salary =	8520.00
Tarik Sisay	Yearly salary =	10020.0 0
Tadesse Belay	Yearly salary =	11760.0 0

## II. Row selection

The above examples show the use of the SELECT statement to retrieve all rows from a table. However, we often need to restrict the rows that are retrieved. This can be achieved with the WHERE clause, which consists of the keyword WHERE followed by a search condition that specifies the rows to be retrieved. The five basic search conditions are as follows:

<b>Comparison</b>	Compare the value of one or more expression to the value of another expression using the following operators. (<, >, =, <=, >=, and, <>) and the logical operators (NOT, AND and OR).
<b>Range</b>	Test whether the value of an expression falls within a specified range (BETWEEN low, high).
<b>Set membership</b>	Test whether the value of an expression equals one of a set of values. (Using IN).
<b>Pattern match</b>	Test whether a string matches a specified pattern (name LIKE 'abebe' ).
<b>Null</b>	Tests whether a column has a null (unknown) value.

1. List all employees with a salary greater than 800

```
SELECT *
FROM employee
```

WHERE salary > 800

EmpId	Empname	Depid	Salary
A8	Abebe Kebede	1	1050.00
C4	Tarik Sisav	2	835.00

2. List all employees working in the planning department (depid=2) and with salary greater than 800

```
SELECT *  
FROM employee  
WHERE (depid=2) and (salary > 800)
```

EmpId	Empname	Depid	Salary
C4	Tarik Sisav	2	835.00

3. Range search condition (BETWEEN/NOT BETWEEN)

```
SELECT *  
FROM employee  
WHERE salary BETWEEN 500 AND 1000
```

4. Set membership search condition (IN/NOT IN)

```
SELECT *  
FROM employee  
WHERE name IN ('abebe', 'ABEBE', 'Abebe')
```

5. Pattern match search condition (LIKE/ NOT LIKE)

SQL has two special pattern matching symbols:

- % Percent character represents any sequence of zero or more characters
- \_ Underscore character represents any single character.

All other characters in the pattern represent themselves. For example,

address LIKE 'H%'	the first character must start with H, but the rest of the string can be anything.
address LIKE 'H__'	there must be exactly four characters in the string, the first of which must be an H.
address LIKE '%e'	any sequence of characters, of length at least 1, with the last character an e.
address LIKE '%ADDIS ABABA%'	a sequence characters of any length containing ADDIS ABABA.
address NOT LIKE 'H%'	the name cannot start with an H.

```
SELECT *  
FROM employee  
WHERE name LIKE 'Abebe'
```



6. Null search condition (IS NULL/IS NOT NULL)

```
SELECT *  
FROM employee  
WHERE empid IS NULL
```

### III. Sorting results (ORDER BY clause)

In general, the rows of an SQL query result table are not arranged in any particular order. However, we can sort the results of a query using the ORDER BY clause in the SELECT statement. This clause consists of a list of column identifiers that the result is to be sorted on, separated by commas.

1. List all employee ordered by empname column

```
SELECT * FROM employee ORDER BY empname
```

2. List all employee ordered by empname column (in reverse order)

```
SELECT * FROM employee ORDER BY empname DESC
```

3. List all employees by empname and those with similar names by department Id.

```
SELECT * FROM employee ORDER BY empname ASC, depid ASC
```

ASC stands for ascending sort order and DESC stands for reverse sort order.

### IV. Using the SQL Aggregate functions

The ISO standard defines five aggregate functions:

**COUNT** number of values in a specified column

**SUM** sum of values in a specified column.

**AVG** average of the values in a specified column.

**MIN** the smallest value in a specified column

**MAX** the largest value in a specified column

where COUNT(\*) is a special use of COUNT and it counts all records (rows) in a table.

It is important to note that an aggregate function can be used only in the SELECT list and in the HAVING clause. It is incorrect to use it elsewhere. If the SELECT list includes an aggregate function and no GROUP BY clause is being used to group data together then no item in the SELECT list can include any reference to a column unless that column is an argument to an aggregate function. For example, the following query is illegal:

```
SELECT empid, COUNT(salary) FROM employee is illegal.
```

because the SELECT list contains both a column name (empid) and a separate aggregate function (COUNT), without a GROUP BY clause being used. But, the following are correct.

```
SELECT COUNT (*) AS count FROM employee WHERE dep
```

count
2



Or

```
SELECT COUNT (DISTINCT depid) AS count FROM
```

count
3

1. Count the number of employees and Sum their salaries for those employees of salary < 600

```
SELECT COUNT(empid) AS count, SUM(salary) AS SUM FROM employee WHERE salary<600
```

## V. Grouping Results (GROUP BY Clause)

The above summary queries are similar to the totals at the bottom of a report. A query that uses the GROUP BY is called a grouped query, because it groups the data from SELECT table(s) and produces a single summary row for each group. The ISO standard requires that the SELECT clause and the GROUP BY clause be closely integrated. When GROUP BY is used, each item in the SELECT line must be **single-valued per group**. Further, the SELECT clause may only contain: column names, aggregate functions, constants and an expression involving combinations of the above. As an example, let us find the number of employees working in each department and sum of their salaries.

```
SELECT depid, COUNT (depid) as depid, SUM(salary) as sum
FROM employee
GROUP BY depid
```

Will give us the following table.

depid	count	sum
1	3	2530.45
2	2	1510.9

The HAVING clause is designed for use with the GROUP BY clause to restrict the (*unwanted*) groups that appear in the final result table. The ISO standard requires that column names used in the HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function. In practice, the search condition in the HAVING clause always includes at least one aggregate function, otherwise the search condition could be moved to the WHERE clause and applied to individual rows (Remember that aggregate functions can not be used with the WHERE clause). The HAVING clause is not a necessary part of SQL as any query expressed using the HAVING clause can always be rewritten without the HAVING clause. As an example, let us display, for each department with more than one employee in it, find the number of workers and sum of their salaries. Basically, this question is similar to the example given above. What we need to include is a condition that selects those departments that have only one employee working. So, the above statement can be rewritten and result displayed as follows:

```
SELECT depid, COUNT (depid) as depid, SUM(salary) as sum
```



```
FROM employee
GROUP BY depid
HAVING COUNT(depid)>1
```

Will give us the following table.

depid	count	sum
1	3	2530.45

## VI. Subqueries

Some SQL statements can have a complete SELECT statement embedded within them. An outer SQL statement can have another SQL statement that is part of the first SQL statement. The results of this *inner* SELECT or (*subselect*) statement are used in the *outer* statement to help determine the contents of the final result. A subselect can be used in the WHERE and HAVING clauses of an outer SELECT statement, where it is called a **subquery** or **nested query**. Subselects may also appear in INSERT, UPDATE and DELETE statements. Following are examples that describe each subquery type.

The general syntax of such nesting looks like:

SQL statement ... WHERE [value operator [keyword1]][keyword2] SELECT Column(s) FROM tablename, etc.

Where SQL statement can be a proper SELECT or INSERT or another statement; **operator** can be any relational operator (<, >, =, etc.) ; **keyword1** can be ALL, ANY or SOME and **keyword2** can be EXISTS/NOT EXISTS or IN or NOT IN

The second SELECT statement is what is referred to as the subquery.

### Using a subquery with relational operator

1. List the workers who work in the 'Finance' department

```
SELECT empid, empname, salary FROM employee
WHERE depid = (SELECT depid FROM department WHERE depname='Finance')
```

2. List all workers whose salary is greater than the average salary

```
SELECT empid, empname, salary FROM employee
WHERE salary > (SELECT avg(salary) FROM employee)
```

3. List all workers working in the 'Finance' department

```
SELECT empid, empname, salary FROM employee
WHERE depid IN (SELECT depid FROM department WHERE depname='Finance')
```

The IN operator checks if depid is found inside the *entire* resulting table.



2. List a worker whose salary is larger than the salary of at least one worker at the 'Finance' department (depid='1').

```
SELECT * FROM employee
```

```
WHERE salary > SOME (SELECT salary FROM employee WHERE depid='1')
```

5. List workers whose salary is larger than the salary of *every* worker at the 'Finance' department (depid='1').

```
SELECT * FROM employee
```

```
WHERE salary > ALL (SELECT salary FROM employee WHERE depid='1')
```

3. List all departments with no employee assigned in them

```
SELECT * FROM department
```

```
WHERE NOT EXISTS (SELECT * FROM employee WHERE department.depid=employee.depid)
```

You can see the opposite by dropping the NOT operator of EXISTS.

The following rules apply to subqueries:

1. The ORDER BY clause can not be used in a subquery (although it may be used in the outermost SELECT statement)
2. The subquery SELECT must consist of a single column or expression except for subqueries that use EXISTS
3. By default, column names in a subquery refer to the table in the subquery. It is possible to refer to a table in FROM clause in an outer query by qualifying the column name.
4. When a subquery is one of the two operands involved in a comparison, the subquery must appear on the right hand side of the comparison. For example, the following is **illegal**.

```
SELECT empid, empname, salary
```

```
FROM employee
```

```
WHERE (SELECT avg(salary) FROM employee) > salary is illegal
```

### Multi table Queries

All the examples considered so far are based on a single table. To combine columns from several tables we need to use the **join** operation. If we need to obtain information from more than one table, the choice is between *using a subquery* and **using a join**. If the final result table is to contain columns from different tables, then we must use join. When joining tables we need to specify a column(s) in which the join is to be based. However, we can join tables with no common columns and in such cases the join type is known as **cross join**. In this case all possible pairs of rows from the two tables will be the product of the join. Commonly, such join is not important. The other join type is the **equi-join** or **inner join** or



**natural join**, in this case, all rows from both tables that have a matching value in the common columns (in both tables) will be selected. The remaining join type is the **outer join** in which case the rows that don't have matching values in the common fields are included in the result table. For the unmatched rows each column will be filled with NULLs. To illustrate the join types consider the following tables.

**Case 1:** Cross join of table1 and table2 gives the following result table.

SELECT \* FROM table1 CROSS JOIN table2

Result(Cross join)

A	Btable 1	Btable 2	C
a1	b1	b1	c1
a1	b1	b2	c2
a1	b1	b3	c3
a2	b1	b1	c1
a2	b1	b2	c2
a2	b1	b3	c3
a3	b2	b1	c1
a3	b2	b2	c2

This join creates a table that holds 3 (rows from table1) \* 3 (rows from table2) = 9 rows. For large systems such join may take long time and also consume a lot of space.

Result(Cross join)

A	Btable1	Btable 2	C
a1	b1	b1	c1
a2	b1	b1	c1

**Case 2:** Equi-join of table1 and table2 gives the following result table.

Equi join is expressed in one of the three methods:

SELECT \* FROM table1 JOIN table2 ON table1.b = table2.b

or

SELECT \* FROM table1 JOIN table2 USING b

or

SELECT \* FROM table1 NATURAL JOIN table2

This join creates a table that holds all rows with matching values in the B column of both tables.

**Case 3:** Outer join can be LEFT JOIN or RIGHT JOIN

LEFT OUTER JOIN takes all the rows from the table at the left side and those rows with matching values in the right side table and if there is no matching value it fills the blank columns with NULL.

SELECT \* FROM table1 LEFT JOIN table2 ON table1.b=table2.b

Gives us the following result table.

Result(Left join)

A	Btable1	Btable 2	C
a1	b1	b1	c1
a2	b1	b1	c1

and the RIGHT JOIN does the opposite of the LEFT JOIN ,i.e., all rows from the right side table are included and from the left side table only those rows with matching values.

`SELECT * FROM table1 RIGHT JOIN table2 ON table1.b=table2.b`

The resulting table is shown below.

Result(Right join)

A	Btable1	Btable 2	C
a1	b1	b1	c1
a2	b1	b2	c2

Note that the result of `table1 LEFT JOIN table2` is identical to `table2 RIGHT JOIN table1`

The table names in the FROM clause can be renamed to any other name (alias) and that name can be used elsewhere inside the SELECT statement. For example,

`SELECT Temp.a,temp.b FROM table1 AS temp`

or

`SELECT T1.a, T2.b FROM table1 AS T1 LEFT JOIN table2 AS T2 ON T1.b=T2.b`

## Database Updates

SQL is a complete data manipulation language that can be used for modifying the data in the database as well as querying the database. In the following sections we will discuss the INSERT, UPDATE and DELETE statements which are designed for data update. The commands for modifying the database are not as complex as the SELECT statement. Therefore, we will present only the syntax and brief examples on each of them.

### Adding data to the database (INSERT)

Allows us to insert new values into an existing table.

The syntax for INSERT is:

`INSERT INTO table_name [(column_list)] VALUES (corresponding_data_value_list )`

or

`INSERT INTO table_name [(column_list)] SELECT ...`

In this case, the SELECT statement is going to provide the data values.

**Examples:**



```
INSERT INTO employee (empid, empname, depid, salary) VALUES ('1', 'Ameneshewa', '2', 456.89)
```

or

```
INSERT INTO employee VALUES ('1', 'Ameneshewa', '2', 456.89)
```

This assumes that all column values are provided and the data types are the same for each column. The number of values must be equal to the number of columns in the table.

Suppose that there is another table called oldEmployee and you need to transfer data from this table into the employee table, you can use the second syntax of INSERT as follows:

```
INSERT INTO employee (SELECT * FROM oldEmployee WHERE oldemployee.empid IS NOT NULL)
```

### **Modifying data in the Database (UPDATE)**

Allows us to change data in an existing table. Its format is:

```
UPDATE table_name  
SET column_name1=data_value1 [,column_name2=data_value2 ...]  
[WHERE search_condition]
```

#### **Examples**

Update all salary by increasing 10% of the salary of each employee.

```
UPDATE employee SET salary = salary + salary * .1
```

Give the 10% salary increment only to those workers in the Finance department (depid='1').

```
UPDATE employee SET salary = salary + salary * .1 WHERE depid='1'
```

### **Deleting data in the Database (DELETE)**

The DELETE statement allows you to delete rows from an existing table. The format of the command is:

```
DELETE FROM table_name [WHERE search_condition]
```

#### **Examples**

Delete all workers working in the planning department (depid='2').

```
DELETE FROM employee WHERE depid='2'
```

Delete all records inside the oldEmployee table.

```
DELETE FROM oldEmployee
```

### **3.1.15. Database Design Fundamentals**

#### **Database Design**

Database design can be defined in a loose way as a process that involves decision on what it contains (what tables, columns, rules, etc.) and how (the DBMS to be used, the programs needed, the security procedures, etc.) it will be used. It needs a design methodology which consists of phases that contain steps which guide the designer in the techniques appropriate at each stage of the project, and also help to plan, manage, control and evaluate the database development projects.

There are two phases of the database design process, namely, Logical database design and Physical database design.

### **Logical Database Design**

Logical database design is concerned with identifying the data (that is, entities and fields), the relationship between the data and constraints (rules) of the data that is to be stored in the database. The logical database designer must have a thorough and complete understanding of the organization's data and the **business rules**. Business rules describe the main characteristics of the data as *viewed by the organization*.

To be effective the logical database designer must involve all prospective database users in the development of the data model, and the involvement should begin as early in the process as possible. Logical database design is independent of implementation details such as the target DBMS, application programs, programming languages or any other physical considerations. The steps for the logical designer are:

#### **Build conceptual data model from user view**

1. Identify entity types
2. Identify relationship types
3. Identify and associate fields with entity or relationship types
4. Determine field domains
5. Determine candidate and primary keys
6. Draw entity-relationship diagram
7. Review the conceptual data model with user

#### **Build and validate logical data model**

8. Map the conceptual model to a logical model
9. Derive relations from the logical data model
10. Validate model using normalization



11. Validate model against user transactions
12. Draw entity-relationship diagram
13. Define integrity constraints
14. Check for future growth

## Physical Database Design

The physical database design takes the logical data model (the result of the logical database design phase) and decides how it is to be physically realized. This involves:

- Mapping the logical data model into a set of tables and integrity constraints(rules);
- Selecting specific storage structures and access methods for the data to achieve good performance for the database activities;
- Designing any security measures required on the data.

Many parts of physical database design are highly dependent on the target DBMS, and there may be more than one way of implementing a mechanism. Consequently, the physical database designer must be fully aware of the functionality of the target DBMS and must understand the advantages and disadvantages of each alternative for a particular implementation. The steps required are:

1. Translate the logical data model for target DBMS: design base tables and integrity rules for target DBMS
2. Design and implement physical representation: analyze transactions, choose file organization and secondary indexes, and estimate disk space.
3. Design and implement security mechanisms: design user views and access rules
4. Monitor and tune the operational system

The logical database designing process uses two important concepts, i.e., normalization and the entity-relationship model. The following sections will deal with these concepts.

### Normalization

A major aim of the relational database design is to group fields into tables so as to minimize information redundancy and thereby reduce the file storage space required by the base tables. A serious difficulty using tables that have redundant information is the problem of *update anomalies*. These are problems that we encounter at the time of insertion or deletion or updating of a particular information in such tables. To avoid these anomalies one would require normalizing all the tables in the system.

Normalization is a technique for analyzing tables based on their primary key (or candidate keys in the case of BCNF (to be discussed later)). The technique, each normal form, involves a set of rules that can



be tested against individual tables. When a requirement is not met, the table violating the requirement must be decomposed (split) into tables that individually meet the requirements of normalization. Each higher form is based on the previous one; and when applied in sequence, the forms progressively make the database more efficient. In relational database design, it is important to recognize that it is only the first normal form (1NF) that is critical in creating appropriate tables. All of the subsequent normal forms are optional.

Before discussing the normalization process let us introduce a concept, known as functional dependency, which is important for the foregoing discussions.

### Functional dependency

One of the main concepts associated with normalization is functional dependency. A functional dependency describes the relationship between attributes (fields) in a relation (table). Suppose we have two fields A and B in a table, called T,; and if each value of A inside the table, T, is associated with exactly one value of B in the same table, we say,

B is functionally dependent on A, or pictorially:

$$A \rightarrow B$$

And A is called the **determinant** of the functional dependency A to B.

The fields A and B can each of them be a single field or group of fields. As an example, let us consider the following table.

Staff\_branch

Staff_no	Sname	Saddress	Position	Salary	Branch_no	Baddress	Tel_no
SL21	Abebe belew	W20 K10 Hno 190	Manager	30000	B5	W20 K10 Hno 190	5
SG37	Ahmed Mohammed	W1 K10 Hno 190	Sr Assistant	12000	B3	W11 K22Hno 345	3
SG14	Genet Adam	W11 K22Hno 345	Deputy	18000	B3	W11 K22Hno 345	3
SA9	Zahara Yusuf	W24 K12Hno 23	Assistant	9000	B7	W1 K20 Hno 190	7
SG5	Mulatu Tasew	W2 K09 Hno 766	Manager	24000	B3	W11 K22Hno 345	3
SL41	Almaz Abebe	W2 K15 Hno 987	Assistant	9000	B5	W20 K10 Hno 190	5

The functional dependencies of the Staff\_branch table are:

Staff\_no  $\rightarrow$  Sname      Staff\_no  $\rightarrow$  Saddress      Staff\_no  $\rightarrow$  Branch\_no  
 Staff\_no  $\rightarrow$  Salary      Staff\_no  $\rightarrow$  Position      Staff\_no  $\rightarrow$  Baddress      Staff\_no  $\rightarrow$  Tel\_no  
 Branch\_no  $\rightarrow$  Baddress      Branch\_no  $\rightarrow$  Tel\_no  
 Baddress  $\rightarrow$  Branch\_no      Tel\_no  $\rightarrow$  Branch\_no



There are 11 functional dependencies in the Staff\_branch table and relations with Staff\_no, Branch\_no, Baddress and Tel\_no as determinants. An alternative format for displaying such functional dependencies is shown below.

Staff\_no → Sname, Saddress, Position, Salary, Branch\_no, Baddress, Tel\_no

Branch\_no → Baddress, Tel\_no

Baddress → Branch\_no

Tel\_no → Branch\_no

Note that the only candidate key for this table is the Staff\_no field.

### First Normal Form (1NF)

In a table, if the intersection of each row and column contains one and only one value. Then that table is called in the first normal form, 1NF normalized.

The 1NF dictates the following:

- Fields contain only scalar values, not array. As an example for this case, we can cite the names or proper names of people. Such as 'Ato Abebe' or 'Abebe Kebede' stored as a name in a table or if we have address data to be stored and we put the Woreda, Kebele and House no info in a single field.
- There can be only one value per column-row position (field) in a table. For example, in the following table except the owner field all the other fields are made to hold multiple values. Abebe's record has plate nos; 01-2277 and 03-3556, colors: red and Marine blue, and models: VW beetles and Alfa Romeo.

CarOwners

Owner	Plate_no	Color	Engine no	Model
Abebe	03-3556	Red	477-789	VW beetles
	01-2277	Marine blue		Alfa Romeo (78)
Ahmed	01-3567	White	660-8799	VW Variant
Tasew	02-6689	Red	999	
	03-17889		63445-54	Fiat N3 682

To make this table in 1NF we may need to enter an owner value in the *owner* field for every car. We may, also, split this table into two tables: one that holds owner and plate numbers and another table that holds plate number, color, engine no and model of each car.

- There can be no repeating groups. A repeating group is a field or group of fields within a table that occurs with multiple values for a single occurrence of that field. To elaborate this case, we may redesign the previous table in to the following form.



SecondCarOwners

Owner	Plate_no1	Plate_no2	Color1	Color2	Engine no1	Engine no2	Model1	Model2
Abebe	03-3556	01-2277	Red	Marine blue	477-789		VW beetles	Alfa Romeo (78)
Ahmed	01-3567	01-3567	White	White	660-8799	660-8799	VW Variant	VW Variant
Tasew	02-6689	03-17889	Red		999	63445-54		Fiat N3 682

The repeating groups here are the repeating fields: plate\_no1, plate\_no2, color1, color2, etc. To normalize a table to 1NF, we identify and remove repeating groups. Therefore, to normalize this table into 1NF we can employ one of the above methods. The general procedure for normalizing a table is to decompose (break) the table into another tables that would satisfy that particular normal form. The process of

Owners

Owner	Plate_no
Abebe	03-3556
Abebe	01-2277
Ahmed	01-3567
Tasew	02-6689

Cars

Plate_no	Color	Engine no	Model
03-3556	Red	477-789	VW beetles
01-2277	Marine blue		Alfa Romeo (78)
01-3567	White	660-8799	VW Variant

decomposition should not incur any data loss. That is, the normalized tables must have the entire data in the un-normalized table. So, the above table can be decomposed into Owners and Cars table shown below both of them in 1NF.

Note that the Plate\_no is the primary key field in both tables.

## Second Normal Form (2NF)

The second form (2NF) applies to tables with composite primary key (the primary key is not a single field rather it is composed of two or more fields).

A table is in 2NF if it is in 1NF and every non-primary key field is fully functionally dependent on the primary key.

If A and B are fields in a table, B is fully functionally dependent on A if B is functionally dependent on A, but not any proper subset of A. In other words, a functional dependency  $A \rightarrow B$  is full functional dependency if removal of any field from A results in the dependency not being sustained any more.

For example, consider the following functional dependency:

$$(\text{Staff\_no}, \text{Sname}) \rightarrow \text{Branch\_no}$$

It is correct to say that each value of (Staff\_no, Sname) is associated with a single value of Branch\_no. However, it is not a full functional dependency because Branch\_no is also functionally dependent on a subset of (Staff\_no, Sname). In other words, Branch\_no is fully functionally dependent on **only** Staff\_no. Let us consider the following table.

Patients

PatientId	RelativeId	Relationship	Patient_tel
-----------	------------	--------------	-------------



249	GGP001	FATHER	123
249	GGP002	GUARDIAN	123
249	GGP003	MOTHER	123
803	PDE1	BROTHER	789
803	PDE2	UNCLE	789
984	AGE1	SISTER	421
984	AGE2	MOTHER	421

It is clear that this table is in 1NF. And, the primary key for this table is the composite key (PatientId, RelativeId). So, to determine if it satisfies 2NF, you have to find out if all other fields in it depend fully on both PatientId and RelativeId; that is, you need to decide whether the following conditions are true:

(PatientId, RelativeId)  $\rightarrow$  Relationship; and (PatientId, RelativeId)  $\rightarrow$  Patient\_tel.

However, on the dependencies in the patient table, only the following are true:

(PatientId, RelativeId)  $\rightarrow$  Relationship; and (PatientId)  $\rightarrow$  Patient\_tel.

In other words, Patient\_tel depends only on PatientId not on the composite primary key consisting of both PatientId and RelativeId. Therefore, table Patients is not in 2NF. Now, consider the cases, how can we insert the fact that a patient called Abebe has telephone number, 3445. This is not possible unless that the patient has some relative, otherwise the RelativeId field is going to store a null value and violate the definition of a primary key (any component of the primary key is allowed to have a null value).

In order to normalize table Patients to 2NF we can break it into two normalized tables. The Patient\_tel field really doesn't belong to Patients table because the patients' telephone numbers have nothing to do with patients' relatives and should be associated with patients only. To decompose this table, remove the

Patients			Patient info	
PatientId	RelativeId	Relationship	PatientId	Patient tel
249	GGP001	FATHER	249	123
249	GGP002	GUARDIAN	803	789
249	GGP003	MOTHER	984	421
803	PDE1	BROTHER		
803	PDE2	UNCLE		
984	AGE1	SISTER		

Patients\_tel field from the original table, place it in another table, say Patient\_info, and put the rest of the information in Patients table, using PatientId as the joining field between Patient\_info and Patients tables.

The two resultant tables are shown below.

As we can simply verify, the above two tables are in 2NF.

### Third Normal Form (3NF)

A table is in 3NF if and only if it is 2NF, and no non-primary key field is transitively dependent on the primary key.

Let us consider the following table.

Employee1

EmpId	Empname	Department	Dept_tel
A8	Abebe Kebede	Finance	1
B4	Girma Belew	Finance	1
A5	Ahmed Mohammed	Planning	2
B6	Zahara Hagos	Administrati on	3
C4	Tarik Sisay	Planning	2
C5	Tadesse Belay	Finance	1

The primary key of this table is EmpId. Assuming that Empname holds scalar values, this table is in 1NF and also 2NF.

Moreover, the fields: Empname and Department are all directly associated with EmpId, the primary key. The last field, Dept\_tel, however, contains the telephone number of departments and therefore is determined by the department, which is not part of the primary key. In short, the following holds true in this table:

EmpId  $\rightarrow$  Department and Department  $\rightarrow$  Dept\_tel

These dependencies can be put together to show the fact that the following transitive dependency holds true.

EmpId  $\rightarrow$  Department  $\rightarrow$  Dept\_tel

The normalization of 2NF tables to 3NF involves the removal of transitive dependencies. We remove the transitively dependent field(s) from the table by placing the field(s) in a new table along with a copy of the determinant(s).

Therefore, the above table can be decomposed into two 3NF tables shown below.

Employee2

EmpId	Empname	Department
A8	Abebe Kebede	Finance
B4	Girma Belew	Finance
A5	Ahmed Mohammed	Planning
B6	Zahara Hagos	Administration
C4	Tarik Sisay	Planning
C5	Tadesse Belay	Finance

Department

Department	Dept tel
Finance	1
Planning	2
Administrati	3

Transitive dependencies could result in various insertions, update or delete anomalies.



### Insertion Anomalies

Suppose a new department has just been created, but the company hasn't hired anyone for this new department. An error would occur if you attempted to add data into the table because there is no EmpId associated with the new department. Since EmpId is the primary key, you can't add a new record into the table with EmpId being null. In the normalized table the new directory data can be inserted into the Department table with no problem.

### Deletion Anomalies

A deletion anomaly can occur to the Employee1 table if, for example, Zahara Hagos leaves the company and her record is deleted from the Employee1 table. Because Zahara Hagos is the only member of the Administration department, all information associated with the Administration department will be wiped out even though the department itself has not been eliminated.

### Update Anomalies

An update anomaly could occur to the UN-normalized employee table (Employee1) if the Finance department changes the current telephone number to a new one. You would have to update three records in the table even though one piece of information about a department has changed. However, in the normalized table, Department, you need to change only once, i.e., the Dept\_tel data of the record that holds Finance as department.

### Boyce-Codd Normal Form (BCNF)

The Boyce-Codd Normal Form is an extension to the 3NF for the special case where:

- ◆ There are at least *two candidate keys* in the table,
- ◆ All the candidate keys *are composite keys*, and
- ◆ There is *overlapping field(s)* in the candidate keys (there is at least one field in common).

When a table satisfies all these conditions, the 3NF can't eliminate all forms of transitive dependency. For a table to be in the BCNF, it must be in the 3NF, and all fields in all its candidate keys must be *functionally independent*. Violation of the BCNF is quite rare, it may only happen under the above conditions.

The following table shows a case in which the BCNF is not fulfilled. Suppose that the following conditions apply to the table below. For each subject, each student of that subject is taught by only one teacher, and each teacher teaches only one subject (but each subject is taught by several teachers).

Student\_subject\_lecturer



Student	Subject	Teacher
Abebe	Maths	Dr. Estifanos
Mulat	Physics	Dr. Tadesse
Ahmed	Maths	Dr. Estifanos
Zahara	Physics	Dr. Jemal

The candidate keys for this table are (Student, Subject) and (Student, Teacher). Observe here that both candidate keys are *composite* and the Student field is the *overlapped* field in the candidate keys. This implies that the table is not in BCNF. Moreover, this table suffers from certain update anomalies. For

Student_Teacher		Teacher_Subject	
Student	Teacher	Teacher	Subject
Abebe	Dr. Estifanos	Dr. Tadesse	Physics
Mulat	Dr. Tadesse	Dr. Estifanos	Maths
Ahmed	Dr. Estifanos	Dr. Jemal	Physics

example, if we wish to delete the information that Zahara is studying Physics, we cannot do so without at the same time losing the information that Dr. Jemal teaches Physics. Normalizing this table to BCNF would require the decomposition of the above table into two tables:

Here is also another table that violates the BCNF. Assume that the employee name is unique.

Employee\_Overtime

EmpId	Empname	Overtime month	OT Hours
A8	Abebe Kebede	January	5
A8	Abebe Kebede	April	10
A5	Ahmed Mohammed	January	2
B6	Zahara Hagos	April	3
A5	Ahmed Mohammed	March	2

The candidate keys are, (EmpId, Overtime\_month) and (Empname, Overtime\_month) and the overlapping field is the Overtime\_month. Normalizing this table to BCNF requires this table to be decomposed in to two tables having the following fields. Table1 having the EmpId and Empname, and Table2 having EmpId, Overtime\_month, and Othours fields.

## Entity Relationship Model

The Entity-Relationship (ER) model is a high-level conceptual data model that facilitates database design. A conceptual data model is a set of concepts that describe the structure of a database and the associated retrieval and update transactions in the database. The main purpose for developing a high-level data model is to support a user's perception of data, and conceal the more technical aspects associated



with database design. Furthermore, a conceptual data model is independent of the particular DBMS and hardware platform that is used to implement the database.

## The Concepts of the Entity Relationship Model

The basic concepts of the ER model include *entities*, *relationships* and *attributes*.

### Entity

An object or concept that is uniquely identifiable is known as an **Entity**. For example, Staff, property, customer, part, supplier and product are entities with physical existence. And, Sale, inspection, viewing and work-experience are entities with conceptual existence. Entities can be classified as weak and strong types.

**Weak entity:** an entity that is **existence-dependent** on some other entity. In other words, a weak entity is dependent on the existence of another entity. For example, dependents of members of staff are weak entities because they can not exist in the model if the corresponding member staff does not exist.

**Strong entity:** an entity that is **not existence-dependent** on some other entity, e.g., Employee and Branches.

### Attributes

A property of an entity or a relationship type is called an attribute. For example, Branch entity may be described by the branch number (Branch\_no), address (Baddress), phone number (Tel\_no) and fax number (Fax\_no). The attributes of an entity hold values that describe each entity. The values held by attributes represent the main part of the data stored in the database. Each attribute is associated with a set of values called a domain. Attributes can be classified as:

**Simple:** an attribute composed of a single component with an independent existence. Simple attributes cannot be further subdivided, for example, Sex, Age, and Salary.

**Composite:** an attribute composed of a multiple components, each with an independent existence. Composite attributes can be further subdivided, for example, Address (Woreda, Kebele and House no) or Full name (name and father name).

**Single valued:** an attribute that holds a single value for a single entity. The majority of attributes are single valued.

**Multi-valued:** an attribute that holds multiple values for a single entity. A telephone number attribute can have more than one values in it (e.g., 112233 and 123344).



**Derived:** an attribute that represents a value that is derivable from the value of a related attribute or set of attributes, not necessarily in the same entity. For example, Age is derivable from Date of Birth, or Tax derived from Salary.

**Keys** are data item that allows us to uniquely identify individual occurrences of an entity type.

**Candidate key:** an attribute or set of attributes that uniquely identifies individual occurrences of an entity type.

**Primary key:** one of the candidate keys selected to be a primary key.

**Composite Key:** candidate key that consists of two or more attributes.

## Relationship

A relationship is an association of entities where the association includes one entity from each participating entity type.

**Attributes on relationships:** attributes can be assigned to relationships.

**Degree of a relationship** the number of participating entities in a relationship. A relationship of degree two is called **binary**. The next higher being ternary, quaternary, etc.

**Cardinality ratio** determines the number of possible relationships for each participating entity. The most common degree for relationships is binary and the cardinality ratios (also called relationship types) for binary relationships are one-to-one(1:1), one-to-many (1:M), and many-to-many (M:M) .

## Entity Relationship Diagrams (ER-Diagrams)

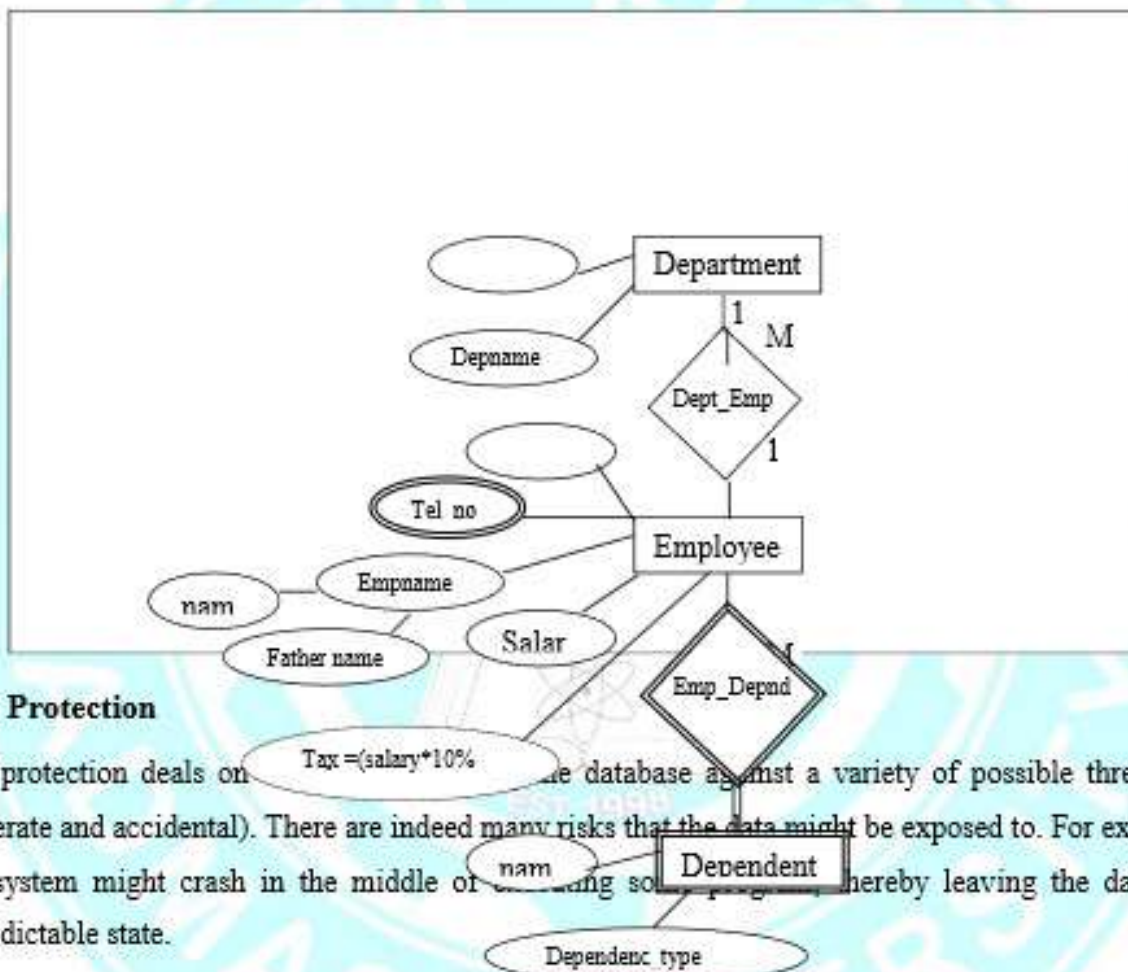
ER-Diagrams constitute a technique for representing the logical structure of a database in a pictorial manner. It provides a simple and readily understood means of communicating significant features of the design of any given database. The following sections describes the rules for constructing an ER diagram.

**Entities:** each entity type is shown as a rectangle, labeled with the name of the entity type in question. For weak entity the rectangle is doubled.

**Attributes** are shown in ellipses, with the name of the attribute written inside the ellipse, and attached to the relevant entity or relationship by means of a continuous line. The ellipse is dotted if the attribute is derived, doubled if the attribute is multi-valued. If the attribute is composite, its component attributes are shown as further ellipses, connected to the ellipse in question by a continuous line. Key attributes are underlined.



**Relationships:** each relationship type is shown as a diamond, labeled with the name of the relationship. The diamond is doubled if the relationship involves a weak entity type. The participants in each relationship are connected to the relevant relationship by means of continuous lines: each such line is labeled '1' or 'M' to indicate whether the relationship is one-to-one, many-to-one or many-to-many. The following diagram shows an ER diagram for Department-Employee-Dependents tables. Please verify the different types of entities, attributes and relationships discussed above.



### Data Protection

Data protection deals on the database against a variety of possible threats (both deliberate and accidental). There are indeed many risks that the data might be exposed to. For example, The system might crash in the middle of a transaction, thereby leaving the database in unpredictable state.

- Two programs executing concurrently might interfere with one another's operation, thereby producing incorrect results.
- Sensitive data might be exposed to – or, worse, changed by – unauthorized user.
- Updates might change the database in an invalid way.

And so on (the possibilities are endless). The system therefore has to provide an extensive set of controls to protect the database against such threats, specifically, **recovery, concurrency and security** controls. We will examine in brief each of these issues in the next section.

## Data Recovery

Recovery in a database system means, primarily, recovering the database itself, i.e., restoring the database to a state that is known to be correct after some failure has rendered the current state incorrect, or at least suspect. The underlying principles on which such recovery is based on can be generalized to application of redundancy at different levels and functions. To make sure that the database is recoverable is to make sure that any piece of information it contains can be reconstructed from some other information stored, redundantly, somewhere else in the system. There are many different types of failures that can affect database processing, each of which has to be dealt with in a different manner. Among the causes of failure are:

- ◆ **System crashes** due to hardware or software errors, resulting in loss of main memory;
- ◆ **Media failures**, such as head crashes or unreadable media, resulting in the loss of parts of secondary storage;
- ◆ **Application software errors**, such as logical errors in the program that is accessing the database, which cause one or more transactions to fail;
- ◆ **Natural physical disasters**, such as fires, floods, earthquakes or power failures;
- ◆ **Carelessness** or unintentional destruction of data or facilities by operators or users;
- ◆ **Sabotage** or intentional corruption or destruction of data, hardware or software facilities.

A DBMS should provide the following facilities to assist with data recovery:

- ◆ A backup mechanism, which makes periodic backup copies of the database;
- ◆ Logging facilities, which keep track of the current state of transactions and database changes;
- ◆ A checkpoint facility, which enables updates to the database which are in progress to be made permanent;
- ◆ A recovery manager, which allows the system to restore the database to a consistent state following a failure.

The recovery mechanisms employed can be categorized into three, transaction recovery, system recovery, and media recovery.

## Transaction Recovery

Transaction recovery means recovering the database after some individual transaction has failed for some reason and left the database in an incorrect state. It is important, here, to discuss what transactions are and how they are dealt with before looking at the recovery scheme.



## Transactions

A transaction is an action or series of actions, carried out by a single user or application program, which accesses or changes the contents of the database. A transaction is a logical unit of work on the database. It may be an entire program, a part of a program or a single command (for example, the SQL command INSERT or UPDATE), and it may involve any number of operations on the database.

A transaction can have one of two outcomes (results). If it completes its operation successfully, the transaction is said to have **committed** and the database reaches a new consistent state. On the other hand, if the transaction does not execute successfully, the transaction is aborted. If a transaction is aborted, the database must be restored to the consistent state it was in before the transaction started. Such a transaction is **rolled back** or undone. A committed transaction can not be aborted.

The DBMS has no inherent way of knowing which operations are grouped together to form a single logical transaction. The keywords **Begin transaction**, **commit** and **Rollback** (or their equivalent) are available in most DBMS to delimit transactions. If these keywords are not used the entire program is usually regarded as a single transaction.

## Properties of Transactions

There are properties all transactions should possess. The four basic, so-called ACID, properties of a transaction are:

**Atomicity** the 'all or nothing' property. A transaction is an indivisible property that is either performed by its entirety or it is not performed at all.

**Consistency** A transaction must transform the database from one consistent state to another consistent state.

**Independence** Transactions execute independently of one another. In other words, the partial effects of incomplete transactions should not be visible to other transactions.

**Durability** The effects of a successfully completed (committed) transaction are permanently recorded in the database and must not be lost because of subsequent failure.

It follows, from the above discussions on transactions, that transactions are not only the unit of work but also units of **recovery**. For if a transaction successfully commits, then the system will guarantee that its updates will be permanently installed in the database, even if the system crashes the very next moment. For such recovery to be possible the system maintains a **log** or **journal** on tape or (more commonly) disk, on which details of all update operations (before and after values of the updated object) are recorded. Thus, if it becomes necessary to undo some particular update, the system can use the corresponding log entry to restore the updated object to its previous value (It follows that the log must be physically written



before **Commit** processing can complete. This important rule is known as the **write-ahead log rule**). The log file can be used to reconstruct the database at the restarting point after a system crash has occurred

### System Recovery

Involves the recovery on system failure (e.g., power failure), which affect all transactions currently in progress but do not physically damage the database.

The critical point regarding system failure is that the *contents of main memory are lost* (in particular database buffers are lost). The precise state of any transaction that was in progress at the time of the failure is therefore no longer known; such a transaction can therefore never be successfully completed, and so must be *undone* (rolled back) when the system restarts. Here, also, the log file is the important tool to reconstruct (undo) those uncompleted transactions.

### Media Recovery

Media recovery involves the procedures or ways applied to recover the database from media failure. Media failure is such as a disk head crash, or a disk controller failure, in which some portion of the database has been physically destroyed. Recovery from such a failure involves reloading (or *restoring*) the database from a backup copy (or *dump*), and then use the log redo all transactions that completed since the backup copy was taken. There is no need to undo transactions that were still in progress at the time of failure, since by definition all updates of such transactions have been 'undone' (actually lost) any way.

### Concurrency

The issue of concurrency applies only to a multi-user system. The term concurrency refers to the fact that DBMSs typically allow many transactions to access the same data at the same time. In such a system some kind of **concurrency control** is needed to ensure that the concurrent transactions do not interfere with each other's operation. There are three problems related to concurrency, these are:

- ◆ The lost update problem
- ◆ The uncommitted dependency, and
- ◆ The inconsistent analysis problem

#### The lost update problem

An apparently successfully completed update operation by one user can be overridden by another user. Consider the situation shown below. Transaction T1 is executing concurrently with transaction T2. T1 is withdrawing 10 from an account with balance balx, initially 100, and T2 is depositing 100 into the same



account. If these transactions are executed **serially**, one after the other with no interleaving operations, the final balance would be 100 no matter which transaction performed first.

Time	T1	T2	balx
t1		Begin transaction	100
t2	Begin transaction	read(balx)	100
t3	read(balx)	balx=balx+100	100
t4	balx=balx-10	write(balx)	200
t5	write(balx)	Commit	90
t6	Commit		90

Transactions T1 and T2 start at nearly the same time, and both read the balance as 100. T2 increases the balance by 100 to 200 and stores the update in the database. Meanwhile, T1 decrements its copy of the balance, balx, by 10 to 90 and stores this value in the database, overwriting the previous update, and thereby 'losing' the 100 previously added to the balance. The loss of T2's update is avoided by preventing t1 from reading the value of balx until after T2's update has been completed.

### The uncommitted dependency problem

This problem occurs when one transaction is allowed to see the intermediate results of another transaction before it has committed. Following table shows an example of uncommitted dependency. Using the same initial value in the previous example, here, transaction T4 updates balx to 200, but it rolled back the transaction so that balx should be back at the original value of 100. However, by this time, transaction T3 has read the new value of balx (200), and using this value as the basis of the 10 reduction, giving a new incorrect balance of 190, instead of 90.

Time	T3	T4	balx
t1		Begin transaction	100
t2		read(balx)	100
t3		balx=balx+100	100
t4	Begin transaction	write(balx)	200

t5	read(balx)		200
t6	balx=balx-10	Rollback	100
t7	write(balx)		190
t8	Commit		190

The cause of this problem is the assumption by T3 that T4's update completed successfully, although the update was subsequently rolled back. The problem is avoided by preventing T3 from reading balx until after the decision has been made to either commit or rollback T4's effects.

### The inconsistent analysis problem

The above two problems concentrate on transactions that are updating the database and their interference may corrupt the database. However, transactions that only read the database can also produce inaccurate results, if they are allowed to read partial results of incomplete transactions that are simultaneously updating the database.

The problem of inconsistent analysis occurs when a transaction reads several values but a second transaction updates some of them during the execution of the first. From the example below, T6 is executing concurrently with T5. T6 is totaling the balances of account x(100), account y(50) and account z(25). However, in the meantime, T5 has transferred 10 from balx to balz, so that T6 now has the wrong result (10 too high).

Time	T5	T6	balx	baly	balz	sum
t1		Begin transaction	100	50	25	
t2	Begin transaction	Sum=0	100	50	25	0
t3	read(balx)	read(balx)	100	50	25	0
t4	balx=balx-10	Sum = Sum+balx	90	50	25	100
t5	write(balx)	read(baly)	90	50	25	100
t6	read(balz)	Sum = Sum+baly	90	50	25	150
t7	balz=balz+10		90	50	25	150
t8	write(balz)		90	50	35	150
t9	Commit	read(balz)	90	50	35	150



t10		Sum =	90	50	35	185
		Sum+balz				
t11		Commit	90	50	35	185

This problem may be avoided by preventing T6 from reading balx and balz until after T5 has completed its updates.

## Concurrency Controls

There are two basic concurrency control techniques that allow transactions to execute safely in parallel subject to certain constraints: locking and timestamping methods.

### Locking

Locking is process used to control concurrent access to data. In this method, a transaction must claim a **read** (*shared*) or **write** (*exclusive*) lock on a data item before the corresponding database read or write operation. The **lock** prevents another transaction from modifying the item or even reading it, in the case of a write lock. Data items of various sizes, ranging from the entire database down to a field, may be locked. Locks are used in the following way:

- ◆ Any transaction that needs to access a data item must first lock the item, requesting a read lock for read only access and a write lock for both read and write access.
- ◆ If the item is not already locked by another transaction, the lock will be granted.
- ◆ If the item is currently locked, the DBMS determines whether the request is compatible with the existing lock. If a read lock is requested on an item that already has a read lock on it, the request will be granted; otherwise, the transaction must **wait** until the existing write lock is released.
- ◆ A transaction continues to hold a lock until it explicitly releases it either during execution or when it terminates (rollbacks or commits). It is only when the write lock has been released that the effects of the write operation will be made visible to other transactions.

Now we are in a position to see how the locking strategy solves the three problems. Again we consider the at a time.

### Locks and the lost update problem

To prevent the lost update problem, occurring, T2 first requests a write lock on balx. It can then proceed to read the value of balx from the database, increment it by 100 and write the new value back to the database. When T1 starts, it also requests a write lock on balx. However, because the data item balx is currently write locked by T2, the request is not immediately granted and T1 has to **wait** until the lock is released by T2.

Time	T1	T2	balx
------	----	----	------

t1		<b>Begin transaction</b>	100
t2	<b>Begin transaction</b>	Write_lock(balx)	100
t3	Write_lock(balx)	read(balx)	100
t4	<b>Wait</b>	balx=balx+100	100
t5	<b>Wait</b>	write(balx)	200
t6	<b>Wait</b>	<b>Commit</b>	200
t7	<b>Wait</b>	Unlock(balx)	200
t8	read(balx)		200
t9	balx=balx-10		190
t10	write(balx)		190
t11	<b>Commit</b>		190
t12	Unlock(balx)		190

### Locks and the uncommitted dependency problem

To prevent this problem occurring, T4 first requests a write lock on balx. It can then proceed to read the value of balx from the database, increment it by 100 and write the new value back to the database. When the rollback is executed, the updates of T4 are undone and the value of balx in the database is returned to its original value of 100. When T3 starts, it also requests a write lock on balx. However, because the data item balx is currently write locked by T4, the request is not immediately granted and T3 has to **wait** until the lock is released by T4. This only occurs once the rollback of T4 has been completed.

Time	T3	T4	balx
t1		<b>Begin transaction</b>	100
t2		Write_lock(balx)	100
t3		read(balx)	100
t4	<b>Begin transaction</b>	balx=balx+100	100
t5	Write_lock(balx)	write(balx)	200



t6	Wait	Rollback	100
t7	Wait	Unlock(balx)	100
t8	read(balx)		100
t9	balx=balx-10		100
t10	write(balx)		90
t11	Commit		90
t12	Unlock(balx)		90

### Locks and the inconsistent analysis problem

To prevent this problem occurring, T5 must precede its reads by write locks and T6 must precede its reads with read locks. Therefore, when T5 starts it requests a write lock on balx. Now, when T6 tries to read lock balx the request is not immediately granted and T6 has to wait until the lock is released, which is when T5 commits.

Time	T5	T6	balx	baly	balz	sum
t1		Begin transaction	100	50	25	
t2	Begin transaction	Sum=0	100	50	25	0
t3	write_lock(balx)		100	50	25	0
t4	read(balx)	read_lock(balx)	100	50	25	0
t5	balx=balx-10	Wait	100	50	25	0
t6	write(balx)	Wait	90	50	25	0
t7	write_lock(balz)	Wait	90	50	25	0
t8	read(balz)	Wait	90	50	25	0
t9	balz=balz+10	Wait	90	50	25	0
t10	write(balz)	Wait	90	50	35	0
t11	Commit	Wait	90	50	35	0
t12	Unlock(balx, balz)	Wait	90	50	25	0
t13		read(balx)	90	50	35	0
t14		Sum = Sum+balx	90	50	35	90

t15		read_lock(baly)	90	50	35	90
t16		read(baly)	90	50	35	90
t17		Sum = Sum+baly	90	50	25	140
t18		read_lock(balz)	90	50	35	140
t19		read(balz)	90	50	35	140
t20		Sum = Sum+balz	90	50	35	175
t21		<b>Commit</b>	90	50	35	175
t22		<b>Unlock(balx,baly,ba lz)</b>	90	50	35	175

### Deadlock

A situation when two or more transactions are each waiting for locks held by the other to be released. The following example shows such a situation. T10 and T11, are two transactions that are deadlocked because each is waiting for the other to release a lock on item it holds. At time t2, T10 requests and obtains a write lock on item balx, and at time t3 transaction T11 obtains a write lock on item baly. Then at time t6, T10 requests a write lock on item baly. Since T11 holds a lock on baly, transaction T10 waits. Meanwhile, at time t7, T11 requests a lock on item balx, which is held by transaction T10. Neither transaction can continue because each is waiting for a lock it cannot obtain until the other completes. Once a deadlock occurs, the application involved cannot resolve the problem. Instead, the DBMS has to recognize that deadlock exists and break the deadlock in some way.

Time	T10	T11
t1	Begin transaction	
t2	Write_lock(balx)	Begin transaction
t3	read(balx)	Write_lock(baly)
t4	balx=balx-10	read(baly)
t5	write(balx)	baly=baly+100
t6	Write_lock(baly)	write(baly)
t7	Wait	Write_lock(balx)
t8	Wait	Wait
t9	Wait	Wait
t10	.	Wait



t11	.	.
t12	.	.

Unfortunately, there is only one way to break a deadlock: abort one or more of the transactions. This involves undoing all the changes made by the transaction(s). In the above example, we may decide to abort transaction T11. Once this is complete, the locks held by transaction T11 are released and T10 is able to continue again. There are two general techniques for handling deadlock: **deadlock preventing**, here, the DBMS looks ahead to see if a transaction would cause a deadlock and never allows a deadlock to occur. The second method is, **deadlock detection and recovery**, the DBMS allows a deadlock to occur but recognizes occurrences of a deadlock and then breaks it. Since is easier to test for a deadlock and break it when it occurs than to prevent it, many systems use the detection and recovery method.

### Timestamping

Timestamp methods of concurrency control are quite different from locking methods. No locks are involved, and therefore there can be no deadlock. Locking methods generally involve transactions that make conflicting requests wait. With timestamp methods, there is no waiting; transactions involved in conflict are simply rolled back and restarted. A **timestamp** is a unique identifier created by the DBMS that indicates the relative starting time of a transaction. **Timestamping** is a concurrency control protocol in which the fundamental goal is to order transactions globally in such a way that older transactions, transactions with *smaller* timestamps, get priority in the event of conflict. With timestamping, if a transaction attempts to read or write a data item, then the read or write is only allowed to proceed if the *last update on that data item* was carried out by an older transaction; otherwise, the transaction requesting the read/write is restarted and given a new timestamp.

### Data Security

Security involves the protection of data against unauthorized disclosure, alteration, or destruction. Database security involves protecting a database against potential threats using both technical and administrative controls. Consequently, the scope of database security includes hardware, software and people.

Software security is enforced by the DBMS's security subsystem, which checks all access requests against **security rules** stored in the system catalog. The type of privilege that an authorized user may be given include:

- use of specified named databases;
- selection of retrieval of data;
- creation of tables and other objects;
- update of data (may be restricted to certain column(s));



- deletion of data (may be restricted to certain column(s));
- insertion of data (may be restricted to certain column(s));
- execution of specific procedures and utility programs;
- creation of databases;
- membership of a group or groups of users, and consequent inheritance of the group's privileges.

SQL provides two commands, for defining/removing from the system, GRANT and REVOKE.

### Granting privileges users

The GRANT statement is used to grant privileges on database objects to specific users. The format of the statement:

```
GRANT    privilege_list | ALL PRIVILEGE
ON       object_name
TO       user_list | PUBLIC
[WITH GRANT OPTION]
```

The *privilege\_list* consists of one or more of the following privileges separated by commas:

```
SELECT
DELETE
INSERT [(column_name,...)]
UPDATE [(column_name,...)]
REFERENCES [(column_name,...)]
USAGE
```

For convenience, the GRANT statement allows the keyword ALL PRIVILEGES to be used to grant all privileges to all user instead of having to specify the six privileges individually. It also provides the keyword PUBLIC to allow access to be granted to all present and future authorized users. The object\_name can be the name of a base table, view, domain, etc. The WITH GRANT OPTION keyword allows the user(s) in *user\_list* to pass the privileges they have been given for the named object on to other users.

### Examples

1. Give the user named *manager* full privileges to the table called *staff*.

```
GRANT ALL PRIVILEGES
ON staff
TO manager WITH GRANT OPTION
```



The user identified as manager can now be retrieve rows from the staff table, and also insert, update and delete data from this table. We also specified the keyword WITH GRANT OPTION, so that manager can pass these privileges on to other users he or she sees fit.

2. Give the user, admin, the privileges SELECT and UPDATE on the salary column of the staff table.

```
GRANT SELECT, UPDATE (salary)
```

```
ON staff
```

```
TO admin
```

3. Give the users, personnel and deputy the privilege SELECT on the staff table.

```
GRANT SELECT
```

```
ON staff
```

```
TO personnel, deputy
```

4. Give all users the privilege SELECT on the branch table.

```
GRANT SELECT
```

```
ON branch
```

```
TO PUBLIC
```

### Revoking privileges from users

The REVOKE statement is used to take away privileges that were granted with the GRANT statement. Its format looks like:

```
REVOKE    privilege_list | ALL PRIVILEGES
ON        object_name
FROM      user_list | PUBLIC [RESTRICT|CASCADES]
```

### Examples

1. Revoke the SELECT privilege from the branch table from all users

```
REVOKE SELECT
```

```
ON branch
```

```
FROM PUBLIC
```

2. Revoke all the privileges given to user deputy on the staff table.

```
REVOKE ALL PRIVILEGES
```

```
ON staff
```

```
FROM deputy
```

In some systems security rules are defined on views rather than base tables. The above two statements can be used for such systems as well. We need to create the view before any grant operation.

For example, to grant a user, called Abebe, SELECT privileges on the salary and name fields of the staff table. First, we need to create a view that holds the data and selected columns, as follows.

```
CREATE VIEW viewname AS  
SELECT salary, name FROM staff
```

And, then

```
GRANT SELECT  
ON viewname  
TO Abebe
```

### Data Encryption

Encryption is the encoding of the data by a special algorithm that makes the data unreadable by any program without the decryption key. If a database system holds particularly sensitive data, it may be deemed necessary to encode it as a precaution against possible external (i.e., external to the DBMS) threats or attempts to access it. Encryption also protects data transmitted over communication lines. Some DBMSs provide an encryption facility for these purposes. The DBMS can access the data (after decoding it), although there is a degradation in performance because of the time taken to decode it.

#### 3.1.16. Distributed Database and Client/Server Systems

##### Distributed Database Systems

A distributed database system consists of a collection of sites, connected together via some kind of communications network, in which

4. Each site is a database system in its own right, and
5. The sites have agreed to work together so that a user at any site can access data anywhere in the network exactly as if the data were all stored at the user's own site.

The distributed database system can thus be regarded as a kind of partnership among the individual local DBMSs at the individual local sites; a new software component at each site (logically an extension of the local DBMS) provides the necessary partnership functions, and it is the combination of this new component together with the existing DBMS that constitutes what is usually called the **Distributed Database Management System (DDBMS)**. It is common to assume that the component sites are physically dispersed (or geographically dispersed.) The **fundamental principle of a distributed database is, to the user, a distributed system should look exactly like a non-distributed system.** In order to have this effect, the following objectives must be met:



*Local autonomy*  
*No reliance on a central site*  
*Continuous operation*  
*Location independence*  
*Fragmentation independence*  
*Replication independence*  
*Distributed query processing*  
*Distributed transaction management*  
*Hardware independence*  
*Operating system independence*  
*Network independence*  
*DBMS independence*

Distributed database systems are being installed on *massively parallel* computer systems. Such computer systems typically consist of a large number of separate processors connected together by means of a high speed-bus; each processor has its own main memory and its own disk drives and runs its copy of the DBMS software, and the complete database is spread across the complete set of disk drives.

### **Client/Server Systems**

A client/server system can be regarded as a simple special case of distributed systems in general. More precisely, a client/server system is a distributed system in which

- a) Some sites are *client* sites (the clients may not be database systems) and others are *server* sites.
- b) All data resides on the server sites, and
- c) All applications execute at the client sites.

Client/server primarily refers to an architecture or logical division of responsibilities: the **client** is the application (also known as the *front-end*), and the **server** is the DBMS (also known as the *backend*). A client/server can be thought of as a distributed system in which all requests originate at one site all processing is performed at another (assuming for simplicity that there is just one client site and one server site).

The **server** is the DBMS itself. It supports all of the basic DBMS functions, i.e., data definition, data manipulation, data security and integrity, and so on.

The **clients** are the various applications that run on top of the DBMS (both user-written and built-in applications, i.e., application provided either by the vendor of the DBMS or by some “third-party” software vendor). As far as the server is concerned, of course, there is no difference between user-written and built-in applications.



The programming activity may require queries done on row (record) levels. The number of messages between client and server can be reduced if the system provides some kind of **stored procedure** mechanism. A stored procedure is basically a precompiled program that is *stored at the server site (and is known to the server)*. It is invoked from the client by a **remote procedure call (RPC)**. In particular, therefore, the performance penalty associated with record-level processing can be partly offset by creating a suitable stored procedure to do that processing directly at the server. Other advantages of stored procedures are:

- Can be used to conceal a variety of systems specific and/or database specific details from the user, thereby providing a greater degree of data independence than might otherwise be the case.
- One stored procedure can be shared by many clients.
- Optimization can be done at the time the stored procedure is created instead of run time.
- Stored procedures can provide better security.
- One problem is that there are currently no standards in this area, and different products provide very different facilities.

### **3.2. Advanced Database Systems**

#### **Module Objective**

On completion of the module successfully, students will be able to:

- Develop database based on object-oriented paradigm
- Compare a set of query processing strategy and select the optimal strategy.
- Describe the basics of transaction management and concurrency control
- Demonstrate database security
- Use different recovery methods when there is a database failure
- Design a distributed database system in homogenous and heterogeneous environments

#### **3.2.1. Concepts for Object-Oriented Databases**

##### **3.2.1. Introduction**

Data is essentially represented in a relational model. Every possible kind of data is reduced to a set of tuples or a set of rows other kinds of data DBMS requirement cannot be easily mapped in to relational model. Those kinds of data need to be represented using Object-oriented database

##### **What kind of data? Complex data objects**

Example of complex data objects



**a. Multimedia databases**

- Menu, Scroll bar, Drawing area, Flash animation

**b. CAD for electrical design**

- PCB, Resistor, Capacitor, Transistor, IC, Voltage source, Ground

**Characteristics of complex data objects**

- Not amenable (agreeable) to reduction to a tuple
- Represents abstractions to comprising not only structure, but also behavior
- Instances distinct from classes represented by current state

**Drawbacks of Relational DBMS:**

- Poor representation of “real world” entities.
- Semantic overloading
- Poor support for integrity and enterprise constraints.
- Homogenous data structure.
- Limited operations.
- Difficulty handling recursive queries
- Impedance mismatch.
- Other problems concerning: concurrency, schema access, navigational access, and so on

**3.2.2. Object Oriented Concepts**

*What is Object-Oriented Database? What objectives does an Object-Oriented database want to achieve?*

Simply put, an object-oriented database, or OODBMS (Object Oriented Database Management System), is a database that can store **objects**.

When queried, these databases return the objects in their entirety, which means a returned object's attributes and methods are as usable as they were before the object was ever stored in the database

The fundamentals of object databases with a specific focus on conceptual modeling of object database designs. The main concepts are EER (Enhanced Entity Relation), LINQ (Language Integrated Query, object databases, object-oriented databases, object-relational databases, UML (Unified Modeling Language).

**Object has three characteristics**

- State
- Behavior
- Identifier

## Example

### Bank account

- State: Id, name, balance
- Behavior: Deposit , withdraw
- Identifier Solomon's acct is similar to Helen's acct, but its states has different value

An **Object Database (ODB)** aims to make objects persistent, in addition to support the large number of other features of a database system. These expected database features include: the efficient management of persistent data; transactions, concurrency and recovery control; an ad hoc query language. An ODB has to provide these database features in the context of the complexities introduced by object-orientation. That could be challenging.

**What is an object?** It refers to an abstract concept that generally represents an entity of interest in the enterprise to be modeled by a database application. An object has to reflect a state and some behavior. The object's state shows the internal structure of the object and the internal structure are the properties of the object. We can view a student as an object. The **state** of the object has to contain descriptive information such as an identifier, a name, and an address.

The **behavior** of an object is the set of methods that are used to create, access, and manipulate the object. A student object, for example, may have methods to create the object, to modify the object state, and to delete the object. The object may also have methods to relate the object to other objects, such as enrolling a student in a course or assign a note to a student in a course. Objects having the same state and behavior are described by a **class**.

### Advantages of Object-oriented Database

- Designer can specify the structure of objects and their behavior (methods)
- Better interaction with object-oriented languages such as Java and C++
- Definition of complex and user-defined types
- Encapsulation of operations and user-defined methods



## Key Terms

Given this fundamental definition of an object, the following object-oriented concepts are typically associated with an ODB:

- class
- complex objects
- object identity
- object structure
- object constructor
- method and persistence
- encapsulation
- extensibility
- polymorphism
- class hierarchies and inheritance (multiple and selective inheritance)
- overriding, overloading and late binding

**Activity 1** – Definition terms now, we are going to present each of them below.

### **Class**

A class essentially defines the type of the object where each object is viewed as an instance of the class. For example, Person is a class and Jacqueline is an instance or an object of this class.

### **Complex Objects**

Complex objects or nested relations are objects which are designed by combining simple objects. These objects used to be created by constructors.

An airplane and a car are examples of complex objects because they can be viewed as a higher level object that is constructed from lower level objects, such as engine and body (for both), the airplane wings and tail. Each of these objects can be complex objects that are constructed from other simple or complex objects.

### **Object Identity**

An object identity is an internal identifier of object that could not be changed. An object identity (oid) remains constant during object lifetime. The oid is use by object database to uniquely identify the object whose attributes can change in contrast to relational database which does not allow the attributes changing for a tuple in the table. The identity of an object does not depend on properties



value, the *iod* are unique references and they are useful to construct complex objects [Dietrich and Urban, 2011].

### Object Structure

A class object contains information about the state described but attributes and the behavior allowed by methods. An attribute and a method that are for the instances of the class are called instance attribute and an instance method. **For example**, let *Student* be a class with following attributes: *stid*, *fname*, *lname*. Each student will have his/her *stid*, *fname*, *lname*. So, two different objects (students) of *Student* class will have different values of instance attributes.

An attribute and a method that are common to all instances of the class are called **instance attribute** and an instance method. They are called **class attributes** and class methods. For example, let *Student* be a class with following attributes: *stid*, *fname*, *lname*, *city*. The default value of *city* can be “**Bamako**” so that all students will get Bamako as *city*. So, two different objects (students) of *Student* class will have the same value of class attributes.

### Type Constructors

Also called type generator, a type constructor is particular kind of method in a class. It is called to create an object of this class and initialize its attributes. The constructor has the same name as the **class**. Unlike to ordinary methods, the **constructor** has not explicit return type because.

### Encapsulation of Operations

- Implementation of operations and object structure hidden.
- Binding properties with functions

Encapsulation consists to gather data and methods within a structure with interface in order to hide the object implementation. The goal is to prevent access to data by any means other than the allowed services defined through interface. Encapsulation therefore guarantees the integrity of the object data. So, the implementation of a class can change without affecting the interface that the class provides to the rest of the application. Encapsulation is useful to separate class specification level from class implementation level, this is very important as software engineering approach. User-defined types allow object database supporting the encapsulation [Dietrich and Urban, 2011].

### Extensibility

The encapsulation property in an ODB also makes them extensible. “Extensibility refers to the ability to extend an existing system without introducing changes to it” [Won, 1990]. It easily allows evolvement of systems, in particular which are large and complex.

Extensibility can be introduced through behavioral **extension** and **inheritance**.

When the extending and object behavior, we add new programs to augment its functionality and that should not degrade the older functionality. For example, a new program named *Borrow* may be



added to Item inherited by books, newspapers, CD/DVD. Through extensibility by reusing or inheritance, the behavior and the attributes defined for an Item object may be reused by the specialized objects. For instance, a new object named Book may be defined as a specialized Item object. The Book object inherits (reuses) the behavior (Borrow) and attributes (isbn, author-name) defined for the Item object. Also, we can either add new behavior/ attributes to Book object, or re-define existing behavior/attributes inherited.

### **Persistence**

Storing permanently the values of properties of an object is called object persistence. In other words, persistence refers to the ability for objects to remain after stopping the process that created them [Dogac et al., 1994]. The properties of persistent object are created and stored in main memory.

Object-oriented database systems are based on the concept of persistent objects. **Methods**

In object-oriented approach, the behavior of an object is described by a set of operations that are called methods. A method has a signature that describes the name of the method and the names and types of the method parameters. *A method is a particular implementation of a method signature.* They are defined in object class and allow to relate different others objects and manipulating them.

### **Activity 2 Some object-oriented terms**

#### **Type Hierarchies & Inheritance**

In object-oriented approach, there is the possible to define class hierarchies and to express the inheritance of state and behavior. **Inherited objects** are related by “Is-a” relationships. For example, Circle is a Shape and also Triangle is a Shape and we say Circle and Triangle classes inherited the class Shape. Another class called Equilateral Triangle is a Triangle. But we have two hierarchy levels between Shape and Equilateral Triangle.

The ODB have advantages to completely support class hierarchies and inheritance.

#### **Polymorphism**

Also called overloading, polymorphism refers to the capability of an object to behave differently to same message. The ODB support polymorphism. For example, there might be a method for area computing like Area() for all Shape objects. However, Circle and Triangle objects can adapt this method to their particular area computation. So, we will get Area() method for Circle object and Area() for Triangle. That is called polymorphism or overloading.

#### **Multiple Inheritances**

When an object inherits from only one class, we have simple inheritance. But, when an object inherits from two or more classes, it is multiple inheritances. For example, a Hybrid Car is a Gasoline Car and also Electric Car. So, there is double inheritance because Hybrid Car inherits Gasoline Car and Electric Car.



### **Late binding**

Polymorphism is used with late binding, which means that the translation of an operation name to its appropriate method implementation must be dynamically resolved, i.e. at run-time (Dietrich and Urban, 2011).

## **3.2.3. Query Processing and Optimization**

### **Introduction**

Database performance tuning often requires a deeper understanding of how queries are processed and optimized within the database management system. In this set of notes we provide a general overview of how rule based and cost-based query optimizers operate and then provide some specific examples of optimization in commercial DBMS.

**Query optimization** is a function of many relational database management systems. The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans. Generally, the query optimizer cannot be accessed directly by users: once queries are submitted to database server, and parsed by the parser, they are then passed to the query optimizer where optimization occurs.

### **Specific Teaching and Learning Objectives**

At end of the section, the learner should be aware of:

- Different steps of query processing
- Translating SQL queries into Relational Algebra
- Using Heuristics in Query Optimization

### **Activity 1 Definitions**

A **query** is a high-level specification of a set of objects of interest from the database. It is usually specified in a special language (Data Manipulation Language - DML) that describes what the result must contain.

**Query Optimization** aims to choose an efficient execution strategy for query execution.

**Query-processing** in object-oriented databases is almost the same as in relation database with only few changes because of semantic differences between relational and object-oriented queries. Moreover, all the techniques applicable to one are also to another. Indeed, without the class hierarchy, queries have similar structures in both databases.

Therefore, in order to simplify, we are going see queries processing without distinguishing between object databases and relational databases.

**Query** is processed in two phases: the *query-optimization phase* and the *query-processing phase*. In order to facilitate the understanding, we will add the query-compilation phase before the two previous



phases because queries are viewed by user as Data Manipulation Language (DML) scripts. So, the figure 3 presents the whole process.

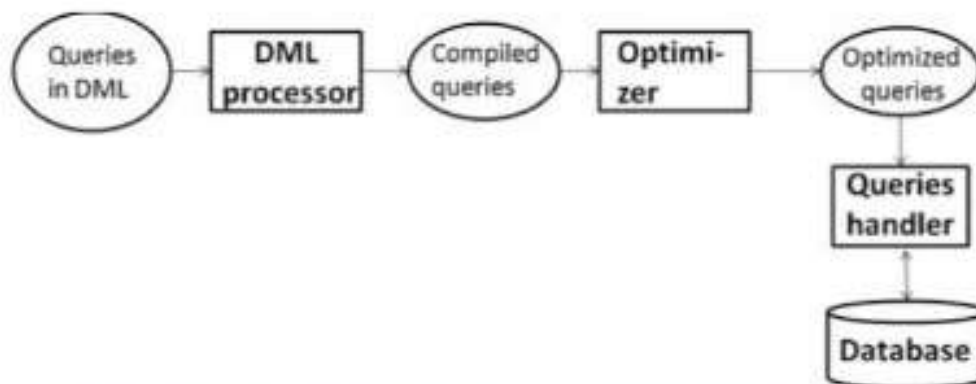


Figure3: Steps of query Processing

**Query-compilation:** DML processor translates DML statements into low-level instructions (compiled query) that the query optimizer can understand.

**Query-optimization:** the optimizer automatically generates a set of reasonable strategies for processing a given query, and selects one optimal on the basis of the expected cost of each of the strategies generated.

**Query-processing:** the system executes the query using the optimal strategy generated.

In query-optimization, a SQL query is first translated into an equivalent relational algebra expression using a query tree data structure before to be optimized. We will therefore set out below how to pass from a SQL query to an expression in Relational Algebra.

#### General Transformation Rules for Relational Algebra Operations

We will state some transformation rules that are useful in query optimization, without proving them:

1. **Cascade of  $\sigma$ .** A conjunctive selection condition can be broken up into a cascade (that is, a sequence) of individual  $\sigma$  operations:

$$\sigma_{c_1 \text{ AND } c_2 \text{ AND } \dots \text{ AND } c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\dots(\sigma_{c_n}(R))\dots))$$

2. **Commutativity of  $\sigma$ .** The  $\sigma$  operation is commutative:

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$

3. **Cascade of  $\pi$ .** In a cascade (sequence) of  $\pi$  operations, all but the last one can be ignored:

$$\pi_{\text{List}_1}(\pi_{\text{List}_2}(\dots(\pi_{\text{List}_n}(R))\dots)) \equiv \pi_{\text{List}_1}(R)$$

4. **Commuting  $\sigma$  with  $\pi$ .** If the selection condition  $c$  involves only those attributes  $A_1, \dots, A_n$  in the projection list, the two operations can be commuted:

$$\pi_{A_1, A_2, \dots, A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1, A_2, \dots, A_n}(R))$$

5. **Commutativity of  $\bowtie$  (and  $\times$ ).** The join operation is commutative, as is the  $\times$  operation:

$$R \bowtie_c S \equiv S \bowtie_c R$$

$$R \times S \equiv S \times R$$

Notice that although the order of attributes may not be the same in the relations resulting from the two joins (or two Cartesian products), the *meaning* is the same because the order of attributes is not important in the alternative definition of relation.

6. **Commuting  $\sigma$  with  $\bowtie$  (or  $\times$ ).** If all the attributes in the selection condition  $c$  involve only the attributes of one of the relations being joined—say,  $R$ —the two operations can be commuted as follows:

$$\sigma_c(R \bowtie S) \equiv (\sigma_c(R)) \bowtie S$$

Alternatively, if the selection condition  $c$  can be written as  $(c_1 \text{ AND } c_2)$ , where condition  $c_1$  involves only the attributes of  $R$  and condition  $c_2$  involves only the attributes of  $S$ , the operations commute as follows:

$$\sigma_c(R \bowtie S) \equiv (\sigma_{c_1}(R)) \bowtie (\sigma_{c_2}(S))$$

The same rules apply if the  $\bowtie$  is replaced by a  $\times$  operation.

7. **Commuting  $\pi$  with  $\bowtie$  (or  $\times$ ).** Suppose that the projection list is  $L = \{A_1, \dots, A_n, B_1, \dots, B_m\}$ , where  $A_1, \dots, A_n$  are attributes of  $R$  and  $B_1, \dots, B_m$  are attributes of  $S$ . If the join condition  $c$  involves only attributes in  $L$ , the two operations can be commuted as follows:

$$\pi_L(R \bowtie_c S) \equiv (\pi_{A_1, \dots, A_n}(R)) \bowtie_c (\pi_{B_1, \dots, B_m}(S))$$

If the join condition  $c$  contains additional attributes not in  $L$ , these must be added to the projection list, and a final  $\pi$  operation is needed. For example, if attributes

$A_{n+1}, \dots, A_{n+k}$  of  $R$  and  $B_{m+1}, \dots, B_{m+p}$  of  $S$  are involved in the join condition  $c$  but are not in the projection list  $L$ , the operations commute as follows:

$$\pi_L(R \bowtie_c S) \equiv \pi_L((\pi_{A_1, \dots, A_n, A_{n+1}, \dots, A_{n+k}}(R)) \bowtie_c (\pi_{B_1, \dots, B_m, B_{m+1}, \dots, B_{m+p}}(S)))$$

For  $\times$ , there is no condition  $c$ , so the first transformation rule always applies by replacing  $\bowtie$  with  $\times$ .

8. **Commutativity of set operations.** The set operations  $\cup$  and  $\cap$  are commutative, but  $-$  is not.

9. **Associativity of  $\bowtie$ ,  $\times$ ,  $\cup$ , and  $\cap$ .** These four operations are individually associative; that is, if both occurrences of  $\theta$  stand for the same operation that is



**14. Pushing  $\sigma$  to only one argument in  $\cap$ .**

If in the condition  $\sigma_c$  all attributes are from relation  $R$ , then:

$$\sigma_c (R \cap S) = \sigma_c (R) \cap S$$

**15. Some trivial transformations.**

If  $S$  is empty, then  $R \cup S = R$

If the condition  $c$  in  $\sigma_c$  is true for the entire  $R$ , then  $\sigma_c (R) = R$ .

There are other possible transformations. For example, a selection or join condition  $c$  can be converted into an equivalent condition by using the following standard rules from Boolean algebra

(De Morgan's laws):

$$\text{NOT } (c_1 \text{ AND } c_2) \equiv (\text{NOT } c_1) \text{ OR } (\text{NOT } c_2)$$

$$\text{NOT } (c_1 \text{ OR } c_2) \equiv (\text{NOT } c_1) \text{ AND } (\text{NOT } c_2)$$

**A. Heuristic Approach in query optimization**

Heuristic Approach will be implemented by using the above transformation rules in the following sequence or steps.

***Sequence for Applying Transformation Rules***

1. Use  
Rule-1  $\rightarrow$  Cascade SELECTION
2. Use  
Rule-2: Commutativity of SELECTION  
Rule-4: Commuting SELECTION with PROJECTION  
Rule-6: Commuting SELECTION with JOIN and CARTESIAN  
Rule-10: commuting SELECTION with SET OPERATIONS
3. Use  
Rule-9: Associativity of Binary Operations (JOIN, CARTESIAN, UNION and INTERSECTION). Rearrange nodes by making the most restrictive operations to be performed first ( moving it as far down the tree as possible)
4. Perform Cartesian Operations with the subsequent Selection Operation
5. Use  
Rule-3: Cascade of PROJECTION  
Rule-4: Commuting PROJECTION with SELECTION  
Rule-7: Commuting PROJECTION with JOIN and CARTESIAN  
Rule-11: commuting PROJECTION with UNION

## Query block:

The basic unit that can be translated into the algebraic operators and optimized. A query block contains a single SELECT-FROM-WHERE expression, as well as GROUP BY and HAVING clause if these are part of the block. Nested queries within a query are identified as separate query blocks.

Process for heuristics optimization

1. The parser of a high-level query generates an initial internal representation;
  2. Apply heuristics rules to optimize the internal representation.
  3. A query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query.
- The main heuristic is to apply first the operations that reduce the size of intermediate results.
    1. E.g. Apply SELECT and PROJECT operations before applying the JOIN or other binary operations.

## Query tree:

A tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as leaf nodes of the tree, and represents the **relational algebra operations** as internal nodes.

## Query graph:

A graph data structure that corresponds to a **relational calculus expression**. It does not indicate an order on which operations to perform first. There is only a single graph corresponding to each query.

Example:

For every project located in 'Stafford', retrieve the project number, the controlling department number and the department manager's last name, address and birthdate.

## Relation algebra:

PNUMBER, DNUM, LNAME, ADDRESS, BDATE (((PLOCATION='STAFFORD')) (PROJECT))  
DNUM=DNUMBER (DEPARTMENT)) MGRSSN=SSN (EMPLOYEE))

SQL query:

```
Q2: SELECT P.NUMBER,P.DNUM,E.LNAME,E.ADDRESS, E.BDATE
      FROM PROJECT AS P,DEPARTMENT AS D, EMPLOYEE AS E
      WHERE P.DNUM=D.DNUMBER AND
            D.MGRSSN=E.SSN AND
            P.PLOCATION='STAFFORD';
```

## B. Cost Estimation Approach to Query Optimization

The main idea is to minimize the cost of processing a query. The cost function is comprised of: *I/O cost* + *CPU processing cost* + *communication cost* + *Storage cost*. These components might have different weights in different processing environments. The DBMs will use information stored in the system



catalogue for the purpose of estimating cost. The main target of query optimization is to minimize the size of the intermediate relation. The size will have effect in the cost of:

- Disk Access
- Data Transportation
- Storage space in the Primary Memory
- Writing on Disk

### **1. Access Cost of Secondary Storage**

Data is going to be accessed from secondary storage, as a query will be needing some part of the data stored in the database. The disk access cost can again be analyzed in terms of:

- Searching
- Reading, and
- Writing, data blocks used to store some portion of a relation.

Remark: The disk access cost will vary depending on

The file organization used and the access method implemented for the file organization.

Whether the data is stored contiguously or in scattered manner, will affect the disk access cost

### **2. Storage Cost**

While processing a query, as any query would be composed of many database operations, there could be one or more intermediate results before reaching the final output. These intermediate results should be stored in primary memory for further processing. The bigger the intermediate relation, the larger the memory requirement, which will have impact on the limited available space.

### **3. Computation Cost**

Query is composed of many operations. The operations could be database operations like reading and writing to a disk, or mathematical and other operations like:

- Searching
- Sorting
- Merging
- Computation on field values

### **4. Communication Cost**

In most database systems the database resides in one station and various queries originate from different terminals. This will have impact on the performance of the system adding cost for query processing. Thus, the cost of transporting data between the database site and the terminal from where the query originate should be analyzed.

## **C. Query Execution Plans**

An execution plan for a relational algebra query consists of a combination of the relational algebra query tree and information about the access methods to be used for each relation as well as the methods to be used in computing the relational operators stored in the tree.

### 3.2.4. Transaction Processing Concepts

#### 3.2.1. What is transaction?

A Transaction is an atomic unit of database access which is either completely executed or not executed at all. All or nothing principle. **Transaction processing systems** are systems with large databases and hundreds of concurrent users executing database transactions.

Examples include: *ATM transactions, credit card approvals, flight reservations, hotel check-in, phone calls, supermarket scanning, academic registration and billing.*

#### 3.2.2. Single-User versus Multiuser Systems

One criterion for classifying a database system is according to the number of users who can use the system concurrently. A DBMS is *single-user* if at most one user at a time can use the system, and it is multiuser if many users can use the system and hence access the database concurrently. Single-user DBMSs are mostly restricted to personal computer systems; most other DBMSs are multiuser. For example, an *airline reservations* system is used by hundreds of users and travel agents concurrently.

Database systems used in *banks, insurance agencies, stock exchanges, supermarkets*, and many other applications are multiuser systems. In these systems, hundreds or thousands of users are typically operating on the database by submitting transactions concurrently to the system. **Multiple** users can access databases and use computer systems simultaneously because of the concept of multiprogramming, which allows the operating system of the computer to execute multiple programs or processes at the same time.

A single

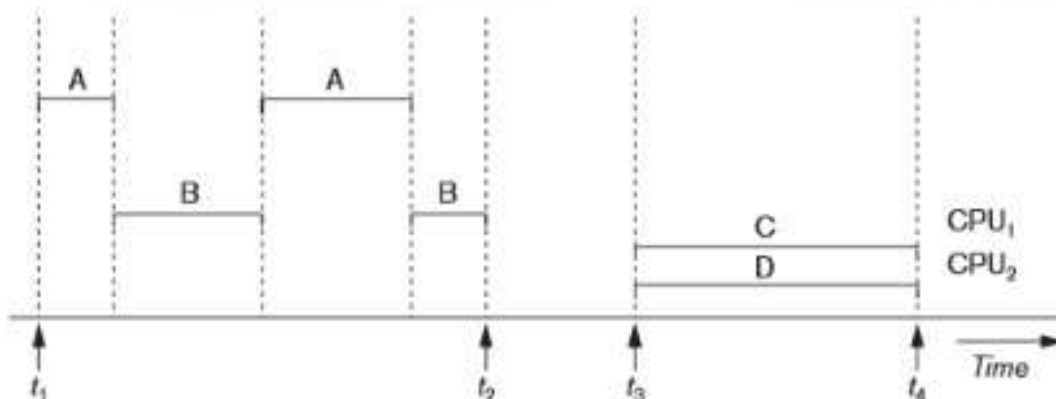


Figure: Interleaved processing versus parallel processing of concurrent transactions

Central processing unit (CPU) can only execute at most one process at a time. However, multiprogramming operating systems execute some commands from one process, then suspend that



process and execute some commands from the next process, and so on. A process is resumed at the point where it was suspended whenever it gets its turn to use the CPU again. Hence, concurrent execution of processes is actually interleaved as illustrated in Figure below, which shows two processes, **A** and **B**, executing concurrently in an interleaved fashion. Interleaving keeps the CPU busy when a process requires an input or output (I/O) operation, such as reading a block from disk. The CPU is switched to execute another process rather than remaining idle during I/O time. Interleaving also prevents a long process from delaying other processes. If the computer system has multiple hardware processors (CPUs), parallel processing of multiple processes is possible, as illustrated by processes **C** and **D** in Figure above. Most of the theory concerning concurrency control in databases is developed in terms of interleaved concurrency, so for the remainder of this chapter we assume this model. In a multiuser DBMS, the stored data items are the primary resources that may be accessed concurrently by interactive users or application programs, which are constantly retrieving information from and modifying the database.

#### **Transaction boundaries:**

Any single transaction in an application program is bounded with Begin and End statement

#### **3.2.3. Transactions States:**

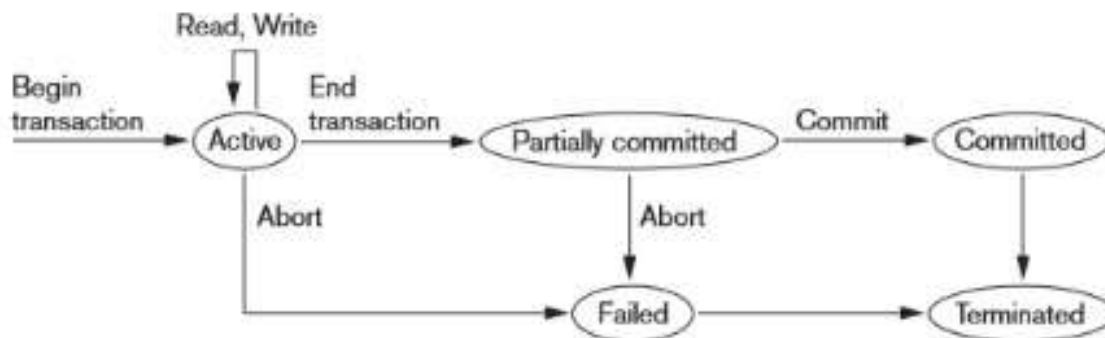
A transaction is an atomic unit of work that should either be completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when each transaction starts, terminates, and commits, or aborts therefore, the recovery manager of the DBMS needs to keep track of the following operations: **BEGIN\_TRANSACTION**. This marks the beginning of transaction execution. **READ** or **WRITE**. These specify read or write operations on the database items that are executed as part of a transaction. **END\_TRANSACTION**. This specifies that **READ** and **WRITE** transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates serializability or for some other reason.

**COMMIT\_TRANSACTION**. This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone. **ROLLBACK (or ABORT)**. This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone. Every transaction has the following states:-

- **Active state** -indicates the beginning of a transaction execution
- **Partially committed state** shows the end of read/write operation but this will not ensure permanent modification on the data base



- **Committed state** -ensures that all the changes done on a record by a transition were done persistently
- **Failed state** happens when a transaction is aborted during its active state or if one of the rechecking is fails
- **Terminated State** -corresponds to the transaction leaving the system



### 3.2.4. Desirable Properties of Transactions

Transactions should possess several properties, often called the ACID properties; they should be enforced by the concurrency control and recovery methods of the DBMS. The following are the ACID properties.

- **Atomicity:** A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.
- **Consistency:** A correct execution of the transaction must take the database from one consistent state to another.
- **Isolation:** A transaction should not make its updates visible to other transactions until it is committed; this property, when enforced strictly, solves the temporary update problem and makes cascading rollbacks of transactions unnecessary
- **Durability or permanency:** Once a transaction changes the database and the changes are committed, these changes must never be lost because of subsequent failure.

#### Example: Transaction

Suppose that  $T_i$  is a transaction that transfer 200 birr from account CA2090( which is 5,000 Birr) to SB2359(which is 3,500 birr) as follows

```

Read(CA2090)
CA2090= CA2090-200
Write(CA2090)
Read(SB2359)
SB2359= SB2359+200
Write(SB2359)
  
```



- **Atomicity**- either all or none of the above operation will be done – this is materialized by transaction management component of DBMS
- **Consistency**-the sum of CA2090 and SB2359 be unchanged by the execution of  $T_i$  i.e 8500- this is the responsibility of application programmer who codes the transaction
- **Isolation**- when several transactions are being processed concurrently on a data item they may create many inconsistent problems. So handling such case is the responsibility of Concurrency control component of the DBMS
- **Durability** - once  $T_i$  writes its update this will remain there when the database restarted from failure. This is the responsibility of recovery management components of the DBMS.

### Characterizing Schedules Based on Serializability

#### Transaction schedule or history:

When transactions are executing concurrently in an interleaved fashion, the order of execution of operations from the various transactions forms what is known as a transaction schedule (or history).

A **schedule**  $S$  of  $n$  transactions  $T_1, T_2, \dots, T_n$ :

It is an ordering of the operations of the transactions subject to the constraint that, for each transaction  $T_i$  that participates in  $S$ , the operations of  $T_i$  in  $S$  must appear in the same order in which they occur in  $T_i$ .

**Note**, however, that operations from other transactions  $T_j$  can be interleaved with the operations of  $T_i$  in  $S$

**Example:** Consider the following example:

<b>T1</b>	<b>T2</b>
	Read_item(A)
	A=A+100
	Write_item(A)
Read_item(A)	
A=A-10	
Write_item(A)	
	Abort

$S_a : r_2(X); w_2(X); r_1(X); w_1(X); a_2;$

Two operations in a schedule are said to be **conflict** if they satisfy all three of the following conditions:

- They belong to different transactions
- They access the same data item X
- At least one of the operations is a write\_item(X)

Eg. **Sa**: r1(X); r2(X); w1(X); r1(Y); w2(X); w1(Y);

- |  |   |                               |
|--|---|-------------------------------|
| <ul style="list-style-type: none"> <li>▪ r1(X) and w2(X)</li> <li>▪ r2(X) and w1(X);</li> <li>▪ w1(X) and w2(X)</li> </ul> | } | <b>Conflicting operations</b> |
| <ul style="list-style-type: none"> <li>▪ r1(X) and r2(X)</li> <li>▪ w2(X) and w1(Y)</li> <li>▪ r1(X) and w1(X)</li> </ul>  | } | <b>No Conflict, why?</b>      |

## Characterizing Schedules Based on Recoverability

It so happens that the transaction fails to execute until completion owing to software issues, hardware problems, or system crash. In such a situation, the failed transaction is rolled back by the rollback operation. However, another transaction may have utilized the value yielded by the failed transaction. Then that transaction will have to be rolled back as well to maintain consistency in the database.

### 1. Recoverable schedule:

- One where no committed transaction needs to be rolled back.
- A schedule S is recoverable if no transaction T in S commits until all transactions T' that have written an item that T reads have committed. Examples,
  - **Sc**: r1(X); w1(X); r2(X); r1(Y); w2(X); c2; a1; not recoverable
  - **Sd**: r1(X); w1(X); r2(X); r1(Y); w2(X); w1(Y); c1; c2; recoverable
  - **Se**: r1(X); w1(X); r2(X); r1(Y); w2(X); w1(Y); a1; a2; recoverable

### 2. Cascadeless schedule:

- One where every transaction reads only the items that were written by committed transactions. Eg.
  - **Sf**: r1(X); w1(X); r1(Y); c1; r2(X); w2(X); w1(Y); c2;

### 3. Strict Schedules:

- A schedule in which a transaction can neither read nor write an item X until the last transaction that wrote X has committed/aborted.
- Eg. **Sg**: w1(X,5); c1; w2(X,8);



## Characterizing Schedules based on Serializability

The concept of Serializable of schedule is used to identify which schedules are correct when concurrent transactions executions have interleaving of their operations in the schedule

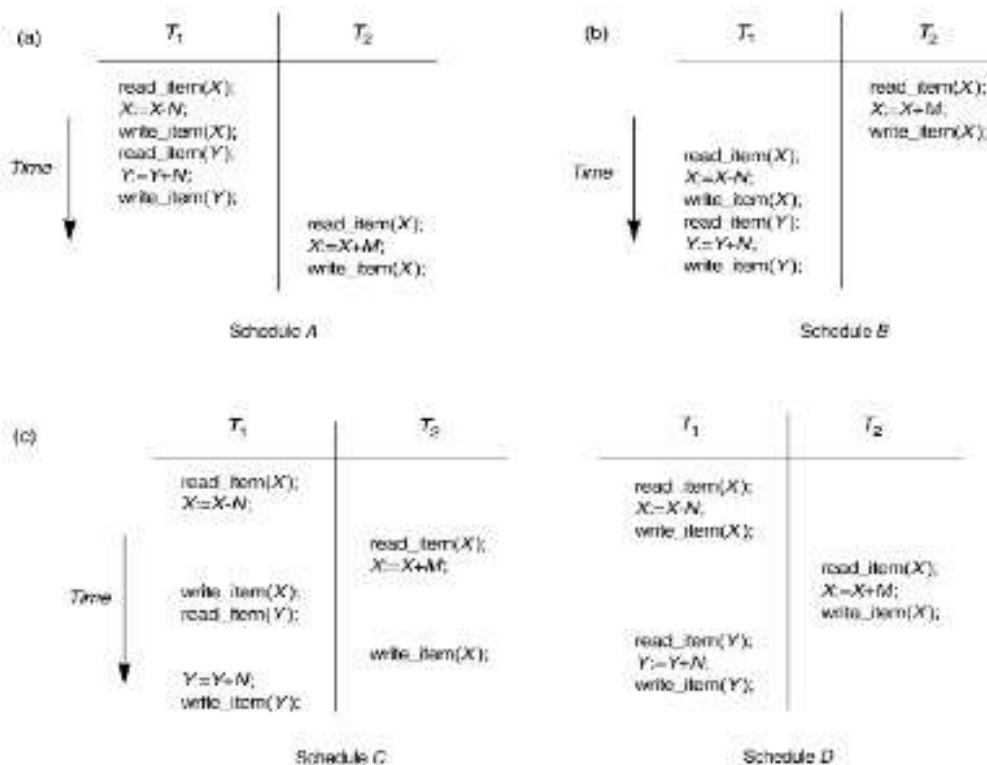
- **Serial schedule:**

- A schedule  $S$  is serial if, for every transaction  $T$  participating in the schedule, all the operations of  $T$  are executed consecutively in the schedule. Otherwise, the schedule is called nonserial schedule.
- For example, in the banking example suppose there are two transactions where one transaction calculate the interest on the account and another deposit some money into the account. Hence the order of execution is important for the final result.

- **Serializable schedule:**

- A schedule whose effect on any consistent database instance is identical to that of some complete serial schedule over the set of *committed* transactions in  $S$ .
- A nonserial schedule  $S$  is serializable, is equivalent to say that it is correct to the result of one of the serial schedule.

Example,



Examples of serial and nonserial schedules involving transactions  $T_1$  and  $T_2$ .

### Result equivalent:

- Two schedules are called result equivalent if they produce the same final state of the database
- Two types of equivalent schedule: Conflict and view

### i. Conflict equivalent:

- Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules.

Example

- S1: r1(x); w2(x) & S2: w2(x); r1(x) } Not conflict equivalent
- S1: w1(x); w2(x); & S2: w2(x); w1(x); }

- Conflict serializable:
- A schedule S is said to be *conflict serializable* if it is conflict equivalent to some serial schedule S'.
- Every conflict serializable schedule is serializable.

### ii. Two schedules are said to be view equivalent if the following three conditions hold:

1. The same set of transactions participates in S and S', and S and S' include the same operations of those transactions.
2. If T<sub>i</sub> reads a value A written by T<sub>j</sub> in S1, it must also read the value of A written by T<sub>j</sub> in S2
3. for each data object A, the transaction that perform the final write on A in S1 must also perform the final write on A in S2

S'			S	
T1: R(A)	W(A)		T1: R(A), W(A)	
T2: W(A)			T2: W(A)	
T3: W(A)			T3: W(A)	
		view		

### Testing for conflict serializable Algorithm

- Looks at only read\_Item (X) & write\_Item (X) operations
- Constructs a precedence graph (serialization graph) - a graph with directed edges
- An edge is created from T<sub>i</sub> to T<sub>j</sub> if one of the operations in T<sub>i</sub> appears before a conflicting operation in T<sub>j</sub>
- The schedule is serializable if and only if the precedence graph has no cycles.



### 3.2.5. Concurrency Control Techniques

Concurrency means executing multiple transactions at a time. It is the method of managing concurrent operations on the database without getting any obstruction with one another.

#### Advantages of concurrency

In general, concurrency means, that more than one transaction can work on a system.

The advantages of a **concurrent system** are:

- **Waiting Time:** It means if a process is in a ready state but still the process does not get the system to get execute is called waiting time. So, concurrency leads to *less waiting time*.
- **Response Time:** The time wasted in getting the response from the cpu for the first time, is called response time. So, concurrency leads to *less Response Time*.
- **Resource Utilization:** The amount of Resource utilization in a particular system is called Resource Utilization. Multiple transactions can run parallel in a system. So, concurrency leads to *more Resource Utilization*.
- **Efficiency:** The amount of output produced in comparison to given input is called efficiency. So, Concurrency leads to *more Efficiency*.

#### Problems of concurrency

The Simultaneous execution of transactions over a shared over a shared database can create several data integrity and consistency problems.

When **multiple transactions** execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems. These problems are commonly referred to as concurrency problems in a database environment. The five concurrency problems that can occur in the database are:

- Temporary Update Problem
- Incorrect Summary Problem
- Lost Update Problem
- Unrepeatable Read Problem
- Phantom Read Problem

#### 1. Temporary Update Problem:

Temporary update or dirty read problem occurs when one transaction updates an item and fails. But the updated item is used by another transaction before the item is changed or reverted back to its last value

#### 2. Incorrect Summary Problem:

Consider a situation, where one transaction is applying the aggregate function on some records while another transaction is updating these records. The aggregate function may calculate some values before the values have been updated and others after they are updated

#### 3. Lost Update Problem:



In the lost update problem, an update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction.

#### 4. Unrepeatable Read Problem:

The unrepeatable problem occurs when two or more read operations of the same transaction read different values of the same variable.

#### 5. Phantom Read Problem:

The phantom read problem occurs when a transaction reads a variable once but when it tries to read that same variable again, an error occurs saying that the variable does not exist.

### Conflicts in serializability of transactions

- Write-read conflict
- Read-write conflict
- Write-write conflict

**Serializability** means serial execution of transactions

Transaction Processor is divided into:

- 3.3. **concurrency-control manager**, or scheduler, responsible for assuring isolation of transactions
- 3.4. **A logging and recovery manager**, responsible for the durability of transactions.

The scheduler (*concurrency-control manager*) must assure that the individual actions of multiple transactions are executed in such an order that the net effect is the same as if the transactions had in fact executed one-at-a-time.

### Purpose of Concurrency Control

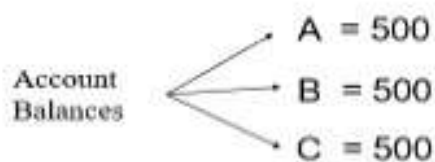
- To enforce Isolation (through mutual exclusion) among conflicting transactions.
- To preserve database consistency through consistency preserving execution of transactions.
- To resolve read-write and write-write conflicts.

A typical scheduler does its work by maintaining locks on certain pieces of the database. These locks prevent two transactions from accessing the same piece of data at the same time. Example: In concurrent execution environment if T1 conflicts with T2 over a data item A, then the existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or **waits**.



### Example:

Bank database: 3 Accounts



Property:  $A + B + C = 1500$

Money does not leave the system

### Example

Transaction T1: Transfer 100 from A to B

Read (A, t)

$t = t - 100$

Write (A, t)

Read (B, t)

$t = t + 100$

Write (B, t)

Transaction T1	Transaction T2	A	B	C
Read (A, t)		500	500	500
$t = t - 100$				
	Read (A, s)			
	$s = s - 100$			
	Write (A, s)	400	500	500
Write (A, t)		400	500	500
Read (B, t)				
$t = t + 100$				
Write (B, t)		400	600	500
	Read (C, s)			
	$s = s + 100$			
	Write (C, s)	400	600	600

### Schedule

$$400 + 600 + 600 = 1600$$

Transaction T1	Transaction T2	A	B	C
Read (A, t)		500	500	500
$t = t - 100$				
Write (A, t)		400	500	500
	Read (A, s)			
	$s = s - 100$			
	Write (A, s)	300	500	500
Read (B, t)				
$t = t + 100$				
Write (B, t)		300	600	500
	Read (C, s)			
	$s = s + 100$			
	Write (C, s)	300	600	600
		300 + 600 + 600 = 1500		

### Alternative Schedule

So What ?

## Concurrency control techniques

The concurrency control techniques are as follows –

### 1. Locking

Lock guarantees exclusive use of data items to a current transaction. It first accesses the data items by acquiring a lock, after completion of the transaction it releases the lock.

Types of Locks

The types of locks are as follows –

- Shared Lock [Transaction can read only the data item values]
- Exclusive Lock [Used for both read and write data item values]

### 2. Time Stamping

Time stamp is a unique identifier created by DBMS that indicates relative starting time of a transaction. Whatever transaction we are doing it stores the starting time of the transaction and denotes a specific time.

This can be generated using a system clock or logical counter. This can be started whenever a transaction is started. Here, the logical counter is incremented after a new timestamp has been assigned.

### 3. Optimistic

It is based on the assumption that conflict is rare and it is more efficient to allow transactions to proceed without imposing delays to ensure serializability.



#### 4. Multiversion Concurrency Control Techniques

This approach maintains a number of versions of a data item and allocates the right version to a read operation of a transaction. Thus, unlike other mechanisms a read operation in this mechanism is never rejected. This algorithm uses the concept of view serializability than conflict serializability

Side effect: Significantly more storage (RAM and disk) is required to maintain multiple versions. To check unlimited growth of versions, a garbage collection is run when some criteria is satisfied.

Two schemes: based on time stamped ordering & 2PL

#### 5. Validation (Optimistic) Concurrency Control Schemes

This technique allow transaction to proceed asynchronously and only at the time of commit, serializability is checked & transactions are aborted in case of non-serializable schedules. Good if there is little interference among transaction. It has three phases: **Read, Validation, and Write phase**

#### Deadlock

A **deadlock** is a condition wherein two or more tasks are waiting for each other in order to be finished but none of the task is willing to give up the resources that other task needs. In this situation no task ever gets finished and is in waiting state forever.

#### Coffman conditions

Coffman stated four conditions for a deadlock occurrence. A deadlock may occur if all the following conditions holds true.

- **Mutual exclusion condition:** There must be at least one resource that cannot be used by more than one process at a time.
- **Hold and wait condition:** A process that is holding a resource can request for additional resources that are being held by other processes in the system.
- **No preemption condition:** A resource cannot be forcibly taken from a process. Only the process can release a resource that is being held by it.
- **Circular wait condition:** A condition where one process is waiting for a resource that is being held by second process and second process is waiting for third process .....so on and the last process is waiting for the first process. Thus making a circular chain of waiting.

#### Deadlock Handling

##### Ignore the deadlock (Ostrich algorithm)

Did that made you laugh? You may be wondering how ignoring a deadlock can come under deadlock handling. But to let you know that the windows you are using on your PC, uses this approach of deadlock handling and that is reason sometimes it hangs up and you have to reboot it to get it working. Not only Windows but UNIX also uses this approach.



**The question is why? Why instead of dealing with a deadlock they ignore it and why this is being called as Ostrich algorithm?**

Well! Let me answer the second question first, This is known as Ostrich algorithm because in this approach we ignore the deadlock and pretends that it would never occur, just like Ostrich behavior “to stick one’s head in the sand and pretend there is no problem.”

**Let’s discuss why we ignore it:** When it is believed that deadlocks are very rare and cost of deadlock handling is higher, in that case ignoring is better solution than handling it. For example: Let’s take the operating system example – If the time requires handling the deadlock is higher than the time requires rebooting the windows then rebooting would be a preferred choice considering that deadlocks are very rare in windows.

### **Deadlock detection**

Resource scheduler is one that keeps the track of resources allocated to and requested by processes. Thus, if there is a deadlock it is known to the resource scheduler. This is how a deadlock is detected.

Once a deadlock is detected it is being corrected by following methods:

- **Terminating processes involved in deadlock:** Terminating all the processes involved in deadlock or terminating process one by one until deadlock is resolved can be the solutions but both of these approaches are not good. Terminating all processes cost high and partial work done by processes gets lost. Terminating one by one takes lot of time because each time a process is terminated, it needs to check whether the deadlock is resolved or not. Thus, the best approach is considering process age and priority while terminating them during a deadlock condition.
- **Resource Preemption:** Another approach can be the preemption of resources and allocation of them to the other processes until the deadlock is resolved.

### **Deadlock prevention**

We have learnt that if all the four Coffman conditions hold true then a deadlock occurs so preventing one or more of them could prevent the deadlock.

- **Removing mutual exclusion:** All resources must be sharable that means at a time more than one processes can get a hold of the resources. That approach is practically impossible.
- **Removing hold and wait condition:** This can be removed if the process acquires all the resources that are needed before starting out. Another way to remove this to enforce a rule of requesting resource when there are none in held by the process.
- **Preemption of resources:** Preemption of resources from a process can result in rollback and thus this needs to be avoided in order to maintain the consistency and stability of the system.
- **Avoid circular wait condition:** This can be avoided if the resources are maintained in a hierarchy and process can hold the resources in increasing order of precedence. This avoid circular wait.



Another way of doing this to force one resource per process rule – A process can request for a resource once it releases the resource currently being held by it. This avoids the circular wait.

### Deadlock Avoidance

Deadlock can be avoided if resources are allocated in such a way that it avoids the deadlock occurrence. There are two algorithms for deadlock avoidance.

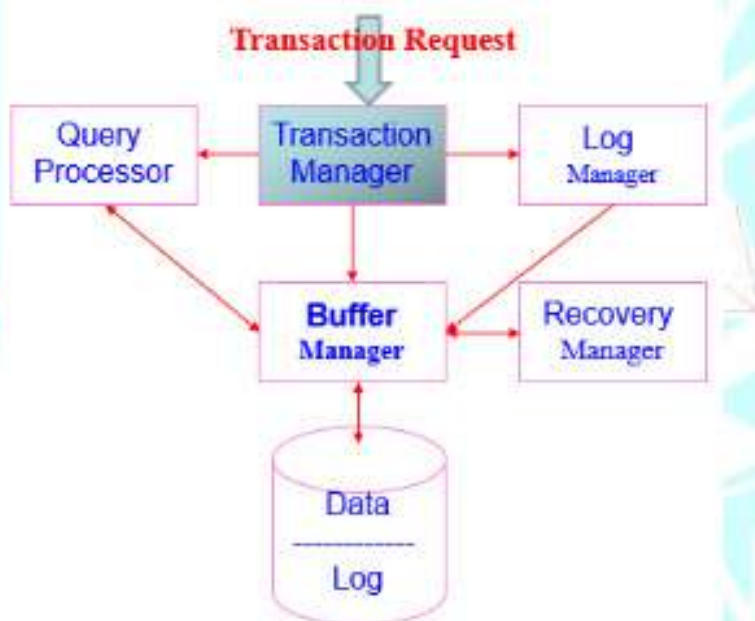
- Wait/Die
- Wound/Wait

Here is the table representation of resource allocation for each algorithm. Both of these algorithms take process age into consideration while determining the best possible way of resource allocation for deadlock avoidance.

### Database Recovery Techniques

#### 1. Recovery Concept

Transaction Manager: Accepts transaction commands from an application, which tell the transaction manager when transactions begin and end, as well as information about the expectations of the application. The transaction processor performs the following tasks:



- **Logging:** In order to assure *durability*, every change in the database is logged separately on disk.
- **Log manager** initially writes the log in buffers and negotiates with the buffer manager to make sure that buffers are written to disk at appropriate times.

- **Recovery Manager:** will be able to examine the log of changes and restore the database to some consistent state.

### Purpose of Database Recovery

- To bring the database into the last consistent state, which existed prior to the failure.
- To preserve transaction properties (Atomicity & Durability).

The recovery manager of DBMS is responsible to ensure atomicity by undoing the action of transaction that do not commit. **Durability** by making sure that all actions of committed transaction survive system crash.

### Example

If the system crashes before a fund transfer transaction completes its execution, then either one or both accounts may have incorrect value. Thus, the database must be restored to the state before the transaction modified any of the accounts.

### Types of Failure

The database may become unavailable for use due to:

- **Transaction failure:** Transactions may fail because of incorrect input, deadlock.
- **System failure:** System may fail because of addressing error, application error, operating system fault, RAM failure, etc.
- **Media failure:** Disk head crash, power disruption, etc.

To recover from system failure, the system keeps information about the change in the system log. Strategy for recovery may be summarized as:

### Recovery from catastrophic

- If there is extensive damage to the wide portion of the database
- This method restore a past copy of the database from the backup storage and reconstructs operation of a committed transaction from the back up log up to the time of failure

### Recovery from non-catastrophic failure

- When the database is not physically damaged but has become inconsistent
- The strategy uses *undoing and redoing some operations* in order to restore to a consistent state.

### Types of Data Update

#### Immediate Update:

- As soon as a data item is modified in cache, the disk copy is updated.

#### Deferred Update:

- All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.

#### Shadow update:



- The modified version of a data item does not overwrite its disk copy but is written at a separate disk location.

#### **In-place update:**

- The disk version of the data item is overwritten by the cache version.

#### **Data Caching**

Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk. When DBMS request for read /write operation on some item It check the requested data item is in the cache or not If it is not, the appropriate disk block are copied to the cache If the cache is already full, some buffer replacement policy can be used. Like Least recent used (LRU)

#### **FIFO**

While replacing buffers, first of all the updated value on that buffer should be saved on the appropriate block in the data base.

#### **Write-Ahead Logging**

- When *in-place* update (immediate or deferred) is used then log is necessary for recovery
- This log must be available to recovery manager
- This is achieved by *Write-Ahead Logging (WAL)* protocol. WAL states that
- **For Undo:** Before a data item's AFIM is flushed to the database disk (overwriting the BFIM) its BFIM must be written to the log and the log must be saved on a stable store (log disk).
- **For Redo:** Before a transaction executes its commit operation, all its AFIMs must be written to the log and the log must be saved on a stable store.

## Review Questions

1. In object-oriented database management system, all external entities should interact with the object through \_\_\_\_\_
  - A. Encapsulation
  - B. Abstraction
  - C. Signature
  - D. Interface
  - E. C and D
2. A query typically has many possible execution strategies, and the process of choosing a suitable one for query processing is
  - A. Query optimization
  - B. Query tree
  - C. Transaction management
  - D. Serializability
  - E. None
3. Which of the following is NOT a disadvantage of distributed database management system?
  - A. Complexity
  - B. Data replication
  - C. Data integrity and security problem
  - D. Problem of connecting dissimilar machines
  - E. All
4. What does ACID stand for in the context of DBMS transactions?
  - A. Atomicity, Consistency, Isolation, and Durability
  - B. Analysis Console for Intrusion Databases
  - C. Atomicity, Consistency, Isolation, and Data
  - D. Automatic Classification and Interpretation of Data
  - E. Advanced Computing Information Database
5. Consider a super market owner wishes to know in advance customer's preference on items bought based on their income level, gender, age group or any other metrics. Which database technology best solve this issue?
  - A. OO DBMS
  - B. Distributed database
  - C. Data mining
  - D. Data warehouse
  - E. None



6. Degree to which system user may remain unaware of the details of how and where the data items are stored in a distributed system
- A. Fragmentation
  - B. Replication
  - C. Data
  - D. Data transparency
  - E. None
7. Enforcing serializability in concurrent schedules ensures which two of the four desired properties for transactions?
- A. Atomicity and consistency
  - B. Atomicity and isolation
  - C. Atomicity and durability
  - D. Consistency and isolation
  - E. Consistency and durability
8. One of the following is not an application of data warehouse usage
- A. Information processing
  - B. Analytical processing
  - C. Data mining
  - D. A and B
  - E. None
9. Which of the following is true of a distributed DBMS?
- A. There is always one central server that distributes processing to other servers.
  - B. Table columns may be stored on servers in different physical locations.
  - C. Data can always be redistributed for optimal query processing.
  - D. All of the above.
  - E. None of the above
10. Which property is guaranteed by the two-phase locking protocol?
- A. serial schedules
  - B. serializable schedules
  - C. recoverable schedules
  - D. avoiding cascading rollback
11. After translating a query into a relational expression, the expression can be optimized by
- A. using equivalence rules to find alternative expressions that might cost less
  - B. using equivalence rules to make the expression tree more balanced

- C. using equivalence rules to do the cheaper operations first when possible
- D. A and C
- E. none of the above

12. Consider the SQL query `SELECT * FROM R1 WHERE C1=4 AND C2=10 AND C3=11`.

Suppose R1 has one million records stored in 100 disk pages, C1 has 10,000 unique values, C2 has 100,000 unique values, and C3 has 1,000 unique values, and values are distributed uniformly. Which relational algebra expression leads to the most efficient query execution plan?

- A.  $\sigma_{C1=4}(\sigma_{C2=10}(\sigma_{C3=11}(R1)))$
- B.  $\sigma_{C3=11}(\sigma_{C1=4}(\sigma_{C2=10}(R1)))$
- C.  $\sigma_{C2=10}(\sigma_{C1=4}(\sigma_{C3=11}(R1)))$
- D.  $\sigma_{C3=11}(\sigma_{C2=10}(\sigma_{C1=4}(R1)))$
- E. All four are equally efficient

13. Which of the following is not true about granted privileges in SQL?

- A. If user A grants a privilege to user B and user A subsequently loses that privilege, user B may still have that privilege.
- B. Privileges can be granted to users before the users are known to the DBMS.
- C. A user can be granted a privilege without giving that user authorization to grant that privilege to others.
- D. If privileges are revoked from one user, similar privileges may be automatically revoked from other users as well.
- E. All

14. Which type of lock still allows other transactions to have read-only access to the locked resource?

- A. shared lock
- B. two-phased lock
- C. exclusive lock
- D. explicit lock
- E. implicit lock

15. The recovery manager ensures which two of the four desired properties for transactions?

- A. Atomicity and consistency
- B. Atomicity and isolation
- C. Atomicity and durability
- D. Consistency and isolation
- E. Consistency and durability

16. What kind of conflict (if any) is present in the transaction schedule below?

`r1(A);w1(A); r2(A);w2(A); r2(B);w2(B); commit2; r1(B);w1(B); commit1`



- A. read-write conflict
  - B. write-read conflict
  - C. write-write conflict
  - D. no conflict
  - E. All of the above
17. Which of the following is NOT characteristics of homogeneous distributed database management system?
- A. All sites of the database system have identical setup
  - B. At least one of the database must be from different vendor
  - C. The system may have little or no local autonomy
  - D. The underlying operating systems can be a mixture of Linux, Window, Unix
  - E. None
18. In object-oriented database management system, all external entities should interact with the object through \_\_\_\_\_
- A. Encapsulation
  - B. Abstraction
  - C. Signature
  - D. Interface
  - E. C and D
19. Which of the following is NOT true about locks?
- A. Locks may have a table-level granularity
  - B. Locks with large granularity are easier for the DBMS to administer.
  - C. Locks with small granularity cause more conflicts.
  - D. Locks with large granularity produce fewer details for the DBMS to track.
  - E. Locks may have a database-level granularity
20. An algorithm used mining sequence data is
- A. Decision-tree classifiers
  - B. Apriori technique
  - C. Bayesian classifiers
  - D. Vector Machine
  - E. None
21. Which of the following is NOT a drawback of relational DBMS?
- A. Poor representation of "real world" entities.
  - B. Semantic overloading

- C. Homogenous data structure
  - D. Persistent object handling
  - E. Poor support for integrity and enterprise constraints
22. When attackers insert malicious code into the database program to exploit the vulnerabilities in the application, the situation is termed as \_\_\_\_\_
- A. Denial of service
  - B. SQL injection
  - C. Buffer overflow
  - D. Strong authentication
  - E. None
23. One of the following cannot be resolved by concurrency control techniques
- A. Read-Write conflict
  - B. Read-Read conflict
  - C. Write-Write conflict
  - D. Write-Read conflict
  - E. None
24. Set of variables that defines the state of a class in object-oriented database:
- A. Object state
  - B. Attribute
  - C. Method
  - D. Interface
  - E. Class
25. A data structure that corresponds query optimization to a relational calculus expression is
- A. Query graph
  - B. Query block
  - C. Query tree
  - D. nested queries
  - E. All of the above

## Part IV: SOFTWARE ENGINEERING

### 4.1. INTRODUCTION TO SOFTWARE ENGINEERING

The term *software engineering* is composed of two words, software and engineering.



**Software** is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be a collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product**.

**Engineering** on the other hand, is all about developing products, using well-defined, scientific principles and methods.

So, we can define **software engineering** as an engineering branch associated with the development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

IEEE defines software engineering as:

*The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software.*

Without using software engineering principles, it would be difficult to develop large programs. In industry it is usually needed to develop large programs to accommodate multiple functions. A problem with developing such large commercial programs is that the complexity and difficulty levels of the programs increase exponentially with their sizes. Software engineering helps to reduce this programming complexity. Software engineering principles use two important techniques to reduce problem complexity: **abstraction** and **decomposition**. The principle of abstraction implies that a problem can be simplified by omitting irrelevant details. Once the simpler problem is solved, then the omitted details can be taken into consideration to solve the next lower level abstraction, and so on. Abstraction is a powerful way of reducing the complexity of the problem. The other approach to tackle problem complexity is decomposition. In this technique, a complex problem is divided into several smaller problems and then the smaller problems are solved one by one. However, in this technique any random decomposition of a problem into smaller parts will not help. The problem has to be decomposed such that each component of the decomposed problem can be solved independently and then the solution of the different components can be combined to get the full solution.

## NEED OF SOFTWARE ENGINEERING

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

- **Large software** - It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
- **Scalability**- If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
- **Cost**- As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
- **Dynamic Nature**- The always growing and adapting nature of software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.



- **Quality Management-** Better process of software development provides better and quality software product.

## CHARACTERISTICS OF GOOD SOFTWARE

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

- Operational
- Transitional
- Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

### Operational

This tells us how well software works in operations. It can be measured on:

- Budget
- Usability
- Efficiency
- Correctness
- Functionality
- Dependability
- Security
- Safety

### Transitional

This aspect is important when the software is moved from one platform to another:

- Portability
- Interoperability
- Reusability
- Adaptability

### Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

- Modularity
- Maintainability
- Flexibility
- Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

## 4.2. SOFTWARE DEVELOPMENT LIFE CYCLE

### LIFE CYCLE MODEL

A software life cycle model (also called process model) is a descriptive and diagrammatic representation of the software life cycle. A life cycle model represents all the activities required to make a software product transit through its life cycle phases.



When a software product is being developed by a team there must be a clear understanding among team members about when and what to do. Otherwise it would lead to chaos and project failure. This problem can be illustrated by using an example. Suppose a software development problem is divided into several parts and the parts are assigned to the team members. From then on, suppose the team members are allowed the freedom to develop the parts assigned to them in whatever way they like. It is possible that one member might start writing the code for his part, another might decide to prepare the test documents first, and some other engineer might begin with the design phase of the parts assigned to him. This would be one of the perfect recipes for project failure. A software life cycle model defines entry and exit criteria for every phase. A phase can start only if its phase-entry criteria have been satisfied. So without software life cycle model the entry and exit criteria for a phase cannot be recognized. Without software life cycle models, it becomes difficult for software project managers to monitor the progress of the project.

### Different software life cycle models

Many life cycle models have been proposed so far. Each of them has some advantages as well as some disadvantages. A few important and commonly used life cycle models are as follows:

- Classical Waterfall Model
- Iterative Waterfall Model
- Prototyping Model
- Evolutionary Model
- Spiral Model

### CLASSICAL WATERFALL MODEL

The classical waterfall model is intuitively the most obvious way to develop software. Though the classical waterfall model is elegant and intuitively obvious, it is not a practical model in the sense that it cannot be used in actual software development projects. Thus, this model can be considered to be a *theoretical way of developing software*. But all other life cycle models are essentially derived from the classical waterfall model. So, in order to be able to appreciate other life cycle models it is necessary to learn the classical waterfall model. Classical waterfall model divides the life cycle into the following phases as shown in fig.2.1:



Fig 2.1: Classical Waterfall Model



**Feasibility study** - The main aim of feasibility study is to determine whether it would be financially and technically feasible to develop the product.

- At first project managers or team leaders try to have a rough understanding of what is required to be done by visiting the client side. They study different input data to the system and output data to be produced by the system. They study what kind of processing is needed to be done on these data and they look at the various constraints on the behavior of the system.
- After they have an overall understanding of the problem they investigate the different solutions that are possible. Then they examine each of the solutions in terms of what kind of resources required, what would be the cost of development and what would be the development time for each solution.
- Based on this analysis they pick the best solution and determine whether the solution is feasible financially and technically. They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development.

**Requirements analysis and specification:** - The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely

- Requirements gathering and analysis
- Requirements specification

The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed. This is done to clearly understand the customer requirements so that incompleteness and inconsistencies are removed.

The requirements analysis activity is begun by collecting all relevant data regarding the product to be developed from the users of the product and from the customer through interviews and discussions. For example, to perform the requirements analysis of a business accounting software required by an organization, the analyst might interview all the accountants of the organization to ascertain their requirements. The data collected from such a group of users usually contain several contradictions and ambiguities, since each user typically has only a partial and incomplete view of the system. Therefore, it is necessary to identify all ambiguities and contradictions in the requirements and resolve them through further discussions with the customer. After all ambiguities, inconsistencies, and incompleteness have been resolved and all the requirements properly understood, the requirements specification activity can start. During this activity, the user requirements are systematically organized into a Software Requirements Specification (SRS) document. The customer requirements identified during the requirements gathering and analysis activity are organized into a SRS document. The important components of this document are functional requirements, the nonfunctional requirements, and the goals of implementation.



The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the software architecture is derived from the SRS document. Two distinctly different approaches are available: the traditional design approach and the object-oriented design approach.

- **Traditional design approach** -Traditional design consists of two different activities; first a structured analysis of the requirements specification is carried out where the detailed structure of the problem is examined. This is followed by a structured design activity. During structured design, the results of structured analysis are transformed into the software design.
- **Object-oriented design approach** -In this technique, various objects that occur in the problem domain and the solution domain are first identified, and the different relationships that exist among these objects are identified. The object structure is further refined to obtain the detailed design.

**Coding and unit testing:** -The purpose of the coding phase (sometimes called the implementation phase) of software development is to translate the software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested. During this phase, each module is unit tested to determine the correct working of all the individual modules. It involves testing each module in isolation as this is the most efficient way to debug the errors identified at this stage.

**Integration and system testing:** -Integration of different modules is undertaken once they have been coded and unit tested. During the integration and system testing phase, the modules are integrated in a planned manner. The different modules making up a software product are almost never integrated in one shot. Integration is normally carried out incrementally over a number of steps. During each integration step, the partially integrated system is tested and a set of previously planned modules are added to it. Finally, when all the modules have been successfully integrated and tested, system testing is carried out. The goal of system testing is to ensure that the developed system conforms to its requirements laid out in the SRS document. System testing usually consists of three different kinds of testing activities:

- $\alpha$  – testing: It is the system testing performed by the development team.
- $\beta$  –testing: It is the system testing performed by a friendly set of customers.
- Acceptance testing: It is the system testing performed by the customer himself after the product delivery to determine whether to accept or reject the delivered product.

System testing is normally carried out in a planned manner according to the system test plan document. The system test plan identifies all testing-related activities that must be performed, specifies the schedule of testing, and allocates resources. It also lists all the test cases and the expected outputs for each test case.



**Maintenance:** -Maintenance of a typical software product requires much more than the effort necessary to develop the product itself. Many studies carried out in the past confirm this and indicate that the relative effort of development of a typical software product to its maintenance effort is roughly in the 40:60 ratios. Maintenance involves performing any one or more of the following three kinds of activities:

- Correcting errors that were not discovered during the product development phase. This is called corrective maintenance.
- Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called perfective maintenance.
- Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called adaptive maintenance.

#### Shortcomings of the classical waterfall model

The classical waterfall model is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases. However, in practical development environments, the engineers do commit a large number of errors in almost every phase of the life cycle. The source of the defects can be many: oversight, wrong assumptions, use of inappropriate technology, communication gap among the project engineers, etc. These defects usually get detected much later in the life cycle. For example, a design defect might go unnoticed till we reach the coding or testing phase. Once a defect is detected, the engineers need to go back to the phase where the defect had occurred and redo some of the work done during that phase and the subsequent phases to correct the defect and its effect on the later phases. Therefore, in any practical software development work, it is not possible to strictly follow the classical waterfall model.

#### ITERATIVE WATERFALL MODEL

To overcome the major shortcomings of the classical waterfall model, we come up with the iterative waterfall model.



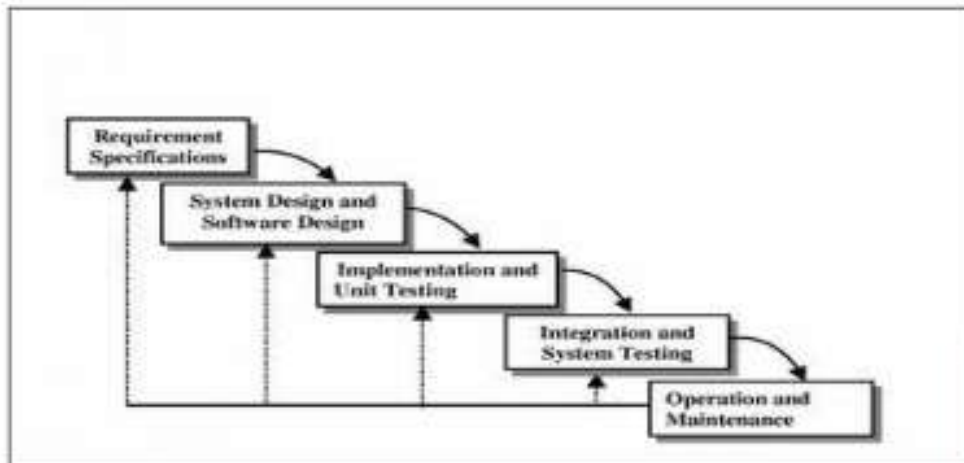


Fig 3.1 : Iterative Waterfall Model

Here, we provide feedback paths for error correction as & when detected later in a phase. Though errors are inevitable, but it is desirable to detect them in the same phase in which they occur. If so, this can reduce the effort to correct the bug.

The advantage of this model is that there is a working model of the system at a very early stage of development which makes it easier to find functional or design flaws. Finding issues at an early stage of development enables to take corrective measures in a limited budget.

The disadvantage with this SDLC model is that it is applicable only to large and bulky software development projects. This is because it is hard to break a small software system into further small serviceable increments/modules.

## PROTOTYPING

### MODEL PROTOTYPE

A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations. A prototype usually turns out to be a very crude version of the actual system.

### Need for a prototype in software development

There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs:

- a. how the screens might look like
- b. how the user interface would behave
- c. how the system would produce outputs

Another reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product.

A prototyping model can be used when technical solutions are unclear to the development team. A developed prototype can help engineers to critically examine the technical issues associated with the product development. Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

A prototype of the actual product is preferred in situations such as:

- User requirements are not complete
- Technical issues are not clear

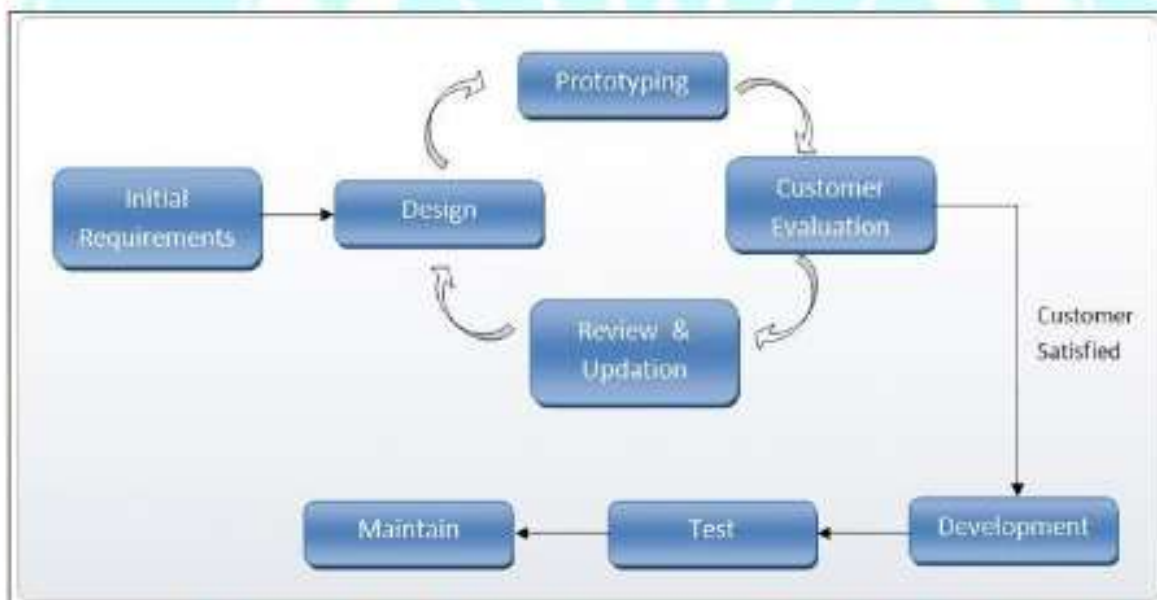


Fig 3.2: Prototype Model



## EVOLUTIONARY MODEL

It is also called *successive versions model* or *incremental model*. At first, a simple working model is built. Subsequently it undergoes functional improvements & we keep on adding new functions till the desired system is built.

Applications:

- A. Large projects where you can easily find modules for incremental implementation. Often used when the customer wants to start using the core features rather than waiting for the full software.
- B. Also used in object oriented software development because the system can be easily portioned into units in terms of objects.

Advantages:

- User gets a chance to experiment partially developed system
- Reduce the error because the core modules get tested thoroughly.

Disadvantages:

- It is difficult to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented & delivered.

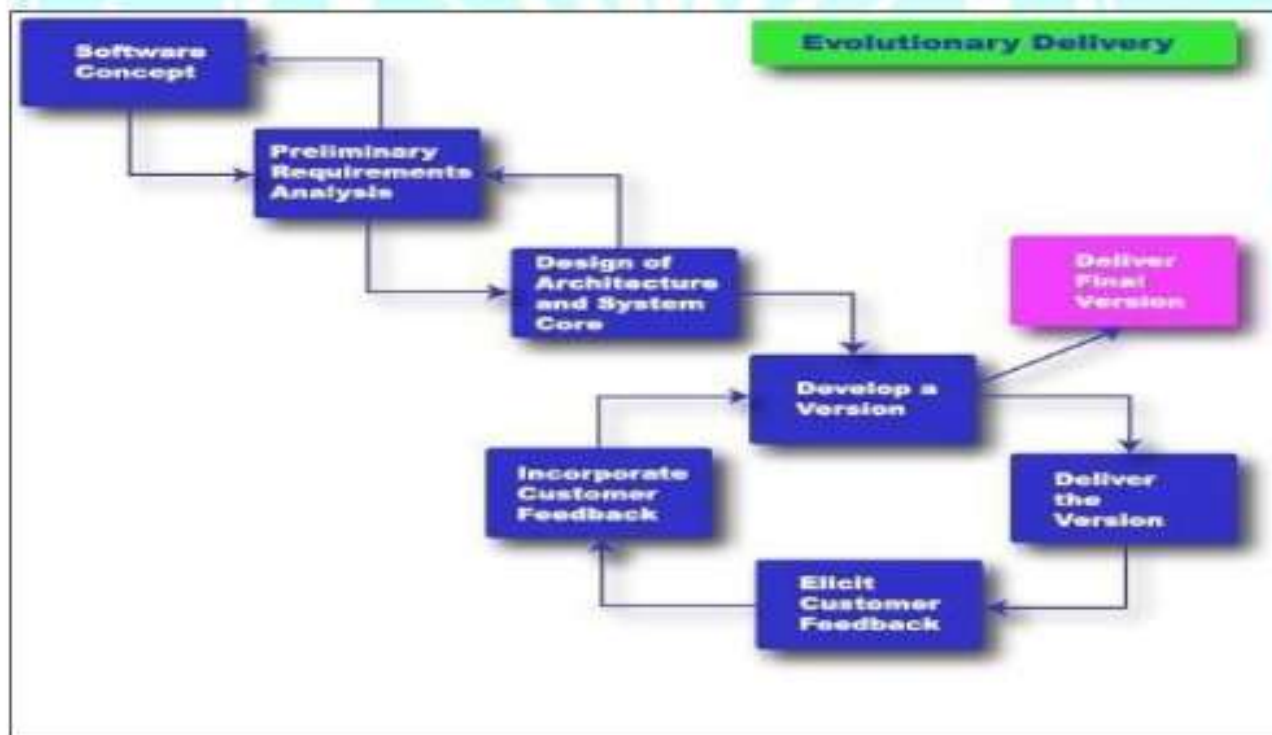


Fig 3.3: Evolutionary Model

## SPIRAL MODEL

The Spiral model of software development is shown in fig. 4.1. The diagrammatic representation of this model appears like a spiral with many loops. The exact number of loops in the spiral is not fixed. Each loop of the spiral represents a phase of the software process. For example, the innermost loop might be concerned with feasibility study, the next loop with requirements specification, the next one with design, and so on. Each phase in this model is split into four sectors (or quadrants) as shown in fig. 4.1. The following activities are carried out during each phase of a spiral model.

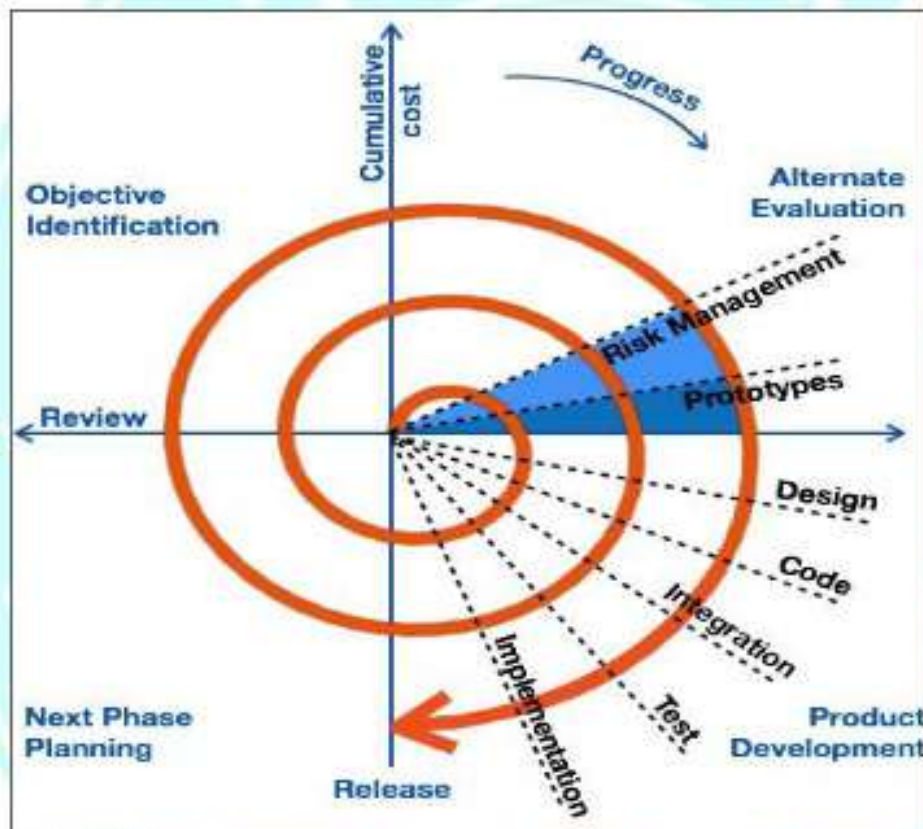


Fig 4.1: Spiral Model

### First quadrant (Objective Setting)

- During the first quadrant, it is needed to identify the objectives of the phase.
- Examine the risks associated with these objectives.



### **Second Quadrant (Risk Assessment and Reduction)**

- A detailed analysis is carried out for each identified project risk.
- Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

### **Third Quadrant (Development and Validation)**

- Develop and validate the next level of the product after resolving the identified risks.

### **Fourth Quadrant (Review and Planning)**

- Review the results achieved so far with the customer and plan the next iteration around the spiral.
- Progressively more complete version of the software gets built with each iteration around the spiral.

### **Circumstances to use spiral model**

The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects.

### **Comparison of different life-cycle models**

The classical waterfall model can be considered as the basic model and all other life cycle models as embellishments of this model. However, the classical waterfall model cannot be used in practical development projects, since this model supports no mechanism to handle the errors committed during any of the phases.

This problem is overcome in the iterative waterfall model. The iterative waterfall model is probably the most widely used software development model evolved so far. This model is simple to understand and use. However this model is suitable only for well-understood problems; it is not suitable for very large projects and for projects that are subject to many risks.

The prototyping model is suitable for projects for which either the user requirements or the underlying technical aspects are not well understood. This model is especially popular for development of the user-interface part of the projects.

The evolutionary approach is suitable for large problems which can be decomposed into a set of modules for incremental development and delivery. This model is also widely used for object-oriented development projects. Of course, this model can only be used if the incremental delivery of the system is acceptable to the customer.

The spiral model is called a meta model since it encompasses all other life cycle models. Risk handling is inherently built into this model. The spiral model is suitable for development of technically challenging software products that are prone to several kinds of risks. However, this model is much more complex than the other models – this is probably a factor deterring its use in ordinary projects.



The different software life cycle models can be compared from the viewpoint of the customer. Initially, customer confidence in the development team is usually high irrespective of the development model followed. During the lengthy development process, customer confidence normally drops off, as no working product is immediately visible. Developers answer customer queries using technical slang, and delays are announced. This gives rise to customer resentment. On the other hand, an evolutionary approach lets the customer experiment with a working product much earlier than the monolithic approaches. Another important advantage of the incremental model is that it reduces the customer's trauma of getting used to an entirely new system. The gradual introduction of the product via incremental phases provides time to the customer to adjust to the new product. Also, from the customer's financial viewpoint, incremental development does not require a large upfront capital outlay. The customer can order the incremental versions as and when he can afford them.

#### 4.3. REQUIREMENTS ANALYSIS AND SPECIFICATION

Before we start to develop our software, it becomes quite essential for us to understand and document the exact requirement of the customer. Experienced members of the development team carry out this job. They are called as *system analysts*.

The analyst starts *requirements gathering and analysis* activity by collecting all information from the customer which could be used to develop the requirements of the system. He then analyzes the collected information to obtain a clear and thorough understanding of the product to be developed, with a view to remove all ambiguities and inconsistencies from the initial customer perception of the problem. The following basic questions pertaining to the project should be clearly understood by the analyst in order to obtain a good grasp of the problem:

- What is the problem?
- Why is it important to solve the problem?
- What are the possible solutions to the problem?
- What exactly are the data input to the system and what exactly are the data output by the system?
- What are the likely complexities that might arise while solving the problem?
- If there are external software or hardware with which the developed software has to interface, then what exactly would the data interchange formats with the external system be?

After the analyst has understood the exact customer requirements, he proceeds to identify and resolve the various requirements problems. The most important requirements problems that the analyst has to identify and eliminate are the problems of anomalies, inconsistencies, and incompleteness. When the analyst detects any inconsistencies, anomalies or incompleteness in the gathered requirements, he resolves them by carrying out further discussions with the end-users and the customers.



•The important parts of

SRS document

are: - - -

Functional

requirements of

the system

- Non-functional requirements of the system, and
- Goals of implementation

#### Functional requirements:-

The functional requirements part discusses the functionalities required from the system. The system is considered to perform a set of high-level functions  $\{f\}$ . The functional view of the System is shown in fig. 5.1. Each function  $f$  of the system can be considered as a transformation Of a set of input data ( $i$ ) to the corresponding set of output data ( $o$ ). The user can get some Meaningful piece of work done using a high-level function.

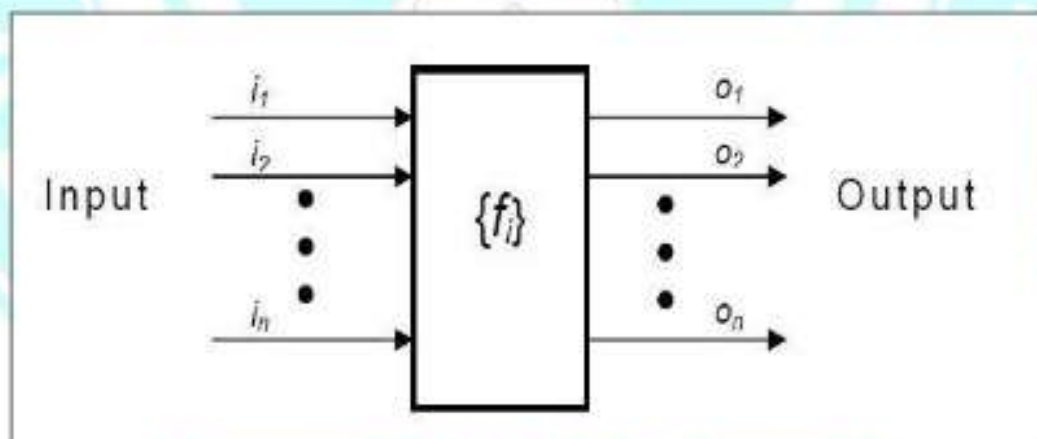


Fig. 5.1: View of a system performing a set of functions

#### Nonfunctional requirements: -

Nonfunctional requirements deal with the characteristics of the system which cannot be expressed as functions - such as the maintainability of the system, portability of the system, usability of the system, etc.

### **Goals of implementation: -**

The goals of implementation part documents some general suggestions regarding development. These suggestions guide trade-off among design goals. The goals of implementation section might document issues such as revisions to the system functionalities that may be required in the future, new devices to be supported in the future, reusability issues, etc. These are the items which the developers might keep in their mind during development so that the developed system may meet some aspects that are not required immediately.

### **Identifying functional requirements from a problem description**

The high-level functional requirements often need to be identified either from an informal problem description document or from a conceptual understanding of the problem. Each high-level requirement characterizes a way of system usage by some user to perform some meaningful piece of work. There can be many types of users of a system and their requirements from the system may be very different. So, it is often useful to identify the different types of users who might use the system and then try to identify the requirements from each user's perspective.

**Example:** - Consider the case of the library system, where –

**F1:** Search Book function

**Input:** an author's name

**Output:** details of the author's books and the location of these books in the library

So the function Search Book (F1) takes the author's name and transforms it into book details.

Functional requirements actually describe a set of high-level requirements, where each high-level requirement takes some data from the user and provides some data to the user as an output. Also each high-level requirement might consist of several other functions.

### **Documenting functional requirements**

For documenting the functional requirements, we need to specify the set of functionalities supported by the system. A function can be specified by identifying the state at which the data is to be input to the system, its input data domain, the output data domain, and the type of processing to be carried on the input data to obtain the output data. Let us first try to document the withdraw-cash function of an ATM (Automated Teller Machine) system. The withdraw-cash is a high-level requirement. It has several sub-requirements corresponding to the different user interactions. These different interaction sequences capture the different scenarios.

**Example:** - Withdraw Cash from ATM

**R1:** withdraw cash



Description: The withdraw cash function first determines the type of account that the user has and the account number from which the user wishes to withdraw cash. It checks the balance to determine whether the requested amount is available in the account. If enough balance is available, it outputs the required cash; otherwise it generates an error message.

R1.1 select withdraw amount option

Input: "withdraw amount" option

Output: user prompted to enter the account type

R1.2: select account type

Input: user option

Output: prompt to enter amount

R1.3: get required amount

Input: amount to be withdrawn in integer values greater than 100 and less than 10,000 in multiples of 100.

Output: The requested cash and printed transaction statement.

Processing: the amount is debited from the user's account if sufficient balance is available, otherwise an error message displayed

### Properties of a good SRS document

The important properties of a good SRS document are the following:

- **Concise.** The SRS document should be concise and at the same time unambiguous, consistent, and complete. Verbose and irrelevant descriptions reduce readability and also increase error possibilities.
- **Structured.** It should be well-structured. A well-structured document is easy to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the customer requirements. Often, the customer requirements evolve over a period of time. Therefore, in order to make the modifications to the SRS document easy, it is important to make the document well-structured.
- **Black-box view.** It should only specify what the system should do and refrain from stating how to do these. This means that the SRS document should specify the external behavior of the system and not discuss the implementation issues. The SRS document should view the system to be developed as black box, and should specify the externally visible behavior of the system. For this reason, the SRS document is also called the black-box specification of a system.
- **Conceptual integrity.** It should show conceptual integrity so that the reader can easily understand it.
- **Response to undesired events.** It should characterize acceptable responses to undesired

events. These are called system response to exceptional conditions.

- **Verifiable.** All requirements of the system as documented in the SRS document should be verifiable. This means that it should be possible to determine whether or not requirements have been met in an implementation.

### **Problems without a SRS document**

The important problems that an organization would face if it does not develop a SRS document are as follows:

- Without developing the SRS document, the system would not be implemented according to customer needs.
- Software developers would not know whether what they are developing is what exactly required by the customer.
- Without SRS document, it will be very much difficult for the maintenance engineers to understand the functionality of the system.
- It will be very much difficult for user document writers to write the users' manuals properly without understanding the SRS document.

### **Problems with an unstructured specification**

- It would be very much difficult to understand that document.
- It would be very much difficult to modify that document.
- Conceptual integrity in that document would not be shown.
- The SRS document might be unambiguous and inconsistent.





#### 4.4. DECISION TREE

A decision tree gives a graphic view of the processing logic involved in decision making and the corresponding actions taken. The edges of a decision tree represent conditions and the leaf nodes represent the actions to be performed depending on the outcome of testing the condition.

**Example: -**

Consider Library Membership Automation Software (LMS) where it should support the following three options:

- New member
- Renewal
- Cancel membership

**New member option-**

**Decision:** When the 'new member' option is selected, the software asks details about the member like the member's name, address, phone number etc.

**Action:** If proper information is entered then a membership record for the member is created and a bill is printed for the annual membership charge plus the security deposit payable.

**Renewal option-**

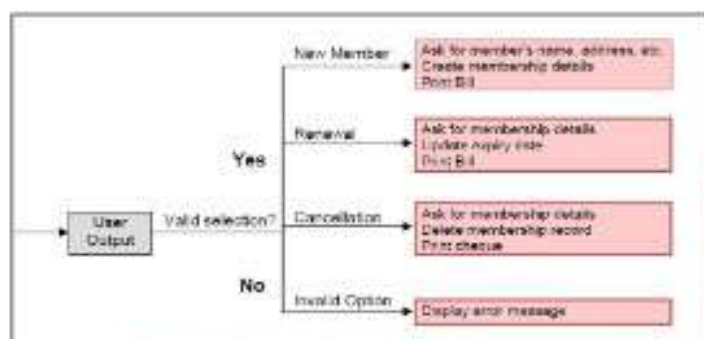
**Decision:** If the 'renewal' option is chosen, the LMS asks for the member's name and his membership number to check whether he is a valid member or not.

**Action:** If the membership is valid then membership expiry date is updated and the annual membership bill is printed, otherwise an error message is displayed.

**Cancel membership option-**

**Decision:** If the 'cancel membership' option is selected, then the software asks for member's name and his membership number.

**Action:** The membership is cancelled, a cheque for the balance amount due to the member is printed and finally the membership record is deleted from the database.



The following tree (fig. 6.1) shows the graphical representation of the above example.

Fig 6.1: Decision Tree of LMS

## DECISION TABLE

A decision table is used to represent the complex processing logic in a tabular or a matrix form. The upper rows of the table specify the variables or conditions to be evaluated. The lower rows of the table specify the actions to be taken when the corresponding conditions are satisfied. A column in a table is called a *rule*. A rule implies that if a condition is true, then the corresponding action is to be executed.

### Example: -

Consider the previously discussed LMS example. The following decision table (fig. 6.2) shows how to represent the LMS problem in a tabular form. Here the table is divided into two parts, the upper part shows the conditions and the lower part shows what actions are taken. Each column of the table is a rule.

Conditions				
Valid selection	No	Yes	Yes	Yes
New member	-	Yes	No	No
Renewal	-	No	Yes	No
Cancellation	-	No	No	Yes
Actions				
Display error message	x	-	-	-
Ask member's details	-	x	-	-
Build customer record	-	x	-	-
Generate bill	-	x	x	-
Ask member's name & membership number	-	-	x	x
Update expiry date	-	-	x	-
Print cheque	-	-	-	x
Delete record	-	-	-	x

Fig. 6.2: Decision table for LMS



From the above table you can easily understand that, if the valid selection condition is false then the action taken for this condition is 'display error message'. Similarly, the actions taken for other conditions can be inferred from the table.

#### 4.5. SOFTWARE DESIGN

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

##### Software Design Levels

Software design yields three levels of results:

**Architectural Design** - The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.

- **High-level Design**- The high-level design breaks the 'single entity-multiple component' concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.
- **Detailed Design**- Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules.

##### Modularization

Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.



Modular design unintentionally follows the rules of 'divide and conquer' problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

Advantage of modularization:

- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again.
- Concurrent execution can be made possible
- Desired from security aspect

### Concurrency

Back in time, all software were meant to be executed sequentially. By sequential execution we mean that the coded instruction will be executed one after another implying only one portion of program being activated at any given time. Say, a software has multiple modules, then only one of all the modules can be found active at any time of execution.

In software design, concurrency is implemented by splitting the software into multiple independent units of execution, like modules and executing them in parallel. In other words, concurrency provides capability to the software to execute more than one part of code in parallel to each other.

It is necessary for the programmers and designers to recognize those modules, which can be made parallel execution.

### Example

The spell check feature in word processor is a module of software, which runs alongside the word processor itself.

### Coupling and Cohesion

When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though, considered as single entity but may refer to each other to work together. There are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion.

### Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

- **Co-incidental cohesion** - It is unplanned and random cohesion, which might be the result



of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.

- **Logical cohesion** - When logically categorized elements are put together into a module, it is called logical cohesion.
- **Temporal Cohesion** - When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.
- **Procedural cohesion** - When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.
- **Communicational cohesion** - When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.
- **Sequential cohesion** - When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.
- **Functional cohesion** - It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.

## Coupling

Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

There are five levels of coupling, namely -

- **Content coupling** - When a module can directly access or modify or refer to the content of another module, it is called content level coupling.
- **Common coupling**- When multiple modules have read and write access to some global data, it is called common or global coupling.
- **Control coupling**- Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.
- **Stamp coupling**- When multiple modules share common data structure and work on different part of it, it is called stamp coupling.
- **Data coupling**- Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

Ideally, no coupling is considered to be the best.

## Design Verification



The output of software design process is design documentation, pseudo codes, detailed logic diagrams, process diagrams, and detailed description of all functional or non-functional requirements.

The next phase, which is the implementation of software, depends on all outputs mentioned above.

It is then becomes necessary to verify the output before proceeding to the next phase. The early any mistake is detected, the better it is or it might not be detected until testing of the product. If the outputs of design phase are in formal notation form, then their associated tools for verification should be used otherwise a thorough design review can be used for verification and validation.

By structured verification approach, reviewers can detect defects that might be caused by overlooking some conditions. A good design review is important for good software design, accuracy and quality.

#### **4.6. SOFTWARE DESIGN STRATEGIES**

Software design is a process to conceptualize the software requirements into software implementation. Software design takes the user requirements as challenges and tries to find optimum solution. While the software is being conceptualized, a plan is chalked out to find the best possible design for implementing the intended solution.

There are multiple variants of software design. Let us study them briefly:

Software design is a process to conceptualize the software requirements into software implementation. Software design takes the user requirements as challenges and tries to find optimum solution. While the software is being conceptualized, a plan is chalked out to find the best possible design for implementing the intended solution.

There are multiple variants of software design. Let us study them briefly:

##### **Structured Design**

Structured design is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. Benefit of structured design is, it gives better understanding of how the problem is being solved. Structured design also makes it simpler for designer to concentrate on the problem more accurately.

Structured design is mostly based on 'divide and conquer' strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.

The small pieces of problem are solved by means of solution modules. Structured design emphasis that these modules be well organized in order to achieve precise solution.

These modules are arranged in hierarchy. They communicate with each other. A good structured design always follows some rules for communication among multiple modules, namely -



**Cohesion** - grouping of all functionally related elements.

**Coupling** - communication between different modules.

A good structured design has **high** cohesion and **low** coupling arrangements.

### Function Oriented Design

In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.

Function oriented design inherits some properties of structured design where divide and conquer methodology is used.

This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation. These functional modules can share information among themselves by means of information passing and using information available globally.

Another characteristic of functions is that when a program calls a function, the function changes the state of the program, which sometimes is not acceptable by other modules. Function oriented design works well where the system state does not matter and program/functions work on input rather than on a state.

### Design Process

- The whole system is seen as how data flows in the system by means of data flow diagram.
- DFD depicts how functions change the data and state of entire system.
- The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.
- Each function is then described at large.

### Object Oriented Design

Object oriented design works around the entities and their characteristics instead of functions involved in the software system. This design strategy focuses on entities and its characteristics. The whole concept of software solution revolves around the engaged entities.

Let us see the important concepts of Object Oriented Design:

- **Objects** - All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes associated to it and has some methods to perform on the attributes.
- **Classes** - A class is a generalized description of an object. An object is an instance of a



class. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object.

In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.

- **Encapsulation** - In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.
- **Inheritance** - OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specific ones.
- **Polymorphism** - OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.

## Design Process

Software design process can be perceived as series of well-defined steps. Though it varies according to design approach (function oriented or object oriented, yet It may have the following steps involved:

- A solution design is created from requirement or previous used system and/or system sequence diagram.
- Objects are identified and grouped into classes on behalf of similarity in attribute characteristics.
- Class hierarchy and relation among them are defined.
- Application framework is defined.

## Software Design Approaches

There are two generic approaches for software designing:

### Top down Design



We know that a system is composed of more than one sub-systems and it contains a number of components. Further, these sub-systems and components may have their one set of sub-system and components and creates hierarchical structure in the system.

Top-down design takes the whole software system as one entity and then decomposes it to achieve more than one sub-system or component based on some characteristics. Each sub-system or component is then treated as a system and decomposed further. This process keeps on running until the lowest level of system in the top-down hierarchy is achieved.

Top-down design starts with a generalized model of system and keeps on defining the more specific part of it. When all components are composed the whole system comes into existence.

Top-down design is more suitable when the software solution needs to be designed from scratch and specific details are unknown.

### **Bottom-up Design**

The bottom up design model starts with most specific and basic components. It proceeds with composing higher level of components by using basic or lower level components. It keeps creating higher level components until the desired system is not evolved as one single component. With each higher level, the amount of abstraction is increased.

Bottom-up strategy is more suitable when a system needs to be created from some existing system, where the basic primitives can be used in the newer system.

Both, top-down and bottom-up approaches are not practical individually. Instead, a good combination of both is used.

## **4.7. SOFTWARE ANALYSIS & DESIGN TOOLS**

Software analysis and design includes all activities, which help the transformation of requirement specification into implementation. Requirement specifications specify all functional and non-functional expectations from the software. These requirement specifications come in the shape of human readable and understandable documents, to which a computer has nothing to do.

Software analysis and design is the intermediate stage, which helps human-readable requirements to be transformed into actual code.

## **OBJECT MODELLING USING UML**

### **Model**

A model captures aspects important for some application while omitting (or abstracting) the rest. A model in the context of software development can be graphical, textual, mathematical, or program code-based. Models are very useful in documenting the design and analysis results.



Models also facilitate the analysis and design procedures themselves. Graphical models are very popular because they are easy to understand and construct. UML is primarily a graphical modeling tool. However, it often requires text explanations to accompany the graphical models.

### **Need for a model**

An important reason behind constructing a model is that it helps manage complexity. Once models of a system have been constructed, these can be used for a variety of purposes during software development, including the following:

- Analysis
- Specification
- Code generation
- Design
- Visualize and understand the problem and the working of a system
- Testing, etc.

In all these applications, the UML models can not only be used to document the results but also to arrive at the results themselves. Since a model can be used for a variety of purposes, it is reasonable to expect that the model would vary depending on the purpose for which it is being constructed. For example, a model developed for initial analysis and specification should be very different from the one used for design. A model that is being used for analysis and specification would not show any of the design decisions that would be made later on during the design stage. On the other hand, a model used for design purposes should capture all the design decisions. Therefore, it is a good idea to explicitly mention the purpose for which a model has been developed, along with the model.

### **Unified Modeling Language (UML)**

UML, as the name implies, is a modeling language. It may be used to visualize, specify, construct, and document the artifacts of a software system. It provides a set of notations (e.g. rectangles, lines, ellipses, etc.) to create a visual model of the system. Like any other language, UML has its own syntax (symbols and sentence formation rules) and semantics (meanings of symbols and sentences). Also, we should clearly understand that UML is not a system design or development methodology, but can be used to document object-oriented and analysis results obtained using some methodology.

### **Origin of UML**

In the late 1980s and early 1990s, there was a proliferation of object-oriented design techniques and notations. Different software development houses were using different notations to document their object-oriented designs. These diverse notations used to give rise to a lot of confusion.

UML was developed to standardize the large number of object-oriented modeling notations that existed and were used extensively in the early 1990s. The principles ones in use were:



- Object Management Technology [Rumbaugh 1991]
- Booch's methodology [Booch 1991]
- Object-Oriented Software Engineering [Jacobson 1992]
- Odell's methodology [Odell 1992]
- Shaler and Mellor methodology [Shaler 1992]

It is needless to say that UML has borrowed many concepts from these modeling techniques. Especially, concepts from the first three methodologies have been heavily drawn upon. UML was adopted by Object Management Group (OMG) as a *de facto* standard in 1997. OMG is an association of industries which tries to facilitate early formation of standards.

We shall see that UML contains an extensive set of notations and suggests construction of many types of diagrams. It has successfully been used to model both large and small problems. The elegance of UML, its adoption by OMG, and a strong industry backing have helped UML find widespread acceptance. UML is now being used in a large number of software development projects worldwide.

## UML Diagrams

UML can be used to construct nine different types of diagrams to capture five different views of a system. Just as a building can be modeled from several views (or perspectives) such as ventilation perspective, electrical perspective, lighting perspective, heating perspective, etc.; the different UML diagrams provide different perspectives of the software system to be developed and facilitate a comprehensive understanding of the system. Such models can be refined to get the actual implementation of the system.

The UML diagrams can capture the following five views of a system:

- User's view
- Structural view
- Behavioral view
- Implementation view
- Environmental view

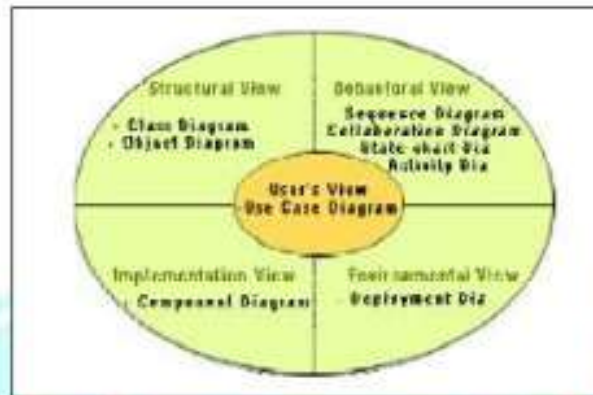


Fig. 12.1: Different types of diagrams and views supported in UML

**User's view:** This view defines the functionalities (facilities) made available by the system to its users. The users' view captures the external users' view of the system in terms of the functionalities offered by the system. The users' view is a black-box view of the system where the internal structure, the dynamic behavior of different system components, the implementation etc. are not visible. The users' view is very different from all other views in the sense that it is a functional model compared to the object model of all other views. The users' view can be considered as the central view and all other views are expected to conform to this view. This thinking is in fact the crux of any user centric development style.

**Structural view:** The structural view defines the kinds of objects (classes) important to the understanding of the working of a system and to its implementation. It also captures the relationships among the classes (objects). The structural model is also called the static model, since the structure of a system does not change with time.

**Behavioral view:** The behavioral view captures how objects interact with each other to realize the system behavior. The system behavior captures the time-dependent (dynamic) behavior of the system.

**Implementation view:** This view captures the important components of the system and their dependencies.

**Environmental view:** This view models how the different components are implemented on different pieces of hardware.

## USE CASE DIAGRAM

### Use Case Model

The use case model for any system consists of a set of "use cases". Intuitively, use cases represent the different ways in which a system can be used by the users. A simple way to find all the use cases of a system is to ask the question: "What the users can do using the system?" Thus for the Library Information System (LIS), the use cases could be:



- issue-book
- query-book
- return-book
- create-member
- add-book, etc

Use cases correspond to the high-level functional requirements. The use cases partition the system behavior into transactions, such that each transaction performs some useful action from the user's point of view. To complete each transaction may involve either a single message or multiple message exchanges between the user and the system to complete.

#### Purpose of use cases

The purpose of a use case is to define a piece of coherent behavior without revealing the internal structure of the system. The use cases do not mention any specific algorithm to be used or the internal data representation, internal structure of the software, etc. A use case typically represents a sequence of interactions between the user and the system. These interactions consist of one mainline sequence. The mainline sequence represents the normal interaction between a user and the system. The mainline sequence is the most occurring sequence of interaction. For example, the mainline sequence of the withdraw cash use case supported by a bank ATM drawn, complete the transaction, and get the amount. Several variations to the main line sequence may also exist. Typically, a variation from the mainline sequence occurs when some specific conditions hold. For the bank ATM example, variations or alternate scenarios may occur, if the password is invalid or the amount to be withdrawn exceeds the amount balance. The variations are also called alternative paths. A use case can be viewed as a set of related scenarios tied together by a common goal. The mainline sequence and each of the variations are called scenarios or instances of the use case. Each scenario is a single path of user events and system activity through the use case.

### Representation of Use Cases

Use cases can be represented by drawing a use case diagram and writing an accompanying text elaborating the drawing. In the use case diagram, each use case is represented by an ellipse with the name of the use case written inside the ellipse. All the ellipses (i.e. use cases) of a system are enclosed within a rectangle which represents the system boundary. The name of the system being modeled (such as Library Information System) appears inside the rectangle.

The different users of the system are represented by using the stick person icon. Each stick person icon is normally referred to as an actor. An actor is a role played by a user with respect to the system use. It is possible that the same user may play the role of multiple actors. Each actor can participate in one or more use cases. The line connecting the actor and the use case is called



the communication relationship. It indicates that the actor makes use of the functionality provided by the use case. Both the human users and the external systems can be represented by stick person icons. When a stick person icon represents an external system, it is annotated by the stereotype <<external system>>.

### Example 1:

#### Tic-Tac-Toe Computer Game

Tic-tac-toe is a computer game in which a human player and the computer make alternative moves on a 3×3 square. A move consists of marking previously unmarked square. The player who first places three consecutive marks along a straight line on the square (i.e. along a row, column, or diagonal) wins the game. As soon as either the human player or the computer wins, a message congratulating the winner should be displayed. If neither player manages to get three consecutive marks along a straight line, but all the squares on the board are filled up, then the game is drawn. The computer always tries to win a game.

The use case model for the Tic-tac-toe problem is shown in fig. 13.1. This software has only one use case “play move”. Note that the use case “get-user-move” is not used here. The name “get-user-move” would be inappropriate because the use cases should be named from the user’s perspective.

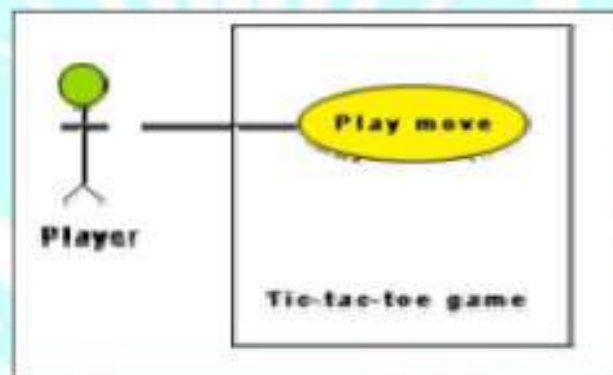


Fig. 13.1: Use case model for tic-tac-toe game

### Text Description

Each ellipse on the use case diagram should be accompanied by a text description. The text description should define the details of the interaction between the user and the computer and other aspects of the use case. It should include all the behavior associated with the use case in terms of the mainline sequence, different variations to the normal behavior, the system responses associated with the use case, the exceptional conditions that may occur in the behavior, etc. The behavior description is often written in a conversational style describing the interactions between



the actor and the system. The text description may be informal, but some structuring is recommended. The following are some of the information which may be included in a use case text description in addition to the mainline sequence, and the alternative scenarios.

**Contact persons:** This section lists the personnel of the client organization with whom the use case was discussed, date and time of the meeting, etc.

**Actors:** In addition to identifying the actors, some information about actors using this use case which may help the implementation of the use case may be recorded.

**Pre-condition:** The preconditions would describe the state of the system before the use case execution starts.

**Post-condition:** This captures the state of the system after the use case has successfully completed.

**Non-functional requirements:** This could contain the important constraints for the design and implementation, such as platform and environment conditions, qualitative statements, response time requirements, etc.

**Exceptions, error situations:** This contains only the domain-related errors such as lack of user's access rights, invalid entry in the input fields, etc. Obviously, errors that are not domain related, such as software errors, need not be discussed here.

**Sample dialogs:** These serve as examples illustrating the use case.

**Specific user interface requirements:** These contain specific requirements for the user interface of the use case. For example, it may contain forms to be used, screen shots, interaction style, etc.

**Document references:** This part contains references to specific domain-related documents which may be useful to understand the system operation

### Example 2:

A supermarket needs to develop the following software to encourage regular customers. For this, the customer needs to supply his/her residence address, telephone number, and the driving license number. Each customer who registers for this scheme is assigned a unique customer number (CN) by the computer. A customer can present his CN to the checkout staff when he makes any purchase. In this case, the value of his purchase is credited against his CN. At the end of

each year, the supermarket intends to award surprise gifts to 10 customers who make the highest total purchase over the year. Also, it intends to award a 22 carat gold coin to every customer whose purchase exceeded Rs.10,000. The entries against the CN are the reset on the day of every year after the prize winners' lists are generated.

The use case model for the Supermarket Prize Scheme is shown in fig. 13.2. As discussed earlier, the use cases correspond to the high-level functional requirements. From the problem description, we can identify three use cases: "register-customer", "register-sales", and "select-winners". As a sample, the text description for the use case "register-customer" is shown.

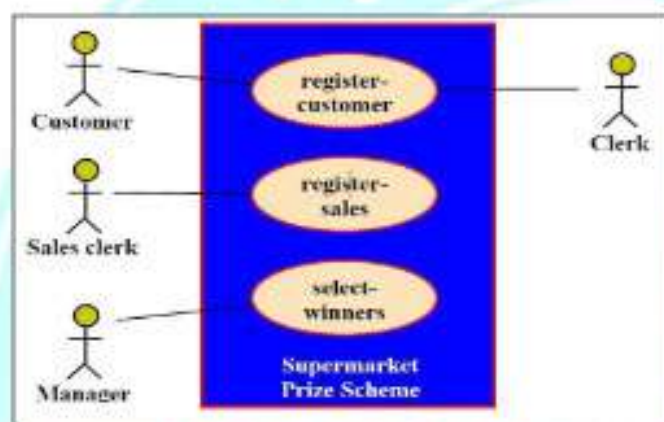


Fig. 13.2 Use case model for Supermarket Prize Scheme

### Text description

U1: register-customer: Using this use case, the customer can register himself by providing the necessary details.

### Scenario 1: Mainline sequence

1. Customer: select register customer option.
2. System: display prompt to enter name, address, and telephone number.  
Customer: enter the necessary values.
4. System: display the generated id and the message that the customer has been successfully registered.

### Scenario 2: at step 4 of mainline sequence

1. System: displays the message that the customer has already registered.

### Scenario 2: at step 4 of mainline sequence

1. System: displays the message that some input information has not been entered. The system displays a prompt to enter the missing value.



The description for other use cases is written in a similar fashion.

### **Utility of use case diagrams**

From use case diagram, it is obvious that the utility of the use cases are represented by ellipses. They along with the accompanying text description serve as a type of requirements specification of the system and form the core model to which all other models must conform. But, what about the actors (stick person icons)? One possible use of identifying the different types of users (actors) is in identifying and implementing a security mechanism through a login system, so that each actor can involve only those functionalities to which he is entitled to. Another possible use is in preparing the documentation (e.g. users' manual) targeted at each category of user. Further, actors help in identifying the use cases and understanding the exact functioning of the system.

### **Factoring of use cases**

It is often desirable to factor use cases into component use cases. Actually, factoring of use cases are required under two situations. First, complex use cases need to be factored into simpler use cases. This would not only make the behavior associated with the use case much more comprehensible, but also make the corresponding interaction diagrams more tractable. Without decomposition, the interaction diagrams for complex use cases may become too large to be accommodated on a single sized (A4) paper. Secondly, use cases need to be factored whenever there is common behavior across different use cases. Factoring would make it possible to define such behavior only once and reuse it whenever required. It is desirable to factor out common usage such as error handling from a set of use cases. This makes analysis of the class design much simpler and elegant. However, a word of caution here. Factoring of use cases should not be done except for achieving the above two objectives. From the design point of view, it is not advantageous to break up a use case into many smaller parts just for the sake of it.

UML offers three mechanisms for factoring of use cases as follows:

#### **1. Generalization**

Use case generalization can be used when one use case that is similar to another, but does something slightly differently or something more. Generalization works the same way with use cases as it does with classes. The child use case inherits the behavior and meaning of the parent use case. The notation is the same too (as shown in fig. 13.3). It is important to remember that the base and the derived use cases are separate use cases and should have separate text descriptions.





Fig. 13.3: Representation of use case generalization

### Includes

The includes relationship in the older versions of UML (prior to UML 1.1) was known as the uses relationship. The includes relationship involves one use case including the behavior of another use case in its sequence of events and actions. The includes relationship occurs when a chunk of behavior that is similar across a number of use cases. The factoring of such behavior will help in not repeating the specification and implementation across different use cases. Thus, the includes relationship explores the issue of reuse by factoring out the commonality across use cases. It can also be gainfully employed to decompose a large and complex use cases into more manageable parts. As shown in fig. 13.4 the includes relationship is represented using a predefined stereotype <<include>>. In the includes relationship, a base use case compulsorily and automatically includes the behavior of the common use cases. As shown in example fig. 13.5, issue-book and renew-book both include check-reservation use case. The base use case may include several use cases. In such cases, it may interleave their associated common use cases together. The common use case becomes a separate use case and the independent text description should be provided for it.



Fig. 13.4 Representation of use case inclusion



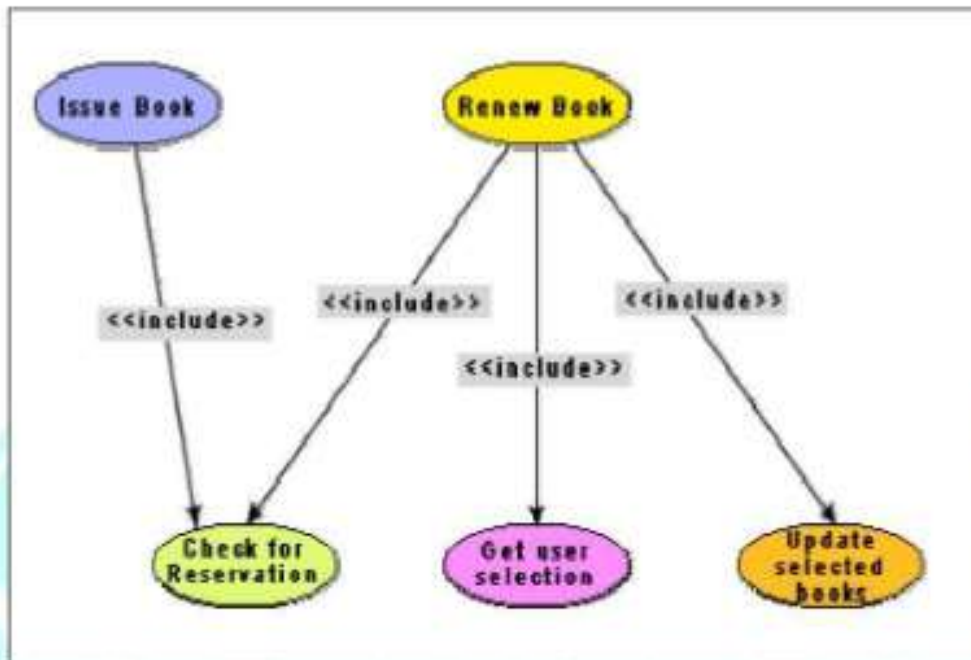


Fig. 13.5: Example use case inclusion

### Extends

The main idea behind the *extends* relationship among the use cases is that it allows you to show optional system behavior. An optional system behavior is extended only under certain conditions. This relationship among use cases is also predefined as a stereotype as shown in fig. 13.6. The *extends* relationship is similar to generalization. But unlike generalization, the extending use case can add additional behavior only at an extension point only when certain conditions are satisfied. The extension points are points within the use case where variation to the mainline (normal) action sequence may occur. The *extends* relationship is normally used to capture alternate paths or scenarios.



Fig. 13.6: Example use case extension

## Organization of use cases

When the use cases are factored, they are organized hierarchically. The high-level use cases are refined into a set of smaller and more refined use cases as shown in fig. 13.7. Top-level use cases are super-ordinate to the refined use cases. The refined use cases are sub-ordinate to the top-level use cases. Note that only the complex use cases should be decomposed and organized in a hierarchy. It is not necessary to decompose simple use cases. The functionality of the super-ordinate use cases is traceable to their sub-ordinate use cases. Thus, the functionality provided by the super-ordinate use cases is composite of the functionality of the sub-ordinate use cases. In the highest level of the use case model, only the fundamental use cases are shown. The focus is on the application context. Therefore, this level is also referred to as the context diagram. In the context diagram, the system limits are emphasized. In the top-level diagram, only those use cases with which external users of the system. The subsystem-level use cases specify the services offered by the subsystems. Any number of levels involving the subsystems may be utilized. In the lowest level of the use case hierarchy, the class-level use cases specify the functional fragments or operations offered by the classes.

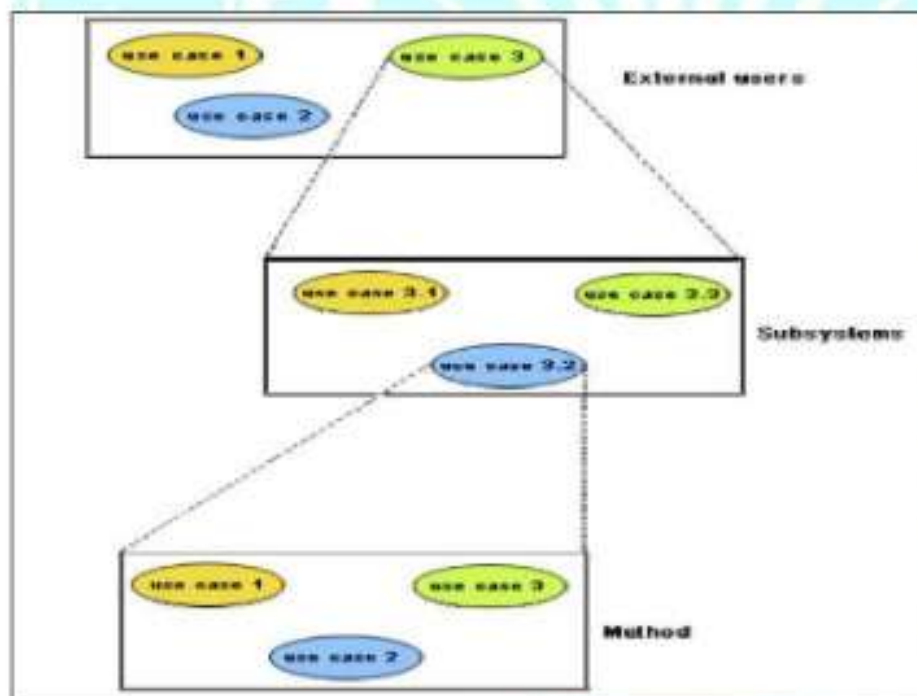


Fig. 13.7: Hierarchical organization of use cases



## CLASS DIAGRAMS

A class diagram describes the static structure of a system. It shows how a system is structured rather than how it behaves. The static structure of a system comprises of a number of class diagrams and their dependencies. The main constituents of a class diagram are classes and their relationships: generalization, aggregation, association, and various kinds of dependencies.

### Classes

The classes represent entities with common features, i.e. attributes and operations. Classes are represented as solid outline rectangles with compartments. Classes have a mandatory name compartment where the name is written centered in boldface. The class name is usually written using mixed case convention and begins with an uppercase. The class names are usually chosen to be singular nouns. Classes have optional attributes and operations compartments. A class may appear on several diagrams. Its attributes and operations are suppressed on all but one diagram.

### Attributes

An attribute is a named property of a class. It represents the kind of data that an object might contain. Attributes are listed with their names, and may optionally contain specification of their type, an initial value, and constraints. The type of the attribute is written by appending a colon and the type name after the attribute name. Typically, the first letter of a class name is a small letter. An example for an attribute is given.

**bookName : String**

### Operation

Operation is the implementation of a service that can be requested from any object of the class to affect behaviour. An object's data or state can be changed by invoking an operation of the object. A class may have any number of operations or no operation at all. Typically, the first letter of an operation name is a small letter. Abstract operations are written in italics. The parameters of an operation (if any), may have a kind specified, which may be 'in', 'out' or 'inout'. An operation may have a return type consisting of a single return type expression. An example for an operation is given.

**issueBook(in bookName):Boolean**

### Association

Associations are needed to enable objects to communicate with each other. An association describes a connection between classes. The association relation between two objects is called object connection or link. Links are instances of associations. A link is a physical or conceptual connection between object instances. For example, suppose Amit has borrowed the book Graph Theory. Here,





Borrowed is the connection between the objects Amit and Graph Theory book. Mathematically, a link can be considered to be a tuple, i.e. an ordered list of object instances. An association describes a group of links with a common structure and common semantics. For example, consider the statement that Library Member borrows Books. Here, borrows is the association between the class LibraryMember and the class Book. Usually, an association is a binary relation (between two classes). However, three or more different classes can be involved in an association. A class can have an association relationship with itself (called recursive association). In this case, it is usually assumed that two different objects of the class are linked by the association relationship. Association between two classes is represented by drawing a straight line between the concerned classes.

Fig. 14.1 illustrates the graphical representation of the association relation. The name of the association is written alongside the association line. An arrowhead may be placed on the association line to indicate the reading direction of the association. The arrowhead should not be misunderstood to be indicating the direction of a pointer implementing an association. On each side of the association relation, the multiplicity is noted as an individual number or as a value range. The multiplicity indicates how many instances of one class are associated with each other. Value ranges of multiplicity are noted by specifying the minimum and maximum value, separated by two dots, e.g. 1..5. An asterisk is a wild card and means many (zero or more). The association of fig. 14.1 should be read as "Many books may be borrowed by a Library Member". Observe that associations (and links) appear as verbs in the problem statement.

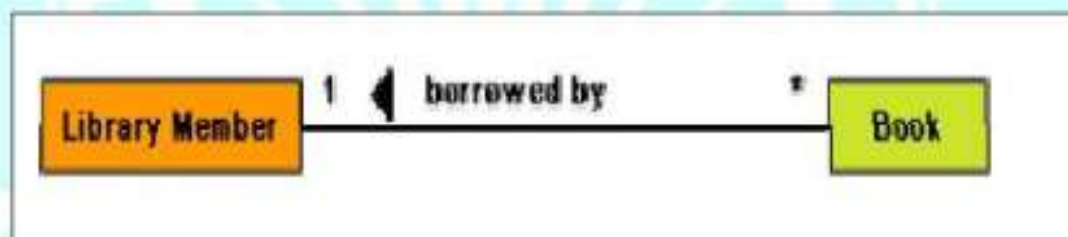


Fig. 14.1: Association between two classes

Associations are usually realized by assigning appropriate reference attributes to the classes involved. Thus, associations can be implemented using pointers from one object class to another. Links and associations can also be implemented by using a separate class that stores which objects of a class are linked to which objects of another class. Some CASE tools use the role names of the association relation for the corresponding automatically generated attribute.

## Aggregation

Aggregation is a special type of association where the involved classes represent a whole-part relationship. The aggregate takes the responsibility of forwarding messages to the appropriate parts. Thus, the aggregate takes the responsibility of delegation and leadership. When an instance of one object contains instances of some other objects, then aggregation (or composition) relationship exists between the composite object and the component object. Aggregation is represented by the diamond



Symbol at the composite end of a relationship. The number of instances of the component class aggregated can also be shown as in fig. 14.2

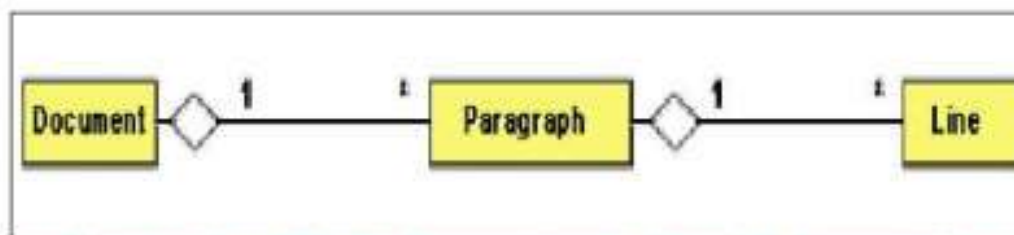


Fig. 14.2: Representation of aggregation

Aggregation relationship cannot be reflexive (i.e. recursive). That is, an object cannot contain objects of the same class as itself. Also, the aggregation relation is not symmetric. That is, two classes A and B cannot contain instances of each other. However, the aggregation relationship can be transitive. In this case, aggregation may consist of an arbitrary number of levels.

### Composition

Composition is a stricter form of aggregation, in which the parts are existence-dependent on the whole. This means that the life of the parts closely ties to the life of the whole. When the whole is created, the parts are created and when the whole is destroyed, the parts are destroyed. A typical example of composition is an invoice object with invoice items. As soon as the invoice object is created, all the invoice items in it are created and as soon as the invoice object is destroyed, all invoice items in it are also destroyed. The composition relationship is represented as a filled diamond drawn at the composite-end. An example of the composition relationship is shown in fig. 14.3



Fig 14.3: Representation of composition

### Association vs. Aggregation vs. Composition

- Association is the most general (m:n) relationship. Aggregation is a stronger relationship where one is a part of the other. Composition is even stronger than aggregation, ties the lifecycle of the part and the whole together.
- Association relationship can be reflexive (objects can have relation to itself), but aggregation cannot be reflexive. Moreover, aggregation is anti-symmetric (If B is a part of A, A cannot be a part of B).
- Composition has the property of exclusive aggregation i.e. an object can be a part of



only one composite at a time. For example, a **Frame** belongs to exactly one **Window**



whereas in simple aggregation, a part may be shared by several objects. For example, a **Wall** may be a part of one or more **Room** objects.

- In addition, in composition, the whole has the responsibility for the disposition of all its parts, i.e. for their creation and destruction.
    - in general, the lifetime of parts and composite coincides
    - parts with non-fixed multiplicity may be created after composite itself
    - parts might be explicitly removed before the death of the composite
- For example, when a **Frame** is created, it has to be attached to an enclosing **Window**. Similarly, when the **Window** is destroyed, it must in turn destroy its **Frame** parts.

### Inheritance vs. Aggregation/Composition

- Inheritance describes 'is a' / 'is a kind of' relationship between classes (base class - derived class) whereas aggregation describes 'has a' relationship between classes. Inheritance means that the object of the derived class inherits the properties of the base class; aggregation means that the object of the whole has objects of the part. For example, the relation "cash payment *is a kind of* payment" is modeled using inheritance; "purchase order has a few items" is modeled using aggregation.
- Inheritance is used to model a "generic-specific" relationship between classes whereas aggregation/composition is used to model a "whole-part" relationship between classes.
- Inheritance means that the objects of the subclass can be used anywhere the super class may appear, but not the reverse; i.e. wherever we could use instances of 'payment' in the system, we could substitute it with instances of 'cash payment', but the reverse cannot be done.
- Inheritance is defined statically. It cannot be changed at run-time. Aggregation is defined dynamically and can be changed at run-time. Aggregation is used when the type of the object can change over time.

For example, consider this situation in a business system. A **BusinessPartner** might be a **Customer** or a **Supplier** or both. Initially we might be tempted to model it as in Fig 14.4(a). But in fact, during its lifetime, a business partner might become a customer as well as a supplier, or it might change from one to the other. In such cases, we prefer aggregation instead (see Fig 14.4(b). Here, a business partner is a **Customer** if it has an aggregated **Customer** object, a **Supplier** if it has an aggregated **Supplier** object and a "**CustomerSupplier**" if it has both. Here, we have only two types. Hence, we are able to model it as inheritance. But what if there were several different types and combinations thereof? The inheritance tree would be absolutely incomprehensible.

Also, the aggregation model allows the possibility for a business partner to be neither - i.e. has neither a customer nor a supplier object aggregated with it.



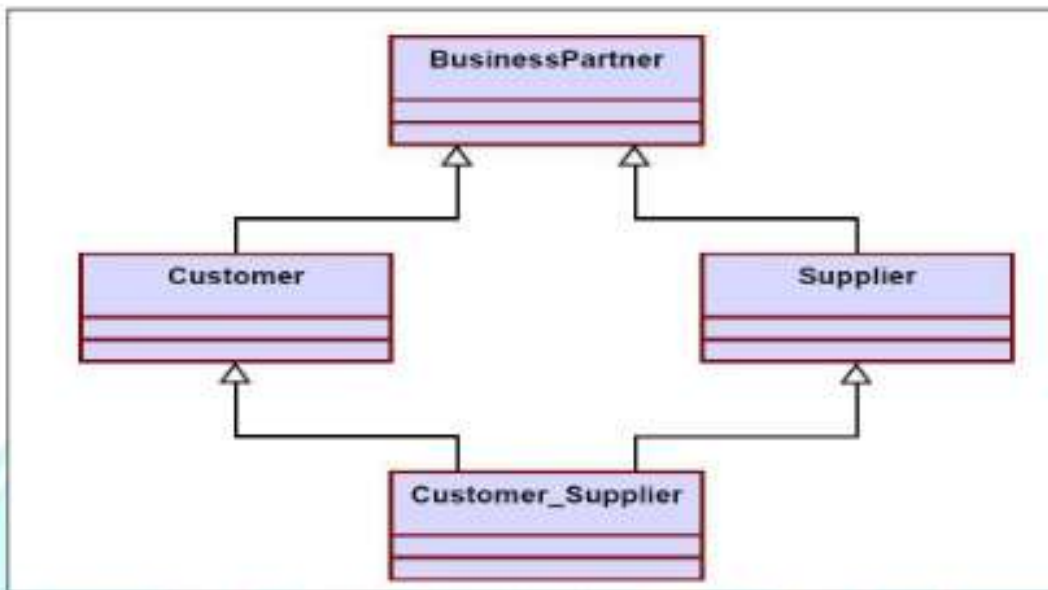


Fig. 14.4 a) Representation of **BusinessPartner**, **Customer**, **Supplier** relationship using inheritance

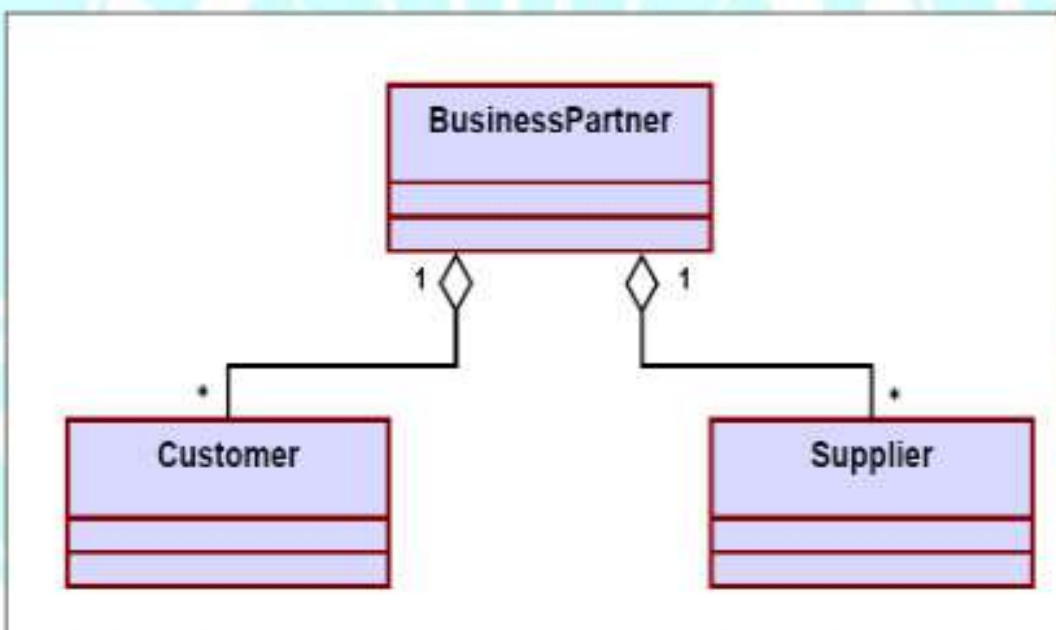


Fig. 14.4 b) Representation of **BusinessPartner**, **Customer**, **Supplier** relationship using aggregation

- The advantage of aggregation is the integrity of encapsulation. The operations of an object are the interfaces of other objects which imply low implementation dependencies. The significant disadvantage of aggregation is the increase in the number of objects and their relationships. On the other hand, inheritance allows for an easy way to modify implementation for reusability. But the significant disadvantage is that it breaks encapsulation, which implies implementation dependence.

## INTERACTION DIAGRAMS

Interaction diagrams are models that describe how group of objects collaborate to realize some behavior. Typically, each interaction diagram realizes the behavior of a single use case. An interaction diagram shows a number of example objects and the messages that are passed between the objects within the use case.

There are two kinds of interaction diagrams: sequence diagrams and collaboration diagrams. These two diagrams are equivalent in the sense that any one diagram can be derived automatically from the other. However, they are both useful. These two actually portray different perspectives of behavior of the system and different types of inferences can be drawn from them. The interaction diagrams can be considered as a major tool in the design methodology.

### Sequence Diagram

A sequence diagram is a Unified Modeling Language (UML) diagram that illustrates the sequence of messages between objects in an interaction. A sequence diagram consists of a group of objects that are represented by lifelines, and the messages that they exchange over time during the interaction. The collaboration diagram for the example is shown in fig. 15.1.

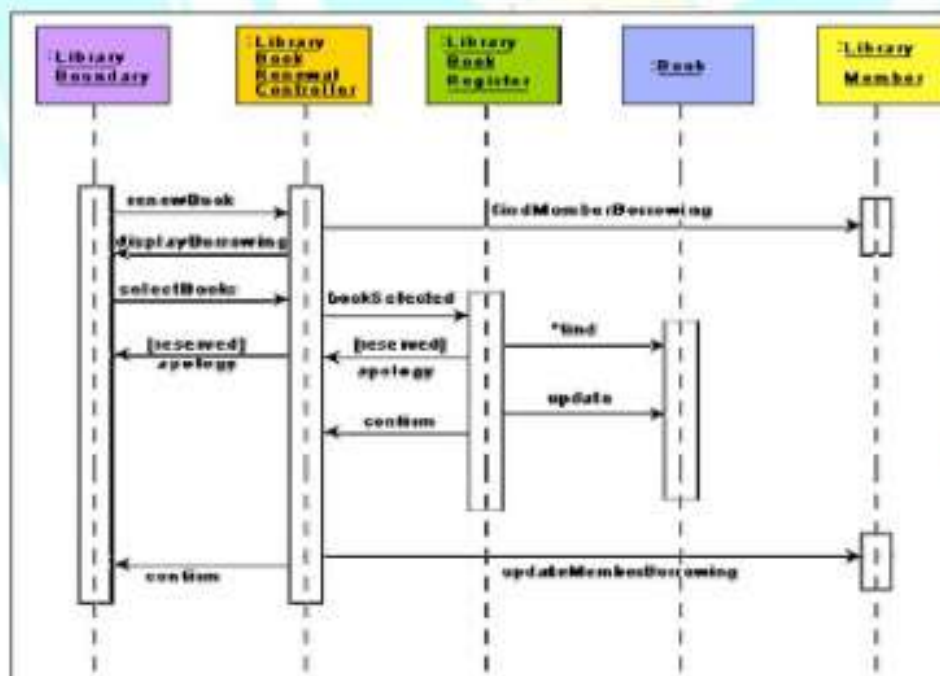


Fig. 15.1: Sequence diagram for the renew book use case



## Collaboration Diagram

A collaboration diagram shows both structural and behavioral aspects explicitly. This is unlike a sequence diagram which shows only the behavioral aspects. The structural aspect of a collaboration diagram consists of objects and the links existing between them. In this diagram, an object is also called a collaborator. The behavioral aspect is described by the set of messages exchanged among the different collaborators. The link between objects is shown as a solid line and can be used to send messages between two objects. The message is shown as a labeled arrow placed near the link. Messages are prefixed with sequence numbers because they are only way to describe the relative sequencing of the messages in this diagram. The collaboration diagram for the example of fig. 15.1 is shown in fig. 15.2. The use of the collaboration diagrams in our development process would be to help us to determine which classes are associated with which other classes.

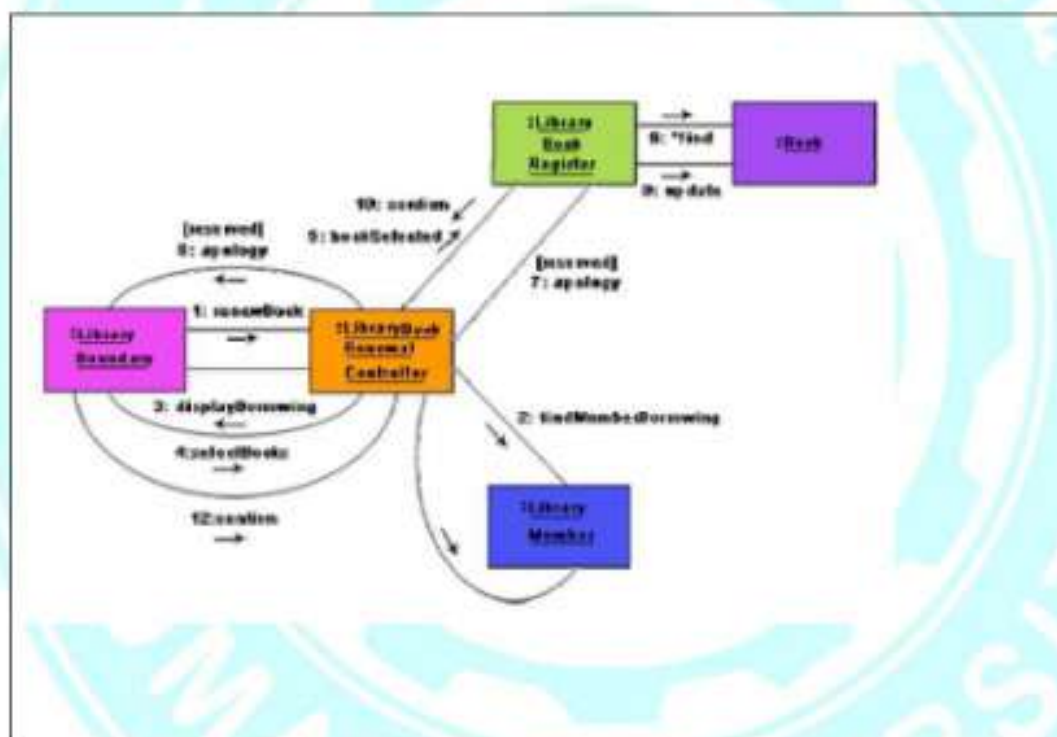


Fig 15.2: Collaboration diagram for the renew book use case

## ACTIVITY AND STATE CHART DIAGRAM

The activity diagram is possibly one modeling element which was not present in any of the predecessors of UML. The activity diagram focuses on representing activities or chunks of processing which may or may not correspond to the methods of classes. An activity is a state with an internal action and one or more outgoing transitions which automatically follow the termination of the internal activity. If an activity has more than one outgoing transitions, then these must be identified through conditions. An interesting feature of the activity diagrams is the swim lanes.

Activity diagrams are normally employed in business process modeling. This is carried out during the initial stages of requirements analysis and specification. Activity diagrams can be very useful to understand complex processing activities involving many components. Later these diagrams can be used to develop interaction diagrams which help to allocate activities (responsibilities) to classes.

The student admission process in a university is shown as an activity diagram in fig. 16.1. This shows the part played by different components of the Institute in the admission procedure. After the fees are received at the account section, parallel activities start at the hostel office, hospital, and the Department. After all these activities complete (this synchronization is represented as a horizontal line), the identity card can be issued to a student by the Academic section.

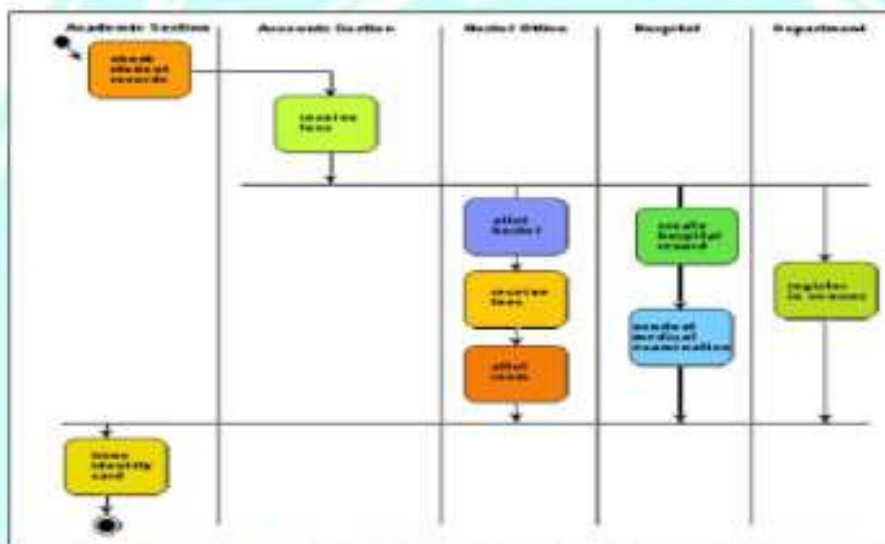


Fig. 16.1: Activity diagram for student admission procedure at a university

### Activity diagrams vs. procedural flow charts

Activity diagrams are similar to the procedural flow charts. The difference is that activity diagrams support description of parallel activities and synchronization aspects involved in different activities.

### STATE CHART DIAGRAM

A state chart diagram is normally used to model how the state of an object changes in its lifetime. State chart diagrams are good at describing how the behavior of an object changes across several use case executions. However, if we are interested in modeling some behavior that involves several objects collaborating with each other, state chart diagram is not appropriate. State chart diagrams are based on the finite state machine (FSM) formalism.



Actions are associated with transitions and are considered to be processes that occur quickly and are not interruptible. Activities are associated with states and can take longer. An activity can be interrupted by an event.

The basic elements of the state chart diagram are as follows:

- **Initial state**- This is represented as a filled circle.
- **Final state**- This is represented by a filled circle inside a larger circle.
- **State**- These are represented by rectangles with rounded corners.
- **Transition**- A transition is shown as an arrow between two states. Normally, the name of the event which causes the transition is placed alongside the arrow. A guard to the transition can also be assigned. A guard is a Boolean logic condition. The transition can take place only if the guard evaluates to true. The syntax for the label of the transition is shown in 3 parts: event [guard]/action.

An example state chart for the order object of the Trade House Automation software is shown in fig. 16.2.

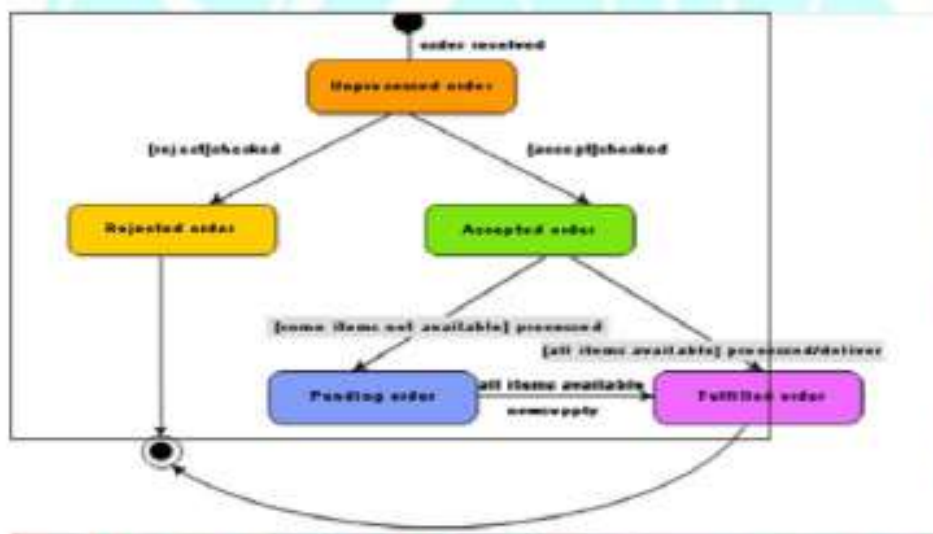


Fig. 16.2: State chart diagram for an order object

#### Activity diagram vs. State chart diagram

- Both activity and state chart diagrams model the dynamic behavior of the system. Activity diagram is essentially a flowchart showing flow of control from activity to activity. A state chart diagram shows a state machine emphasizing the flow of control from state to state.
- An activity diagram is a special case of a state chart diagram in which all or most of the states are activity states and all or most of the transitions are triggered by completion of activities in the source state (An activity is an ongoing non-atomic execution within a state machine).



- Activity diagrams may stand alone to visualize, specify, and document the dynamics of a society of objects or they may be used to model the flow of control of an operation. State chart diagrams may be attached to classes, use cases, or entire systems in order to visualize, specify, and document the dynamics of an individual object.

#### 4.8. CODING

**Coding-** The objective of the coding phase is to transform the design of a system into code in a high level language and then to unit test this code. The programmers adhere to standard and well defined style of coding which they call their coding standard. The main advantages of adhering to a standard style of coding are as follows:

- A coding standard gives uniform appearances to the code written by different engineers
- It facilitates code of understanding.
- Promotes good programming practices.

For implementing our design into a code, we require a good high level language. A programming language should have the following features:

##### Characteristics of a Programming Language

- **Readability:** A good high-level language will allow programs to be written in some ways that resemble a quite-English description of the underlying algorithms. If care is taken, the coding may be done in a way that is essentially self-documenting.
- **Portability:** High-level languages, being essentially machine independent, should be able to develop portable software.
- **Generality:** Most high-level languages allow the writing of a wide variety of programs, thus relieving the programmer of the need to become expert in many diverse languages.
- **Brevity:** Language should have the ability to implement the algorithm with less amount of code. Programs expressed in high-level languages are often considerably shorter than their low-level equivalents.
- **Error checking:** Being human, a programmer is likely to make many mistakes in the development of a computer program. Many high-level languages enforce a great deal of error checking both at compile-time and at run-time.
- **Cost:** The ultimate cost of a programming language is a function of many of its characteristics.
- **Familiar notation:** A language should have familiar notation, so it can be understood by most of the programmers.
- **Quick translation:** It should admit quick translation.
- **Efficiency:** It should permit the generation of efficient object code.
- **Modularity:** It is desirable that programs can be developed in the language as a collection of separately compiled modules, with appropriate mechanisms for ensuring self-consistency between these modules.
- **Widely available:** Language should be widely available and it should be possible to



provide translators for all the major machines and for all the major operating systems. A coding standard lists several rules to be followed during coding, such as the way variables are to be named, the way the code is to be laid out, error return conventions, etc.

### Coding standards and guidelines

Good software development organizations usually develop their own coding standards and guidelines depending on what best suits their organization and the type of products they develop.

The following are some representative coding standards.

1. **Rules for limiting the use of global:** These rules list what types of data can be declared global and what cannot.
2. **Contents of the headers preceding codes for different modules:** The information contained in the headers of different modules should be standard for an organization. The exact format in which the header information is organized in the header can also be specified. The following are some standard header data:
  - Name of the module.
  - Date on which the module was created.
  - Modification history.
  - Synopsis of the module.
  - Different functions supported, along with their input/output parameters.
  - Global variables accessed/modified by the module.
3. **Naming conventions for global variables, local variables, and constant identifiers:** A possible naming convention can be that global variable names always start with a capital letter, local variable names are made of small letters, and constant names are always capital letters.
4. **Error return conventions and exception handling mechanisms:** The way error conditions are reported by different functions in a program are handled should be standard within an organization. For example, different functions while encountering an error condition should either return a 0 or 1 consistently.

The following are some representative coding guidelines recommended by many software development organizations.

1. **Do not use a coding style that is too clever or too difficult to understand:** Code should be easy to understand. Many inexperienced engineers actually take pride in writing cryptic and incomprehensible code. Clever coding can obscure meaning of the code and hamper understanding. It also makes maintenance difficult.
2. **Avoid obscure side effects:** The side effects of a function call include modification of parameters passed by reference, modification of global variables, and I/O operations. An obscure side effect is one that is not obvious from a casual examination of the code.



Obscure side effects make it difficult to understand a piece of code. For example, if a global variable is changed obscurely in a called module or some file I/O is performed which is difficult to infer from the function's name and header information, it becomes difficult for anybody trying to understand the code.

3. **Do not use an identifier for multiple purposes:** Programmers often use the same identifier to denote several temporary entities. For example, some programmers use a temporary loop variable for computing and a storing the final result. The rationale that is usually given by these programmers for such multiple uses of variables is memory efficiency, e.g. three variables use up three memory locations, whereas the same variable used in three different ways uses just one memory location. However, there are several things wrong with this approach and hence should be avoided. Some of the problems caused by use of variables for multiple purposes as follows:
  - Each variable should be given a descriptive name indicating its purpose. This is not possible if an identifier is used for multiple purposes. Use of a variable for multiple purposes can lead to confusion and make it difficult for somebody trying to read and understand the code.
  - Use of variables for multiple purposes usually makes future enhancements more difficult.
4. **The code should be well-documented:** As a rule of thumb, there must be at least one comment line on the average for every three-source line.
5. **The length of any function should not exceed 10 source lines:** A function that is very lengthy is usually very difficult to understand as it probably carries out many different functions. For the same reason, lengthy functions are likely to have disproportionately larger number of bugs.
6. **Do not use goto statements:** Use of goto statements makes a program unstructured and very difficult to understand.

### Code Review

Code review for a model is carried out after the module is successfully compiled and the all the syntax errors have been eliminated. Code reviews are extremely cost-effective strategies for reduction in coding errors and to produce high quality code. Normally, two types of reviews are carried out on the code of a module. These two types of code review techniques are code inspection and code walk through.

### Code Walk Throughs

Code walk through is an informal code analysis technique. In this technique, after a module has been coded, successfully compiled and all syntax errors eliminated. The main objectives of the walk through are to discover the algorithmic and logical errors in the code. The members note down their findings to discuss these in a walk through meeting where the coder of the module is present.

### Code Inspection

In contrast to code walk through, the aim of code inspection is to discover some common types of errors caused due to oversight and improper programming. In other words, during code



inspection the code is examined for the presence of certain kinds of errors, in contrast to the hand simulation of code execution done in code walk throughs. For instance, consider the classical error of writing a procedure that modifies a formal parameter while the calling routine calls that procedure with a constant actual parameter. It is more likely that such an error will be discovered by looking for these kinds of mistakes in the code, rather than by simply hand simulating execution of the procedure. In addition to the commonly made errors, adherence to coding standards is also checked during code inspection. Good software development companies collect statistics regarding different types of errors commonly committed by their engineers and identify the type of errors most frequently committed. Such a list of commonly committed errors can be used during code inspection to look out for possible errors.

Following is a list of some classical programming errors which can be checked during code inspection:

- Use of uninitialized variables.
- Jumps into loops.
- Nonterminating loops.
- Incompatible assignments.
- Array indices out of bounds.
- Improper storage allocation and deallocation.
- Mismatches between actual and formal parameter in procedure calls.
- Use of incorrect logical operators or incorrect precedence among operators.
- Improper modification of loop variables.
- Comparison of equality of floating point variables, etc.

### Clean Room Testing

Clean room testing was pioneered by IBM. This type of testing relies heavily on walk throughs, inspection, and formal verification. The programmers are not allowed to test any of their code by executing the code other than doing some syntax testing using a compiler.

This technique reportedly produces documentation and code that is more reliable and maintainable than other development methods relying heavily on code execution-based testing.

The clean room approach to software development is based on five characteristics:

- **Formal specification:** The software to be developed is formally specified. A state-transition model which shows system responses to stimuli is used to express the specification.
- **Incremental development:** The software is partitioned into increments which are developed and validated separately using the clean room process. These increments are specified, with customer input, at an early stage in the process.
- **Structured programming:** Only a limited number of control and data abstraction constructs are used. The program development process is process of stepwise refinement of the specification.
- **Static verification:** The developed software is statically verified using rigorous software



inspections. There is no unit or module testing process for code components

- **Statistical testing of the system:** The integrated software increment is tested statistically to determine its reliability. These statistical tests are based on the operational profile which is developed in parallel with the system specification. The main problem with this approach is that testing effort is increased as walk throughs, inspection, and verification are time-consuming.

### Software Documentation

When various kinds of software products are developed then not only the executable files and the source code are developed but also various kinds of documents such as users' manual, software requirements specification (SRS) documents, design documents, test documents, installation manual, etc are also developed as part of any software engineering process. All these documents are a vital part of good software development practice. Good documents are very useful and serve the following purposes:

- Good documents enhance understandability and maintainability of a software product. They reduce the effort and time required for maintenance.
- Use documents help the users in effectively using the system.
- Good documents help in effectively handling the manpower turnover problem. Even when an engineer leaves the organization, and a new engineer comes in, he can build up the required knowledge easily.
- Production of good documents helps the manager in effectively tracking the progress of the project. The project manager knows that measurable progress is achieved if a piece of work is done and the required documents have been produced and reviewed.

Different types of software documents can broadly be classified into the following:

- Internal documentation
- External documentation



**Internal documentation** is the code comprehension features provided as part of the source code itself. Internal documentation is provided through appropriate module headers and comments embedded in the source code. Internal documentation is also provided through the useful variable names, module and function headers, code indentation, code structuring, use of enumerated types and constant identifiers, use of user-defined data types, etc. Careful experiments suggest that out of all types of internal documentation meaningful variable names is most useful in understanding the code. This is of course in contrast to the common expectation that code commenting would be the most useful. The research finding is obviously true when comments are written without thought. For example, the following style of code commenting does not in any way help in understanding the code.

```
a = 10; /* a made 10 */
```

But even when code is carefully commented, meaningful variable names still are more helpful in understanding a piece of code. Good software development organizations usually ensure good internal documentation by appropriately formulating their coding standards and coding guidelines.

**External documentation** is provided through various types of supporting documents such as users' manual, software requirements specification document, design document, test documents, etc. A systematic software development style ensures that all these documents are produced in an orderly fashion.

## 4.9. TESTING

### Program Testing

Testing a program consists of providing the program with a set of test inputs (or test cases) and observing if the program behaves as expected. If the program fails to behave as expected, then the conditions under which failure occurs are noted for later debugging and correction.

Some commonly used terms associated with testing are:

- **Failure:** This is a manifestation of an error (or defect or bug). But, the mere presence of an error may not necessarily lead to a failure.
- **Test case:** This is the triplet [I,S,O], where I is the data input to the system, S is the state of the system at which the data is input, and O is the expected output of the system.
- **Test suite:** This is the set of all test cases with which a given software product is to be tested.

### Aim of Testing

The aim of the testing process is to identify all defects existing in a software product. However, for most practical systems, even after satisfactorily carrying out the testing phase, it is not possible to guarantee that the software is error free. This is because of the fact that the input data domain of most software products is very large. It is not practical to test the software



exhaustively with respect to each value that the input data may assume. Even with this practical limitation of the testing process, the importance of testing should not be underestimated. It must be remembered that testing does expose many defects existing in a software product. Thus testing provides a practical way of reducing defects in a system and increasing the users' confidence in a developed system.

### Verification Vs Validation

**Verification** is the process of determining whether the output of one phase of software development conforms to that of its previous phase, whereas **validation** is the process of determining whether a fully developed system conforms to its requirements specification. Thus while verification is concerned with phase containment of errors, the aim of validation is that the final product be error free.

### Design of Test Cases

Exhaustive testing of almost any non-trivial system is impractical due to the fact that the domain of input data values to most practical software systems is either extremely large or infinite. Therefore, we must design an optional test suite that is of reasonable size and can uncover as many errors existing in the system as possible. Actually, if test cases are selected randomly, many of these randomly selected test cases do not contribute to the significance of the test suite, i.e. they do not detect any additional defects not already being detected by other test cases in the suite. Thus, the number of random test cases in a test suite is, in general, not an indication of the effectiveness of the testing. In other words, testing a system using a large collection of test cases that are selected at random does not guarantee that all (or even most) of the errors in the system will be uncovered. Consider the following example code segment which finds the greater of two integer values  $x$  and  $y$ . This code segment has a simple programming error.

```
if (x>y)
    max = x;
else
    max = x;
```

For the above code segment, the test suite,  $\{(x=3,y=2);(x=2,y=3)\}$  can detect the error, whereas a larger test suite  $\{(x=3,y=2);(x=4,y=3);(x=5,y=1)\}$  does not detect the error. So, it would be incorrect to say that a larger test suite would always detect more errors than a smaller one, unless of course the larger test suite has also been carefully designed. This implies that the test suite should be carefully designed than picked randomly. Therefore, systematic approaches should be followed to design an optimal test suite. In an optimal test suite, each test case is designed to detect different errors.

### Functional Testing Vs. Structural Testing

In the black-box testing approach, test cases are designed using only the functional specification of the software, i.e. without any knowledge of the internal structure of the software. For this



reason, black-box testing is known as functional testing. On the other hand, in the white-box testing approach, designing test cases requires thorough knowledge about the internal structure of software, and therefore the white-box testing is called structural testing.

## INTEGRATION TESTING

The primary objective of integration testing is to test the module interfaces, i.e. there are no errors in the parameter passing, when one module invokes another module. During integration testing, different modules of a system are integrated in a planned manner using an integration plan. The integration plan specifies the steps and the order in which modules are combined to realize the full system. After each integration step, the partially integrated system is tested. An important factor that guides the integration plan is the module dependency graph. The structure chart (or module dependency graph) denotes the order in which different modules call each other. By examining the structure chart, the integration plan can be developed.

### Integration test approaches

There are four types of integration testing approaches. Any one (or a mixture) of the following approaches can be used to develop the integration test plan. Those approaches are the following:

- Big bang approach
- Bottom-up approach
- Top-down approach
- Mixed-approach

### Big-Bang Integration Testing

It is the simplest integration testing approach, where all the modules making up a system are integrated in a single step. In simple words, all the modules of the system are simply put together and tested. However, this technique is practicable only for very small systems. The main problem with this approach is that once an error is found during the integration testing, it is very difficult to localize the error as the error may potentially belong to any of the modules being integrated. Therefore, debugging errors reported during big bang integration testing are very expensive to fix.

### Bottom-Up Integration Testing

In bottom-up testing, each subsystem is tested separately and then the full system is tested. A subsystem might consist of many modules which communicate among each other through well-defined interfaces. The primary purpose of testing each subsystem is to test the interfaces among various modules making up the subsystem. Both control and data interfaces are tested. The test cases must be carefully chosen to exercise the interfaces in all possible manners. Large software systems normally require several levels of subsystem testing; lower-level subsystems are successively combined to form higher-level subsystems. A principal advantage of bottom-up integration testing is that several disjoint subsystems can be tested simultaneously. In a pure



bottom-up testing no stubs are required, only test-drivers are required. A disadvantage of bottom-up testing is the complexity that occurs when the system is made up of a large number of small subsystems. The extreme case corresponds to the big-bang approach.

### **Top-Down Integration Testing**

Top-down integration testing starts with the main routine and one or two subordinate routines in the system. After the top-level 'skeleton' has been tested, the immediately subroutines of the 'skeleton' are combined with it and tested. Top-down integration testing approach requires the use of program stubs to simulate the effect of lower-level routines that are called by the routines under test. A pure top-down integration does not require any driver routines. A disadvantage of the top-down integration testing approach is that in the absence of lower-level routines, many times it may become difficult to exercise the top-level routines in the desired manner since the lower-level routines perform several low-level functions such as I/O.

### **Mixed Integration Testing**

A mixed (also called sandwiched) integration testing follows a combination of top-down and bottom-up testing approaches. In top-down approach, testing can start only after the top-level modules have been coded and unit tested. Similarly, bottom-up testing can start only after the bottom level modules are ready. The mixed approach overcomes this shortcoming of the top-down and bottom-up approaches. In the mixed testing approaches, testing can start as and when modules become available. Therefore, this is one of the most commonly used integration testing approaches.

### **Phased Vs. Incremental Testing**

The different integration testing strategies are either phased or incremental. A comparison of these two strategies is as follows:

- In incremental integration testing, only one new module is added to the partial system each time.
- In phased integration, a group of related modules are added to the partial system each time.

Phased integration requires less number of integration steps compared to the incremental integration approach. However, when failures are detected, it is easier to debug the system in the incremental testing approach since it is known that the error is caused by addition of a single module. In fact, big bang testing is a degenerate case of the phased integration testing approach.

#### **System testing**

System tests are designed to validate a fully developed system to assure that it meets its requirements. There are essentially three main kinds of system testing:

- **Alpha Testing.** Alpha testing refers to the system testing carried out by the test team within the developing organization.
- **Beta testing.** Beta testing is the system testing performed by a select group of friendly



customers.

- **Acceptance Testing.** Acceptance testing is the system testing performed by the customer to determine whether he should accept the delivery of the system.

In each of the above types of tests, various kinds of test cases are designed by referring to the SRS document. Broadly, these tests can be classified into functionality and performance tests. The functionality test tests the functionality of the software to check whether it satisfies the functional requirements as documented in the SRS document. The performance test tests the conformance of the system with the nonfunctional requirements of the system.

### Performance Testing

Performance testing is carried out to check whether the system needs the non-functional requirements identified in the SRS document. There are several types of performance testing. Among of them nine types are discussed below. The types of performance testing to be carried out on a system depend on the different non-functional requirements of the system documented in the SRS document. All performance tests can be considered as black-box tests.

- Stress testing
- Volume testing
- Configuration testing
- Compatibility testing
- Regression testing
- Recovery testing
- Maintenance testing
- Documentation testing
- Usability testing

**Stress Testing** -Stress testing is also known as *endurance testing*. Stress testing evaluates system performance when it is stressed for short periods of time. Stress tests are black box tests which are designed to impose a range of abnormal and even illegal input conditions so as to stress the capabilities of the software. Input data volume, input data rate, processing time, utilization of memory, etc. are tested beyond the designed capacity. For example, suppose an operating system is supposed to support 15 multi programmed jobs, the system is stressed by attempting to run 15 or more jobs simultaneously. A real- time system might be tested to determine the effect of simultaneous arrival of several high-priority interrupts.

Stress testing is especially important for systems that usually operate below the maximum capacity but are severely stressed at some peak demand hours. For example, if the non- functional requirement specification states that the response time should not be more than 20 secs per transaction when 60 concurrent users are working, then during the stress testing the response time is checked with 60 users working simultaneously.

**Volume Testing**-It is especially important to check whether the data structures (arrays, queues, stacks, etc.) have been designed to successfully extraordinary situations. For example, a compiler



might be tested to check whether the symbol table overflows when a very large program is compiled.

**Configuration Testing** - This is used to analyze system behavior in various hardware and software configurations specified in the requirements. Sometimes systems are built in variable configurations for different users. For instance, we might define a minimal system to serve a single user, and other extension configurations to serve additional users. The system is configured in each of the required configurations and it is checked if the system behaves correctly in all required configurations.

**Compatibility Testing** - This type of testing is required when the system interfaces with other types of systems. Compatibility aims to check whether the interface functions perform as required. For instance, if the system needs to communicate with a large database system to retrieve information, compatibility testing is required to test the speed and accuracy of data retrieval.

**Regression Testing** - This type of testing is required when the system being tested is an upgradation of an already existing system to fix some bugs or enhance functionality, performance, etc. Regression testing is the practice of running an old test suite after each change to the system or after each bug fix to ensure that no new bug has been introduced due to the change or the bug fix. However, if only a few statements are changed, then the entire test suite need not be run - only those test cases that test the functions that are likely to be affected by the change need to be run.

**Recovery Testing** - Recovery testing tests the response of the system to the presence of faults, or loss of power, devices, services, data, etc. The system is subjected to the loss of the mentioned resources (as applicable and discussed in the SRS document) and it is checked if the system recovers satisfactorily. For example, the printer can be disconnected to check if the system hangs. Or, the power may be shut down to check the extent of data loss and corruption.

**Maintenance Testing** - This testing addresses the diagnostic programs, and other procedures that are required to be developed to help maintenance of the system. It is verified that the artifacts exist and they perform properly.

**Documentation Testing** - It is checked that the required user manual, maintenance manuals, and technical manuals exist and are consistent. If the requirements specify the types of audience for which a specific manual should be designed, then the manual is checked for compliance.

**Usability Testing** - Usability testing concerns checking the user interface to see if it meets all user requirements concerning the user interface. During usability testing, the display screens, report formats, and other aspects relating to the user interface requirements are tested.

### **Error Seeding**

Sometimes the customer might specify the maximum number of allowable errors that may be present in the delivered system. These are often expressed in terms of maximum number of allowable errors per line of source code. Error seed can be used to estimate the number of residual errors in a system. Error seeding, as the name implies, seeds the code with some known errors. In other words, some artificial errors are introduced into the program artificially. The number of these seeded errors



detected in the course of the standard testing procedure is determined. These values in conjunction with the number of unseeded errors detected can be used to predict:

- The number of errors remaining in the product.
- The effectiveness of the testing strategy.

Error seeding works satisfactorily only if the kind of seeded errors matches closely with the kind of defects that actually exist. However, it is difficult to predict the types of errors that exist in a software. To some extent, the different categories of errors that remain can be estimated to a first approximation by analyzing historical data of similar projects. Due to the shortcoming that the types of seeded errors should match closely with the types of errors actually existing in the code, error seeding is useful only to a moderate extent.

#### 4.10. SOFTWARE MAINTENANCE

##### Necessity of Software Maintenance

Software maintenance is becoming an important activity of a large number of software organizations. This is no surprise, given the rate of hardware obsolescence, the immortality of a software product per se, and the demand of the user community to see the existing software products run on newer platforms, run in newer environments, and/or with enhanced features. When the hardware platform is changed, and a software product performs some low-level functions, maintenance is necessary. Also, whenever the support environment of a software product changes, the software product requires rework to cope up with the newer interface. For instance, a software product may need to be maintained when the operating system changes. Thus, every software product continues to evolve after its development through maintenance efforts. Therefore, it can be stated that software maintenance is needed to correct errors, enhance features, port the software to new platforms, etc.

##### Types of software maintenance

There are basically three types of software maintenance. These are:

- **Corrective:** Corrective maintenance of a software product is necessary to rectify the bugs observed while the system is in use.
- **Adaptive:** A software product might need maintenance when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware or software.
- **Perfective:** A software product needs maintenance to support the new features that users want it to support, to change different functionalities of the system according to customer demands, or to enhance the performance of the system.

##### Problems associated with software maintenance



Software maintenance work typically is much more expensive than what it should be and takes more time than required. In software organizations, maintenance work is mostly carried out using ad hoc techniques. The primary reason being that software maintenance is one of the most neglected areas of software engineering. Even though software maintenance is fast becoming an important area of work for many companies as the software products of yester years age, still software maintenance is mostly being carried out as fire-fighting operations, rather than through systematic and planned activities.

Software maintenance has a very poor image in industry. Therefore, an organization often cannot employ bright engineers to carry out maintenance work. Even though maintenance suffers from a poor image, the work involved is often more challenging than development work. During maintenance it is necessary to thoroughly understand someone else's work and then carry out the required modifications and extensions.

Another problem associated with maintenance work is that the majority of software products needing maintenance are legacy products.

### Software Reverse Engineering

Software reverse engineering is the process of recovering the design and the requirements specification of a product from an analysis of its code. The purpose of reverse engineering is to facilitate maintenance work by improving the understandability of a system and to produce the necessary documents for a legacy system. Reverse engineering is becoming important, since legacy software products lack proper documentation, and are highly unstructured. Even well-designed products become legacy software as their structure degrades through a series of maintenance efforts.

The first stage of reverse engineering usually focuses on carrying out cosmetic changes to the code to improve its readability, structure, and understandability, without changing of its functionalities. A process model for reverse engineering has been shown in fig. 24.1. A program can be reformatted using any of the several available prettyprinter programs which layout the program neatly. Many legacy software products with complex control structure and unthoughtful variable names are difficult to comprehend. Assigning meaningful variable names is important because meaningful variable names are the most helpful thing in code documentation. All variables, data structures, and functions should be assigned meaningful names wherever possible. Complex nested conditionals in the program can be replaced by simpler conditional statements or whenever appropriate by case statements.

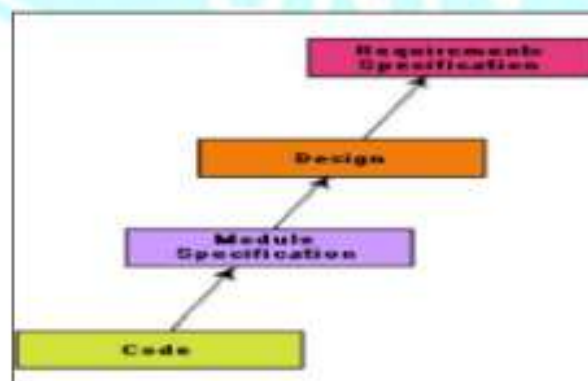




Fig. 24.1: A process model for reverse engineering

After the cosmetic changes have been carried out on a legacy software the process of extracting the code, design, and the requirements specification can begin. These activities are schematically shown in fig. 24.2. In order to extract the design, a full understanding of the code is needed. Some automatic tools can be used to derive the data flow and control flow diagram from the code. The structure chart (module invocation sequence and data interchange among modules) should also be extracted. The SRS document can be written once the full code has been thoroughly understood and the design extracted.

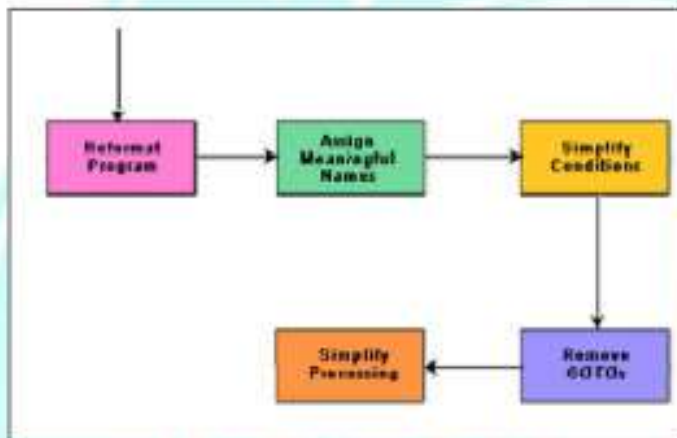


Fig. 24.2: Cosmetic changes carried out before reverse engineering

### Legacy software products

It is prudent to define a legacy system as any software system that is hard to maintain. The typical problems associated with legacy systems are poor documentation, unstructured (spaghetti code with ugly control structure), and lack of personnel knowledgeable in the product. Many of the legacy systems were developed long time back. But, it is possible that a recently developed system having poor design and documentation can be considered to be a legacy system.

The activities involved in a software maintenance project are not unique and depend on several factors such as:

- The extent of modification to the product required
- The resources available to the maintenance team
- The conditions of the existing product (e.g., how structured it is, how well documented it is, etc.)
- The expected project risks, etc.

When the changes needed to a software product are minor and straightforward, the code can be directly modified and the changes appropriately reflected in all the documents. But more elaborate activities are required when the required changes are not so trivial. Usually, for complex maintenance projects for legacy systems, the software process can be represented by a



reverse engineering cycle followed by a forward engineering cycle with an emphasis on as much reuse as possible from the existing code and other documents.

## SOFTWARE MAINTENANCE PROCESS MODELS

Two broad categories of process models for software maintenance can be proposed. The first model is preferred for projects involving small reworks where the code is changed directly and the changes are reflected in the relevant documents later. This maintenance process is graphically presented in fig. 25.1. In this approach, the project starts by gathering the requirements for changes. The requirements are next analyzed to formulate the strategies to be adopted for code change. At this stage, the association of at least a few members of the original development team goes a long way in reducing the cycle time, especially for projects involving unstructured and inadequately documented code. The availability of a working old system to the maintenance engineers at the maintenance site greatly facilitates the task of the maintenance team as they get a good insight into the working of the old system and also can compare the working of their modified system with the old system. Also, debugging of the reengineered system becomes easier as the program traces of both the systems can be compared to localize the bugs.

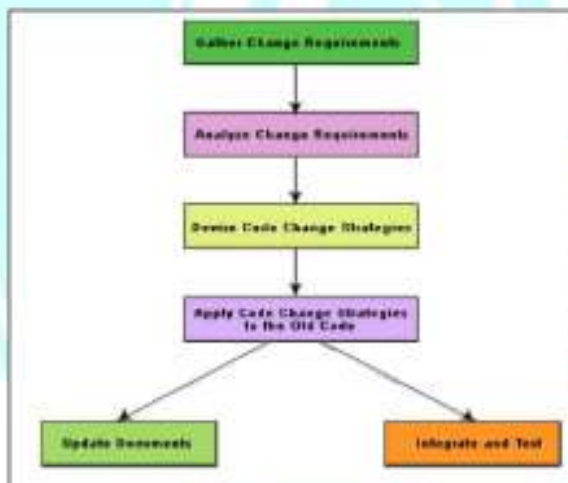


Fig. 25.1: Maintenance process model 1

The second process model for software maintenance is preferred for projects where the amount of rework required is significant. This approach can be represented by a reverse engineering cycle followed by a forward engineering cycle. Such an approach is also known as software reengineering. This process model is depicted in fig. 25.2. The reverse engineering cycle is required for legacy products. During the reverse engineering, the old code is analyzed (abstracted) to extract the module specifications. The module specifications are then analyzed to produce the design. The design is analyzed (abstracted) to produce the original requirements specification. The change requests are then applied to this requirements specification to arrive at the new requirements specification. At the design, module specification, and coding a substantial reuse is made from the reverse engineered products. An important advantage of this approach is that it produces a more structured design



compared to what the original product had, produces good documentation, and very often results in increased efficiency. The efficiency improvements are brought about by a more efficient design. However, this approach is costlier than the first approach. An empirical study indicates that process 1 is preferable when the amount of rework is no more than 15%. Besides the amount of rework, several other factors might affect the decision regarding using process model 1 over process model 2:

- Reengineering might be preferable for products which exhibit a high failure rate.
- Reengineering might also be preferable for legacy products having poor design and code structure.

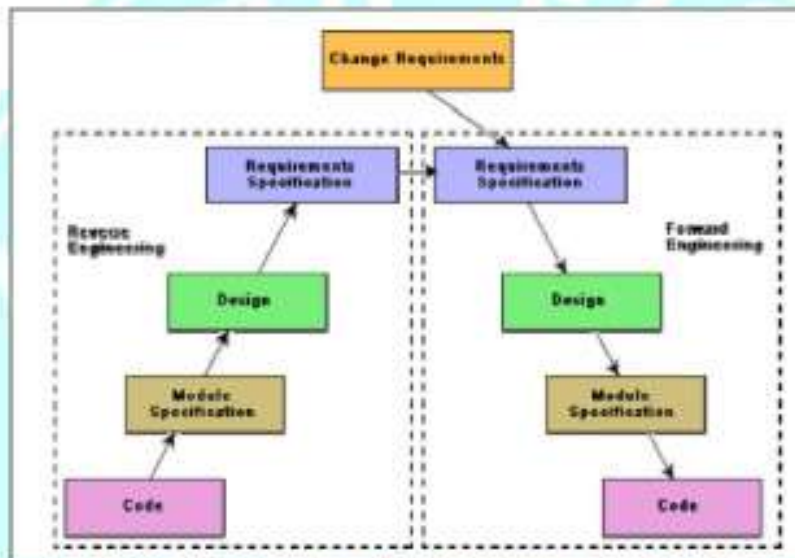


Fig. 25.2: Maintenance process model 2

## Software Reengineering

Software reengineering is a combination of two consecutive processes i.e. software reverse engineering and software forward engineering as shown in the fig. 25.2.

## Estimation of approximate maintenance cost

It is well known that maintenance efforts require about 60% of the total life cycle cost for a typical software product. However, maintenance costs vary widely from one application domain to another. For embedded systems, the maintenance cost can be as much as 2 to 4 times the development cost.

Boehm [1981] proposed a formula for estimating maintenance costs as part of his COCOMO cost estimation model. Boehm's maintenance cost estimation is made in terms of a quantity called the Annual Change Traffic (ACT). Boehm defined ACT as the fraction of a software product's source instructions which undergo change during a typical year either through addition or deletion.

$$ACT = \frac{KLOC_{added}}{KLOC_{total}} + \frac{KLOC_{deleted}}{KLOC_{total}}$$



$$KLOC_{total}$$

where,  $KLOC_{added}$  is the total kilo lines of source code added during maintenance.

$KLOC_{deleted}$  is the total kilo lines of source code deleted during maintenance.

Thus, the code that is changed, should be counted in both the code added and the code deleted. The annual change traffic (ACT) is multiplied with the total development cost to arrive at the maintenance cost:

$$\text{Maintenance cost} = ACT \times \text{development cost.}$$

Most maintenance cost estimation models, however, yield only approximate results because they do not take into account several factors such as experience level of the engineers, and familiarity of the engineers with the product, hardware requirements, software complexity, etc.

#### 4.11. SOFTWARE QUALITY

Traditionally, a quality product is defined in terms of its fitness of purpose. That is, a quality product does exactly what the users want it to do. For software products, fitness of purpose is usually interpreted in terms of satisfaction of the requirements laid down in the SRS document. Although “fitness of purpose” is a satisfactory definition of quality for many products such as a car, a table fan, a grinding machine, etc. – for software products, “fitness of purpose” is not a wholly satisfactory definition of quality. To give an example, consider a software product that is functionally correct. That is, it performs all functions as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally correct, we cannot consider it to be a quality product. Another example may be that of a product which does everything that the users want but has an almost incomprehensible and unmaintainable code. Therefore, the traditional concept of quality as “fitness of purpose” for software products is not wholly satisfactory.

The modern view of a quality associates with a software product several quality factors such as the following:

- **Portability:** A software product is said to be portable, if it can be easily made to work in different operating system environments, in different machines, with other software products, etc.
- **Usability:** A software product has good usability, if different categories of users (i.e. both expert and novice users) can easily invoke the functions of the product.
- **Reusability:** A software product has good reusability, if different modules of the product can easily be reused to develop new products.
- **Correctness:** A software product is correct, if different requirements as specified in the SRS document have been correctly implemented.
- **Maintainability:** A software product is maintainable, if errors can be easily corrected as and when they show up, new functions can be easily added to the product, and the functionalities of the product can be easily modified, etc.



A quality management system (often referred to as quality system) is the principal methodology used by organizations to ensure that the products they develop have the desired quality.

A quality system consists of the following:

**Managerial Structure and Individual Responsibilities-** A quality system is actually the responsibility of the organization as a whole. However, every organization has a separate quality department to perform several quality system activities. The quality system of an organization should have support of the top management. Without support for the quality system at a high level in a company, few members of staff will take the quality system seriously.

**Quality System Activities-** The quality system activities encompass the following:

- Auditing of projects
- Review of the quality system
- Development of standards, procedures, and guidelines, etc.
- Production of reports for the top management summarizing the effectiveness of the quality system in the organization.

### Evolution of Quality Management System

Quality systems have rapidly evolved over the last five decades. Prior to World War II, the usual method to produce quality products was to inspect the finished products to eliminate defective products. Since that time, quality systems of organizations have undergone through four stages of evolution as shown in the fig. 28.1. The initial product inspection method gave way to quality control (QC). Quality control focuses not only on detecting the defective products and eliminating them but also on determining the causes behind the defects. Thus, quality control aims at correcting the causes of errors and not just rejecting the products. The next breakthrough in quality systems was the development of quality assurance principles.

The basic premise of modern quality assurance is that if an organization's processes are good and are followed rigorously, then the products are bound to be of good quality. The modern quality paradigm includes guidance for recognizing, defining, analyzing, and improving the production process. Total quality management (TQM) advocates that the process followed by an organization must be continuously improved through process measurements. TQM goes a step further than quality assurance and aims at continuous process improvement. TQM goes beyond documenting processes to optimizing them through redesign. A term related to TQM is Business Process Reengineering (BPR). BPR aims at reengineering the way business is carried out in an organization. From the above discussion it can be stated that over the years the quality paradigm has shifted from product assurance to process assurance (as shown in fig. 28.1).



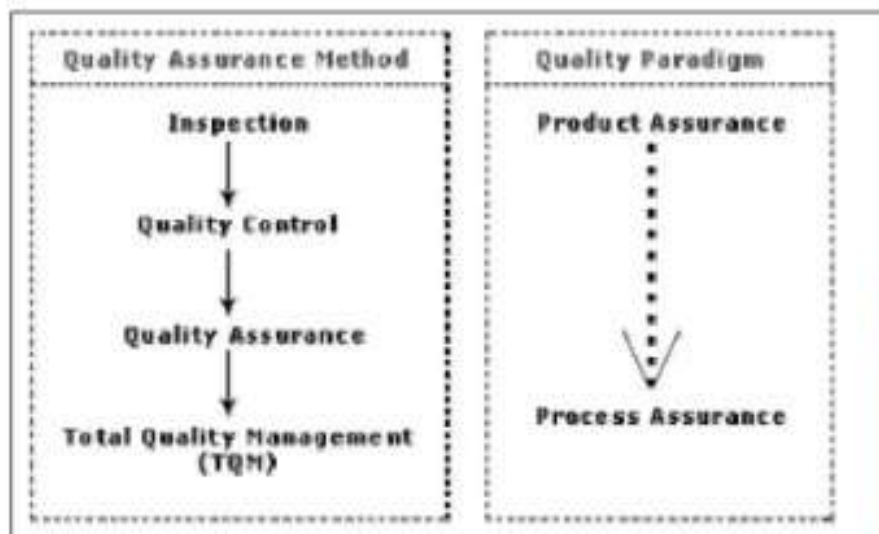


Fig. 28.1: Evolution of quality system and corresponding shift in the quality paradigm

## 4.12. SOFTWARE PROJECT PLANNING

### Project Planning and Project Estimation Techniques

#### Responsibilities of a software project manager

Software project managers take the overall responsibility of steering a project to success. It is very difficult to objectively describe the job responsibilities of a project manager. The job responsibility of a project manager ranges from invisible activities like building up team morale to highly visible customer presentations. Most managers take responsibility for project proposal writing, project cost estimation, scheduling, project staffing, software process tailoring, project monitoring and control, software configuration management, risk management, interfacing with clients, managerial report writing and presentations, etc. These activities are certainly numerous, varied and difficult to enumerate, but these activities can be broadly classified into project planning, and project monitoring and control activities. The project planning activity is undertaken before the development starts to plan the activities to be undertaken during development. The project monitoring and control activities are undertaken once the development activities start with the aim of ensuring that the development proceeds as per plan and changing the plan whenever required to cope up with the situation.

#### Skills necessary for software project management

A theoretical knowledge of different project management techniques is certainly necessary to become a successful project manager. However, effective software project management frequently calls for good qualitative judgment and decision taking capabilities. In addition to having a good grasp of the latest software project management techniques such as cost estimation, risk management, configuration management, project managers need good



communication skills and the ability get work done. However, some skills such as tracking and controlling the progress of the project, customer interaction, managerial presentations, and team building are largely acquired through experience. None the less, the importance of sound knowledge of the prevalent project management techniques cannot be overemphasized.

## Project Planning

Once a project is found to be feasible, software project managers undertake project planning. Project planning is undertaken and completed even before any development activity starts. Project planning consists of the following essential activities:

- Estimating the following attributes of the project:
  - **Project size:** What will be problem complexity in terms of the effort and time required to develop the product?
  - **Cost:** How much is it going to cost to develop the project?
  - **Duration:** How long is it going to take to complete development?
  - **Effort:** How much effort would be required?

The effectiveness of the subsequent planning activities is based on the accuracy of these estimations.

- Scheduling manpower and other resources.
- Staff organization and staffing plans.
- Risk identification, analysis, and abatement planning
- Miscellaneous plans such as quality assurance plan, configuration management plan, etc.

### Precedence ordering among project planning activities

Different project related estimates done by a project manager have already been discussed. Fig. 30.1 Shows the order in which important project planning activities may be undertaken. From fig. 30.1 it can be easily observed that size estimation is the first activity. It is also the most fundamental parameter based on which all other planning activities are carried out. Other estimations such as estimation of effort, cost, resource, and project duration are also very important components of project planning.

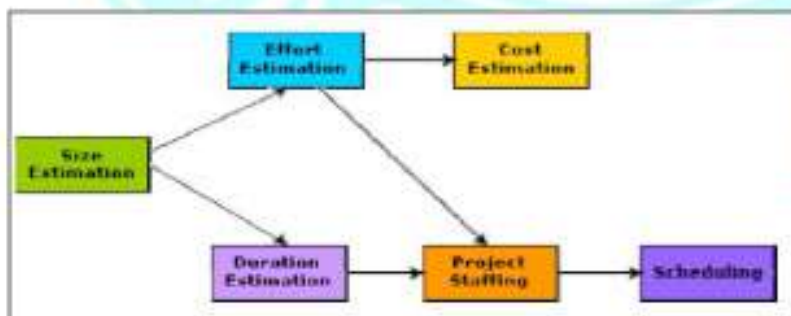


Fig. 30.1: Precedence ordering among planning activities



## Sliding Window Planning

Project planning requires utmost care and attention since commitment to unrealistic time and resource estimates result in schedule slippage. Schedule delays can cause customer dissatisfaction and adversely affect team morale. It can even cause project failure. However, project planning is a very challenging activity. Especially for large projects, it is very much difficult to make accurate plans. A part of this difficulty is due to the fact that the proper parameters, scope of the project, project staff, etc. may change during the span of the project. In order to overcome this problem, sometimes project managers undertake project planning in stages. Planning a project over a number of stages protects managers from making big commitments too early. This technique of staggered planning is known as Sliding Window Planning. In the sliding window technique, starting with an initial plan, the project is planned more accurately in successive development stages. At the start of a project, project managers have incomplete knowledge about the details of the project. Their information base gradually improves as the project progresses through different phases. After the completion of every phase, the project managers can plan each subsequent phase more accurately and with increasing levels of confidence.

## Software Project Management Plan (SPMP)

Once project planning is complete, project managers document their plans in a Software Project Management Plan (SPMP) document. The SPMP document should discuss a list of different items that have been discussed below. This list can be used as a possible organization of the SPMP document.

### Organization of the Software Project Management Plan (SPMP) Document

1. Introduction
  - (a) Objectives
  - (b) Major Functions
  - (c) Performance Issues
  - (d) Management and Technical Constraints
2. Project Estimates
  - (e) Historical Data Used
  - (f) Estimation Techniques Used
  - (g) Effort, Resource, Cost, and Project Duration Estimates
3. Schedule
  - (h) Work Breakdown Structure
  - (i) Task Network Representation
  - (j) Gantt Chart Representation
  - (k) PERT Chart Representation
4. Project Resources
  - (l) People
  - (m) Hardware and Software
  - (n) Special Resources



5. Staff Organization
  - (o) Team Structure
  - (p) Management Reporting
6. Risk Management Plan
  - (q) Risk Analysis
  - (r) Risk Identification
  - (s) Risk Estimation
  - (t) Risk Abatement Procedures
7. Project Tracking and Control Plan
8. Miscellaneous Plans
  - (u) Process Tailoring
  - (v) Quality Assurance Plan
  - (w) Configuration Management Plan
  - (x) Validation and Verification
  - (y) System Testing Plan
  - (z) Delivery, Installation, and Maintenance Plan

## METRICS FOR SOFTWARE PROJECT SIZE ESTIMATION

Accurate estimation of the problem size is fundamental to satisfactory estimation of effort, time duration and cost of a software project. In order to be able to accurately estimate the project size, some important metrics should be defined in terms of which the project size can be expressed. The size of a problem is obviously not the number of bytes that the source code occupies. It is neither the byte size of the executable code. The project size is a measure of the problem complexity in terms of the effort and time required to develop the product.

Currently two metrics are popularly being used widely to estimate size: lines of code (LOC) and function point (FP). The usage of each of these metrics in project size estimation has its own advantages and disadvantages.

### Lines of Code (LOC)

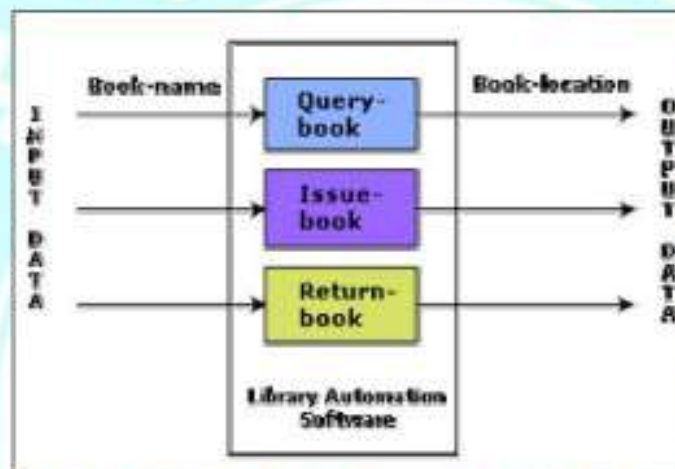
LOC is the simplest among all metrics available to estimate project size. This metric is very popular because it is the simplest to use. Using this metric, the project size is estimated by counting the number of source instructions in the developed program. Obviously, while counting the number of source instructions, lines used for commenting the code and the header lines should be ignored.

Determining the LOC count at the end of a project is a very simple job. However, accurate estimation of the LOC count at the beginning of a project is very difficult. In order to estimate the LOC count at the beginning of a project, project managers usually divide the problem into modules, and each module into submodules and so on, until the sizes of the different leaf-level modules can be approximately predicted. To be able to do this, past experience in developing similar products is helpful. By using the estimation of the lowest level modules, project managers arrive at the total size estimation.

### Function point (FP)



Function point metric was proposed by Albrecht [1983]. This metric overcomes many of the shortcomings of the LOC metric. Since its inception in late 1970s, function point metric has been slowly gaining popularity. One of the important advantages of using the function point metric is that it can be used to easily estimate the size of a software product directly from the problem specification. This is in contrast to the LOC metric, where the size can be accurately determined only after the product has fully been developed. The conceptual idea behind the function point metric is that the size of a software product is directly dependent on the number of different functions or features it supports. A software product supporting many features would certainly be of larger size than a product with less number of features. Each function when invoked reads some input data and transforms it to the corresponding output data.



## PROJECT SCHEDULING

Project-task scheduling is an important project planning activity. It involves deciding which tasks would be taken up when. In order to schedule the project activities, a software project manager needs to do the following:

1. Identify all the tasks needed to complete the project.
2. Break down large tasks into small activities.
3. Determine the dependency among different activities.
4. Establish the most likely estimates for the time durations necessary to complete the activities.
5. Allocate resources to activities.
6. Plan the starting and ending dates for various activities.
7. Determine the critical path. A critical path is the chain of activities that determines the duration of the project.

The first step in scheduling a software project involves identifying all the tasks necessary to complete the project. A good knowledge of the intricacies of the project and the development process helps the managers to effectively identify the important tasks of the project. Next, the large tasks are broken down into a logical set of small activities which would be assigned to different engineers. The work breakdown structure formalism helps the manager to breakdown



the tasks systematically after the project manager has broken down the tasks and created the work breakdown structure, he has to find the dependency among the activities. Dependency among the different activities determines the order in which the different activities would be carried out. If an activity A requires the results of another activity B, then activity A must be scheduled after activity B. In general, the task dependencies define a partial ordering among tasks, i.e. each task may precede a subset of other tasks, but some tasks might not have any precedence ordering defined between them (called concurrent task). The dependency among the activities is represented in the form of an activity network.

Once the activity network representation has been worked out, resources are allocated to each activity. Resource allocation is typically done using a Gantt chart. After resource allocation is done, a PERT chart representation is developed. The PERT chart representation is suitable for program monitoring and control. For task scheduling, the project manager needs to decompose the project tasks into a set of activities. The time frame when each activity is to be performed is to be determined. The end of each activity is called milestone. The project manager tracks the progress of a project by monitoring the timely completion of the milestones. If he observes that the milestones start getting delayed, then he has to carefully control the activities, so that the overall deadline can still be met.

### **Work Breakdown Structure**

Work Breakdown Structure (WBS) is used to decompose a given task set recursively into small activities. WBS provides a notation for representing the major tasks need to be carried out in order to solve a problem. The root of the tree is labeled by the problem name. Each node of the tree is broken down into smaller activities that are made the children of the node. Each activity is recursively decomposed into smaller sub-activities until at the leaf level, the activities requires approximately two weeks to develop. Fig. 36.1 represents the WBS of a MIS (Management Information System) software.

While breaking down a task into smaller tasks, the manager has to make some hard decisions. If a task is broken down into large number of very small activities, these can be carried out independently. Thus, it becomes possible to develop the product faster (with the help of additional manpower). Therefore, to be able to complete a project in the least amount of time, the manager needs to break large tasks into smaller ones, expecting to find more parallelism. However, it is not useful to subdivide tasks into units which take less than a week or two to execute. Very fine subdivision means that a disproportionate amount of time must be spent on preparing and revising various charts.



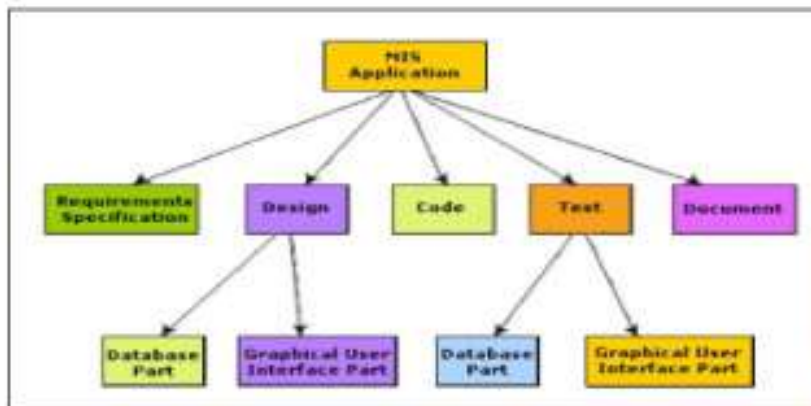


Fig. 36.1: Work breakdown structure of an MIS problem

Activity networks and critical path method WBS representation of a project is transformed into an activity network by representing activities identified in WBS along with their interdependencies. An activity network shows the different activities making up a project, their estimated durations, and interdependencies (as shown in fig. 36.2). Each activity is represented by a rectangular node and the duration of the activity is shown alongside each task.

Managers can estimate the time durations for the different tasks in several ways. One possibility is that they can empirically assign durations to different tasks. This however is not a good idea, because software engineers often resent such unilateral decisions. A possible alternative is to let engineer himself estimate the time for an activity he can assigned to. However, some managers prefer to estimate the time for various activities themselves. Many managers believe that an aggressive schedule motivates the engineers to do a better and faster job. However, careful experiments have shown that unrealistically aggressive schedules not only cause engineers to compromise on intangible quality aspects, but also are a cause for schedule delays. A good way to achieve accurately in estimation of the task durations without creating undue schedule pressures is to have people set their own schedules.

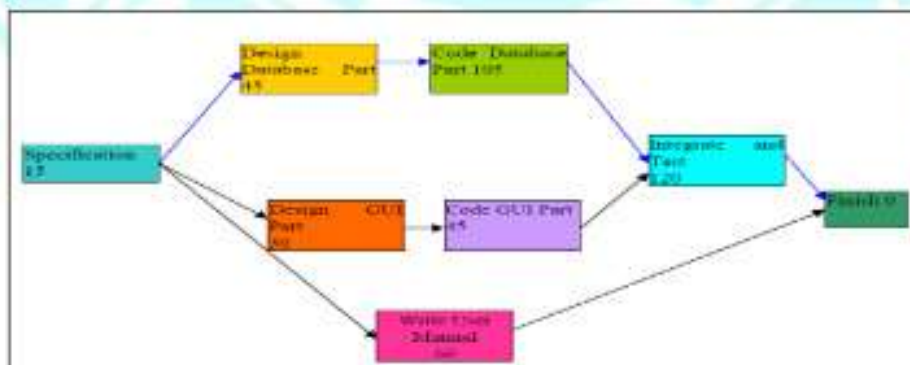


Fig. 36.2: Activity network representation of the MIS problem



## Critical Path Method (CPM)

From the activity network representation following analysis can be made. The minimum time (MT) to complete the project is the maximum of all paths from start to finish. The earliest start (ES) time of a task is the maximum of all paths from the start to the task. The latest start time is the difference between MT and the maximum of all paths from this task to the finish. The earliest finish time (EF) of a task is the sum of the earliest start time of the task and the duration of the task. The latest finish (LF) time of a task can be obtained by subtracting maximum of all paths from this task to finish from MT. The slack time (ST) is  $LS - EF$  and equivalently can be written as  $LF - EF$ . The slack time (or float time) is the total time that a task may be delayed before it will affect the end time of the project. The slack time indicates the “flexibility” in starting and completion of tasks. A critical task is one with a zero slack time. A path from the start node to the finish node containing only critical tasks is called a critical path. These parameters for different tasks for the MIS problem are shown in the following table.

Task	ES	EF	LS	LF	ST
Specification	0	15	0	15	0
Design database	15	60	15	60	0
Design GUI part	15	45	90	120	75
Code database	60	165	60	165	0
Code GUI part	45	90	120	165	75
Integrate and test	165	285	165	285	0
Write user manual	15	75	225	285	210

The critical paths are all the paths whose duration equals MT. The critical path in fig. 36.2 is shown with a blue arrow.

## Gantt chart

Gantt charts are mainly used to allocate resources to activities. The resources allocated to activities include staff, hardware, and software. Gantt charts (named after its developer Henry Gantt) are useful for resource planning. A Gantt chart is a special type of bar chart where each bar represents an activity. The bars are drawn along a time line. The length of each bar is proportional to the duration of time planned for the corresponding activity.

Gantt charts are used in software project management are actually an enhanced version of the standard Gantt charts. In the Gantt charts used for software project management, each bar consists of a white part and a shaded part. The shaded part of the bar shows the length of time each task is estimated to take. The white part shows the slack time, that is, the latest time by which a task must be finished. A Gantt chart representation for the MIS problem of fig. 36.2 is shown in the fig. 36.3.



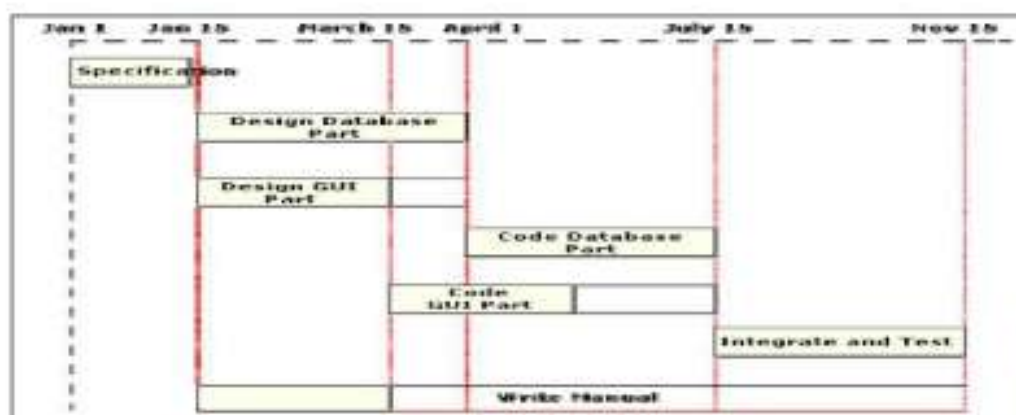


Fig. 36.3: Gantt chart representation of the MIS problem

## PERT Chart

PERT (Project Evaluation and Review Technique) charts consist of a network of boxes and arrows. The boxes represent activities and the arrows represent task dependencies. PERT chart represents the statistical variations in the project estimates assuming a normal distribution. Thus, in a PERT chart instead of making a single estimate for each task, pessimistic, likely, and optimistic estimates are made. The boxes of PERT charts are usually annotated with the pessimistic, likely, and optimistic estimates for every task. Since all possible completion times between the minimum and maximum duration for every task has to be considered, there are not one but many critical paths, depending on the permutations of the estimates for each task. This makes critical path analysis in PERT charts very complex. A critical path in a PERT chart is shown by using thicker arrows. The PERT chart representation of the MIS problem of fig. 36.2 is shown in fig. 36.4. PERT charts are a more sophisticated form of activity chart. In activity diagrams only the estimated task durations are represented. Since, the actual durations might vary from the estimated durations, the utility of the activity diagrams are limited.

Gantt chart representation of a project schedule is helpful in planning the utilization of resources, while PERT chart is useful for monitoring the timely progress of activities. Also, it is easier to identify parallel activities in a project using a PERT chart. Project managers need to identify the parallel activities in a project for assignment to different engineers.



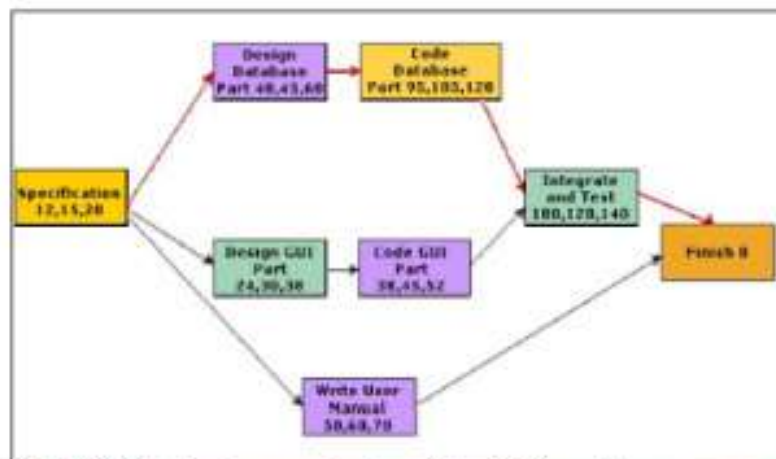


Fig. 36.4: PERT chart representation of the MIS problem

### 4.13. RISK MANAGEMENT

A software project can be affected by a large variety of risks. In order to be able to systematically identify the important risks which might affect a software project, it is necessary to categorize risks into different classes. The project manager can then examine which risks from each class are relevant to the project.

There are three main categories of risks which can affect a software project:

#### 1. Project risks

Project risks concern various forms of budgetary, schedule, personnel, resource, and customer-related problems. An important project risk is schedule slippage. Since, software is intangible, it is very difficult to monitor and control a software project. It is very difficult to control something which cannot be seen. For any manufacturing project, such as manufacturing of cars, the project manager can see the product taking shape. He can, for instance, see that the engine is fitted, after that the doors are fitted, the car is getting painted, etc. Thus he can easily assess the progress of the work and control it. The invisibility of the product being developed is an important reason why many software projects suffer from the risk of schedule slippage.

#### 2. Technical risks

Technical risks concern potential design, implementation, interfacing, testing, and maintenance problems. Technical risks also include ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence. Most technical risks occur due to the development team's insufficient knowledge about the project.

#### 3. Business risks

This type of risks includes risks of building an excellent product that no one wants, losing budgetary or personnel commitments, etc.



The objective of risk assessment is to rank the risks in terms of their damage causing potential. For risk assessment, first each risk should be rated in two ways:

- The likelihood of a risk coming true (denoted as  $r$ ).
- The consequence of the problems associated with that risk (denoted as  $s$ ).

Based on these two factors, the priority of each risk can be computed:

$$p = r * s$$

Where,  $p$  is the priority with which the risk must be handled,  $r$  is the probability of the risk becoming true, and  $s$  is the severity of damage caused due to the risk becoming true. If all identified risks are prioritized, then the most likely and damaging risks can be handled first and more comprehensive risk abatement procedures can be designed for these risks.

### Risk Containment

After all the identified risks of a project are assessed, plans must be made to contain the most damaging and the most likely risks. Different risks require different containment procedures. In fact, most risks require ingenuity on the part of the project manager in tackling the risk.

There are three main strategies to plan for risk containment:

**Avoid the risk-** This may take several forms such as discussing with the customer to change the requirements to reduce the scope of the work, giving incentives to the engineers to avoid the risk of manpower turnover, etc.

**Transfer the risk-** This strategy involves getting the risky component developed by a third party, buying insurance cover, etc.

**Risk reduction-** This involves planning ways to contain the damage due to a risk. For example, if there is risk that some key personnel might leave, new recruitment may be planned.

### Risk Leverage

To choose between the different strategies of handling a risk, the project manager must consider the cost of handling the risk and the corresponding reduction of risk. For this the risk leverage of the different risks can be computed.

Risk leverage is the difference in risk exposure divided by the cost of reducing the risk. More formally,

$$\text{risk leverage} = (\text{risk exposure before reduction} - \text{risk exposure after reduction}) / (\text{cost of reduction})$$

### Risk related to schedule slippage

Even though there are three broad ways to handle any risk, but still risk handling requires a lot of ingenuity on the part of a project manager. As an example, it can be considered the options available to contain an important type of risk that occurs in many software projects – that of schedule slippage. Risks relating to schedule slippage arise primarily due to the intangible nature



of software. Therefore, these can be dealt with by increasing the visibility of the software product. Visibility of a software product can be increased by producing relevant documents during the development process wherever meaningful and getting these documents reviewed by an appropriate team. Milestones should be placed at regular intervals through a software engineering process to provide a manager with regular indication of progress. Completion of a phase of the development process before followed need not be the only milestones. Every phase can be broken down to reasonable-sized tasks and milestones can be scheduled for these tasks too. A milestone is reached, once documentation produced as part of a software engineering task is produced and gets successfully reviewed. Milestones need not be placed for every activity. An approximate rule of thumb is to set a milestone every 10 to 15 days.

### Software Configuration Management

The results (also called as the deliverables) of a large software development effort typically consist of a large number of objects, e.g. source code, design document, SRS document, test document, user's manual, etc. These objects are usually referred to and modified by a number of software engineers throughout the life cycle of the software. The state of all these objects at any point of time is called the configuration of the software product. The state of each deliverable object changes as development progresses and also as bugs are detected and fixed.

### Release vs. Version vs. Revision

A new version of a software is created when there is a significant change in functionality, technology, or the hardware it runs on, etc. On the other hand, a new revision of a software refers to minor bug fix in that software. A new release is created if there is only a bug fix, minor enhancements to the functionality, usability, etc.

For example, one version of a mathematical computation package might run on Unix-based machines, another on Microsoft Windows and so on. As a software is released and used by the customer, errors are discovered that need correction. Enhancements to the functionalities of the software may also be needed. A new release of software is an improved system intended to replace an old one. Often systems are described as version m, release n; or simple m.n. formally, a history relation is version of can be defined between objects. This relation can be split into two sub relations *is revision of* and *is variant of*.

### Necessity of software configuration management

There are several reasons for putting an object under configuration management. But, possibly the most important reason for configuration management is to control the access to the different deliverable objects. Unless strict discipline is enforced regarding updating and storage of different objects, several problems appear. The following are some of the important problems that appear if configuration management is not used.

- **Inconsistency problem when the objects are replicated.** A scenario can be considered where every software engineer has a personal copy of an object (e.g. source code). As



each engineer makes changes to his local copy, he is expected to intimate them to other engineers, so that the changes in interfaces are uniformly changed across all modules. However, many times an engineer makes changes to the interfaces in his local copies and forgets to intimate other teammates about the changes. This makes the different copies of the object inconsistent. Finally, when the product is integrated, it does not work. Therefore, when several team members work on developing an object, it is necessary for them to work on a single copy of the object, otherwise inconsistency may arise.

- **Problems associated with concurrent access.** Suppose there is a single copy of a problem module, and several engineers are working on it. Two engineers may simultaneously carry out changes to different portions of the same module, and while saving overwrite each other. Though the problem associated with concurrent access to program code has been explained, similar problems occur for any other deliverable object.
- **Providing a stable development environment.** When a project is underway, the team members need a stable environment to make progress. Suppose somebody is trying to integrate module A, with the modules B and C, he cannot make progress if developer of module C keeps changing C; this can be especially frustrating if a change to module C forces him to recompile A. When an effective configuration management is in place, the manager freezes the objects to form a base line. When anyone needs any of the objects under configuration control, he is provided with a copy of the base line item. The requester makes changes to his private copy. Only after the requester is through with all modifications to his private copy, the configuration is updated and a new base line gets formed instantly. This establishes a baseline for others to use and depend on. Also, configuration may be frozen periodically. Freezing a configuration may involve archiving everything needed to rebuild it. (Archiving means copying to a safe place such as a magnetic tape).
- **System accounting and maintaining status information.** System accounting keeps track of who made a particular change and when the change was made.
- **Handling variants.** Existence of variants of a software product causes some peculiar problems. Suppose somebody has several variants of the same module, and finds a bug in one of them. Then, it has to be fixed in all versions and revisions. To do it efficiently, he should not have to fix it in each and every version and revision of the software separately.

### Software Configuration Management Activities

Normally, a project manager performs the configuration management activity by using an automated configuration management tool. A configuration management tool provides automated support for overcoming all the problems mentioned above. In addition, a configuration management tool helps to keep track of various deliverable objects, so that the project manager can quickly and unambiguously determine the current state of the project. The configuration management tool enables the engineers to change the various components in a controlled manner.

Configuration management is carried out through two principal activities:



- Configuration identification involves deciding which parts of the system should be kept track of.
- Configuration control ensures that changes to a system happen smoothly.

### **Configuration Identification**

The project manager normally classifies the objects associated with a software development effort into three main categories: controlled, pre controlled, and uncontrolled. Controlled objects are those which are already put under configuration control. One must follow some formal procedures to change them. Pre controlled objects are not yet under configuration control, but will eventually be under configuration control. Uncontrolled objects are not and will not be subjected to configuration control. Controllable objects include both controlled and pre controlled objects. Typical controllable objects include:

- Requirements specification document
- Design documents

Tools used to build the system, such as compilers, linkers, lexical analyzers, parsers, etc.

- Source code for each module
- Test cases
- Problem reports

The configuration management plan is written during the project planning phase and it lists all controlled objects. The managers who develop the plan must strike a balance between controlling too much, and controlling too little. If too much is controlled, overheads due to configuration management increase to unreasonably high levels. On the other hand, controlling too little might lead to confusion when something changes.

### **Configuration Control**

Configuration control is the process of managing changes to controlled objects. Configuration control is the part of a configuration management system that most directly affects the day-to-day operations of developers. The configuration control system prevents unauthorized changes to any controlled objects. In order to change a controlled object such as a module, a developer can get a private copy of the module by a reserve operation as shown in fig. 38.1. Configuration management tools allow only one person to reserve a module at a time. Once an object is reserved, it does not allow anyone else to reserve this module until the reserved module is restored as shown in fig. 38.1. Thus, by preventing more than one engineer to simultaneously reserve a module, the problems associated with concurrent access are solved.



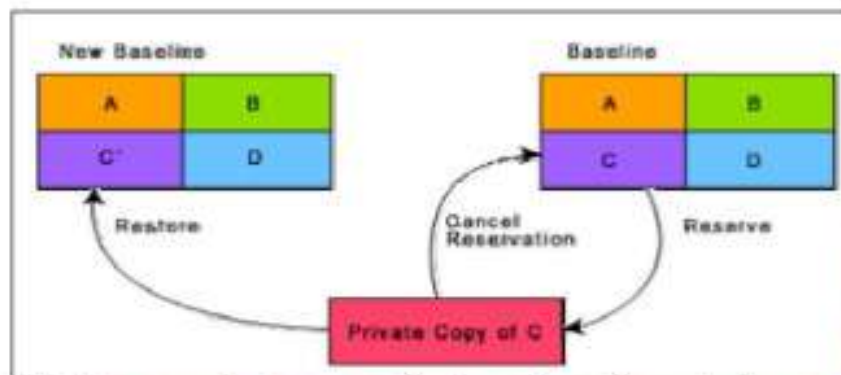


Fig. 38.1: Reserve and restore operation in configuration control

It can be shown how the changes to any object that is under configuration control can be achieved. The engineer needing to change a module first obtains a private copy of the module through a reserve operation. Then, he carries out all necessary changes on this private copy. However, restoring the changed module to the system configuration requires the permission of a change control board (CCB). The CCB is usually constituted from among the development team members. For every change that needs to be carried out, the CCB reviews the changes made to the controlled object and certifies several things about the change:

1. Change is well-motivated.
2. Developer has considered and documented the effects of the change.
3. Changes interact well with the changes made by other developers.
4. Appropriate people (CCB) have validated the change, e.g. someone has tested the changed code, and has verified that the change is consistent with the requirement.

The change control board (CCB) sounds like a group of people. However, except for very large projects, the functions of the change control board are normally discharged by the project manager himself or some senior member of the development team. Once the CCB reviews the changes to the module, the project manager updates the old base line through a restore operation (as shown in fig. 38.1). A configuration control tool does not allow a developer to replace an object he has reserved with his local copy unless he gets an authorization from the CCB. By constraining the developers' ability to replace reserved objects, a stable environment is achieved. Since a configuration management tool allows only one engineer to work on one module at any one time, problem of accidental overwriting is eliminated. Also, since only the manager can update the baseline after the CCB approval, unintentional changes are eliminated.



## 4.14. SOFTWARE REUSE

### Advantages of software reuse

Software products are expensive. Software project managers are worried about the high cost of software development and are desperately look for ways to cut development cost. A possible way to reduce development cost is to reuse parts from previously developed software. In addition to reduced development cost and time, reuse also leads to higher quality of the developed products since the reusable components are ensured to have high quality.

#### Artifacts that can be reused

It is important to know about the kinds of the artifacts associated with software development that can be reused. Almost all artifacts associated with software development, including project plan and test plan can be reused. However, the prominent items that can be effectively reused are:

- Requirements specification
- Design
- Code
- Test cases
- Knowledge

### Pros and cons of knowledge reuse

Knowledge is the most abstract development artifact that can be reused. Out of all the reuse artifacts i.e. requirements specification, design, code, test cases, reuse of knowledge occurs automatically without any conscious effort in this direction. However, two major difficulties with unplanned reuse of knowledge are that a developer experienced in one type of software product might be included in a team developing a different type of software. Also, it is difficult to remember the details of the potentially reusable development knowledge. A planned reuse of knowledge can increase the effectiveness of reuse. For this, the reusable knowledge should be systematically extracted and documented. But, it is usually very difficult to extract and document reusable knowledge.

### Easiness of reuse of mathematical functions

The routines of mathematical libraries are being reused very successfully by almost every programmer. No one in his right mind would think of writing a routine to compute sine or cosine. Reuse of commonly used mathematical functions is easy. Several interesting aspects emerge. Cosine means the same to all. Everyone has clear ideas about what kind of argument should cosine take, the type of processing to be carried out and the results returned. Secondly,



Mathematical libraries have a small interface. For example, cosine requires only one parameter. Also, the data formats of the parameters are standardized.

### **Basic issues in any reuse program**

The following are some of the basic issues that must be clearly understood for starting any reuse program.

- Component creation
- Component indexing and storing
- Component search
- Component understanding
- Component adaptation
- Repository maintenance

**Component creation-** For component creation, the reusable components have to be first identified. Selection of the right kind of components having potential for reuse is important. Domain analysis is a promising technique which can be used to create reusable components.

**Component indexing and storing-** Indexing requires classification of the reusable components so that they can be easily searched when looking for a component for reuse. The components need to be stored in a Relational Database Management System (RDBMS) or an Object-Oriented Database System (ODBMS) for efficient access when the number of components becomes large.

**Component searching-** The programmers need to search for right components matching their requirements in a database of components. To be able to search components efficiently, the programmers require a proper method to describe the components that they are looking for.

**Component understanding-** The programmers need a precise and sufficiently complete understanding of what the component does to be able to decide whether they can reuse the component. To facilitate understanding, the components should be well documented and should do something simple.

**Component adaptation-** Often, the components may need adaptation before they can be reused, since a selected component may not exactly fit the problem at hand. However, tinkering with the code is also not a satisfactory solution because this is very likely to be a source of bugs.

**Repository maintenance-** A component repository once is created requires continuous maintenance. New components, as and when created have to be entered into the repository.

The faulty components have to be tracked. Further, when new applications emerge, the older applications become obsolete. In this case, the obsolete components might have to be removed from the repository.

### **Domain Analysis**

The aim of domain analysis is to identify the reusable components for a problem domain.



**Reuse domain-** A reuse domain is a technically related set of application areas. A body of information is considered to be a problem domain for reuse, if a deep and comprehensive relationship exists among the information items as categorized by patterns of similarity among the development components of the software product. A reuse domain is shared understanding of some community, characterized by concepts, techniques, and terminologies that show some coherence. Examples of domains are accounting software domain, banking software domain, business software domain, manufacturing automation software domain, telecommunication software domain, etc.

Just to become familiar with the vocabulary of a domain requires months of interaction with the experts. Often, one needs to be familiar with a network of related domains for successfully carrying out domain analysis. Domain analysis identifies the objects, operations, and the relationships among them. For example, consider the airline reservation system, the reusable objects can be seats, flights, airports, crew, meal orders, etc. The reusable operations can be scheduling a flight, reserving a seat, assigning crew to flights, etc. The domain analysis generalizes the application domain. A domain model transcends specific applications. The common characteristics or the similarities between systems are generalized.

During domain analysis, a specific community of software developers gets together to discuss community-wide-solutions. Analysis of the application domain is required to identify the reusable components. The actual construction of reusable components for a domain is called domain engineering.

**Evolution of a reuse domain-** The ultimate result of domain analysis is development of problem-oriented languages. The problem-oriented languages are also known as application generators. These application generators, once developed form application development standards. The domains slowly develop. As a domain develops, it is distinguishable the various stages it undergoes:

**Stage 1:** There is no clear and consistent set of notations. Obviously, no reusable components are available. All software is written from scratch.

**Stage 2:** Here, only experience from similar projects is used in a development effort. This means that there is only knowledge reuse.

**Stage 3:** At this stage, the domain is ripe for reuse. The set of concepts are stabilized and the notations standardized. Standard solutions to standard problems are available. There is both knowledge and component reuse.

**Stage 4:** The domain has been fully explored. The software development for the domain can be largely automated. Programs are not written in the traditional sense any more. Programs are written using a domain specific language, which is also known as an application generator.



## REUSE APPROACH

### Components Classification

Components need to be properly classified in order to develop an effective indexing and storage scheme. Hardware reuse has been very successful. Hardware components are classified using a multilevel hierarchy. At the lowest level, the components are described in several forms: natural language description, logic schema, timing information, etc. The higher the level at which a component is described, the more is the ambiguity. This has motivated the Prieto-Diaz's classification scheme.

**Prieto-Diaz's classification scheme:** Each component is best described using a number of different characteristics or facets. For example, objects can be classified using the following:

**Searching-** The domain repository may contain thousands of reuse items. A popular search technique that has proved to be very effective is one that provides a web interface to the repository. Using such a web interface, one would search an item using an approximate automated search using key words, and then from these results do a browsing using the links provided to look up related items. The approximate automated search locates products that appear to fulfill some of the specified requirements. The items located through the approximate search serve as a starting point for browsing the repository. These serve as the starting point for browsing the repository. The developer may follow links to other products until a sufficiently good match is found. Browsing is done using the keyword-to-keyword, keyword-to-product, and product-to-product links. These links help to locate additional products and compare their detailed attributes. Finding a satisfactorily item from the repository may require several locations of approximate search followed by browsing. With each iteration, the developer would get a better understanding of the available products and their differences. However, we must remember that the items to be searched may be components, designs, models, requirements, and even knowledge.

**Repository maintenance** - Repository maintenance involves entering new items, retiring those items which are no more necessary, and modifying the search attributes of items to improve the effectiveness of search. The software industry is always trying to implement something that has not been quite done before. As patterns requirements emerge, new reusable components are identified, which may ultimately become more or less the standards. However, as technology advances, some components which are still reusable, do not fully address the current requirements. On the other hand, restricting reuse to highly mature components, sacrifices one of that creates potential reuse opportunity. Making a product available before it has been thoroughly assessed can be counterproductive. Negative experiences tend to dissolve the trust in the entire reuse framework.

**Application generator** -The problem- oriented languages are known as application generators. Application generators translate specifications into application programs. The specification is usually written using 4GL. The specification might also in a visual form. Application generator



can be applied successfully to data processing application, user interface, and compiler development.

### **Advantages of application generators**

Application generators have significant advantages over simple parameterized programs. The biggest of these is that the application generators can express the variant information in an appropriate language rather than being restricted to function parameters, named constants, or tables. The other advantages include fewer errors, easier to maintain, substantially reduced development effort, and the fact that one need not bother about the implementation details.

### **Shortcomings of application generators**

Application generators are handicapped when it is necessary to support some new concepts or features. Application generators are less successful with the development of applications with close interaction with hardware such as real-time systems.

### **Re-use at organization level**

Achieving organization-level reuse requires adoption of the following steps:

- Assessing a product's potential for reuse
- Refining products for greater reusability
- Entering the product in the reuse repository

**Assessing a product's potential for reuse.** Assessment of components reuse potential can be obtained from an analysis of a questionnaire circulated among the developers. The questionnaire can be devised to assess a component's reusability. The programmers working in similar application domain can be used to answer the questionnaire about the product's reusability. Depending on the answers given by the programmers, either the component be taken up for reuse as it is, it is modified and refined before it is entered into the reuse repository, or it is ignored. A sample questionnaire to assess a component's reusability is the following.

- Is the component's functionality required for implementation of systems in the future?
  - How common is the component's function within its domain?
  - Would there be a duplication of functions within the domain if the component is taken up?
  - Is the component hardware dependent?
  - Is the design of the component optimized enough?
  - If the component is non-reusable, then can it be decomposed to yield some reusable components?
- Can we parameterize a non-reusable component so that it becomes reusable?



**Refining products for greater reusability.** For a product to be reusable, it must be relatively easy to adapt it to different contexts. Machine dependency must be abstracted out or localized using data encapsulation techniques. The following refinements may be carried out:

- **Name generalization:** The names should be general, rather than being directly related to a specific application.
- **Operation generalization:** Operations should be added to make the component more general. Also, operations that are too specific to an application can be removed.
- **Exception generalization:** This involves checking each component to see which exceptions it might generate. For a general component, several types of exceptions might have to be handled.
- **Handling portability problems:** Programs typically make some assumption regarding the representation of information in the underlying machine. These assumptions are in general not true for all machines. The programs also often need to call some operating system functionality and these calls may not be same on all machines. Also, programs use some function libraries, which may not be available on all host machines. A portability solution to overcome these problems is shown in fig. 41.1. The portability solution suggests that rather than call the operating system and I/O procedures directly, abstract versions of these should be called by the application program. Also, all platform-related calls should be routed through the portability interface. One problem with this solution is the significant overhead incurred, which makes it inapplicable to many real-time systems and applications requiring very fast response.

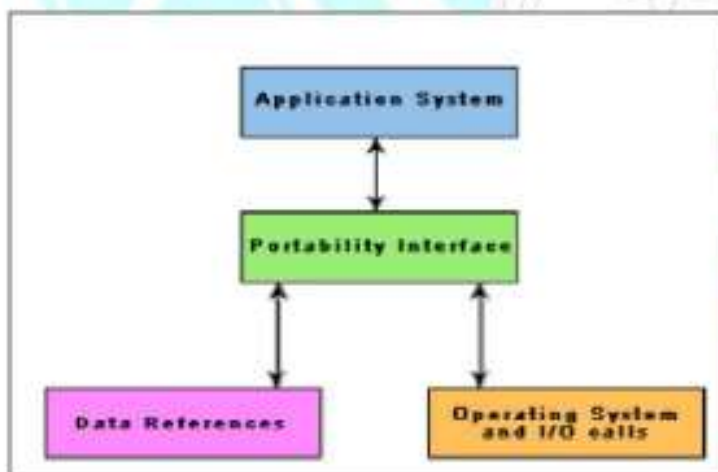


Fig. 41.1: Improving reusability of a component by using a portability interface.



In spite of all the shortcomings of the state-of-the-art reuse techniques, it is the experience of several organizations that most of the factors inhibiting an effective reuse program are non-technical. Some of these factors are the following.

- Need for commitment from the top management.
- Adequate documentation to support reuse.
- Adequate incentive to reward those who reuse. Both the people contributing new reusable components and those reusing the existing components should be rewarded to start a reuse program and keep it going.
- Providing access to and information about reusable components. Organizations are often hesitant to provide an open access to the reuse repository for the fear of the reuse components finding a way to their competitors.

## Software Engineering Multiple Choice Question with Answers

1. What are the characteristics of software?
- a. Software is developed or engineered; it is not manufactured in the classical sense.
  - b. Software doesn't "wear out".
  - c. Software can be custom built or custom build.
  - d. All mentioned above **ANSWER:**

**All mentioned above**

2. Compilers, Editors software come under which type of software?
- a. System software
  - b. Application software
  - c. Scientific software
  - d. None of the above.

**ANSWER: System software**

3. Software Engineering is defined as a systematic, disciplined and quantifiable approach for the development, operation and maintenance of software.
- a. True
  - b. False

**ANSWER: True**

4. RAD Software process model stands for .
- a. Rapid Application Development.
  - b. Relative Application Development.
  - c. Rapid Application Design.
  - d. Recent Application Development.

**ANSWER: Rapid Application Development.**

5. Software project management comprises of a number of activities, which contains \_\_\_\_\_
- a. Project planning
  - b. Scope management
  - c. Project estimation
  - d. All mentioned above

**ANSWER: All mentioned above**

6. COCOMO stands for .
- a. Consumed COst Model
  - b. COnstructive COst Model
  - c. COmmon COntrol Model
  - d. COmposition COst Model

**ANSWER: COnstructive COst Model**



7. Which of the following is not defined in a good Software Requirement Specification (SRS) document?

- a. Functional Requirement.
- b. Nonfunctional Requirement.
- c. Goals of implementation.
- d. Algorithm for software implementation.

ANSWER: Algorithm for software implementation.

8. What is the simplest model of software development paradigm?

- a. Spiral model
- b. Big Bang model
- c. V-model
- d. Waterfall model

ANSWER:

#### Waterfall model

9. Which of the following is the understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc?

- a. Software Design
- b. Feasibility Study
- c. Requirement Gathering
- d. System Analysis

ANSWER: System Analysis

10. Which design identifies the software as a system with many components interacting with each other?

- a. Architectural design
- b. High-level design
- c. Detailed design
- d. Both B & C

ANSWER: Architectural design

11. Software consists of .

- a. Set of instructions + operating procedures
- b. Programs + documentation + operating procedures
- c. Programs + hardware manuals
- d. Set of programs

ANSWER: Programs + documentation + operating procedures

12. Which is the most important feature of spiral model?

- a. Quality management
- b. Risk management
- c. Performance management
- d. Efficiency management

ANSWER: Risk management

13. If every requirement stated in the Software Requirement Specification (SRS) has only one interpretation, SRS is said to be correct .

- a. Unambiguous
- b. Consistent
- c. Verifiable
- d. None of the above

ANSWER: Unambiguous

14. Which is not a step of Requirement Engineering?

- a. Requirements elicitation
- b. Requirements analysis
- c. Requirements design
- d. Requirements documentation

ANSWER: Requirements design

15. FAST stands for .

- a. Functional Application Specification Technique
- b. Fast Application Specification Technique
- c. Facilitated Application Specification Technique
- d. None of the above

ANSWER: Facilitated Application Specification Technique

16. The level at which the software uses scarce resources is .

- a. Reliability
- b. Efficiency
- c. Portability
- d. All of the above

ANSWER: Efficiency

17. Modifying the software to match changes in the ever changing environment is called\_

- a. Adaptive maintenance
- b. Corrective maintenance
- c. Perfective maintenance
- d. Preventive maintenance

ANSWER: Adaptive maintenance

18. If every requirement can be checked by a cost-effective process, then the SRS is \_\_\_\_

- a. Verifiable
- b. Traceable
- c. Modifiable
- d. Complete

ANSWER: Verifiable

19. Aggregation represents .

- a. is\_a relationship
- b. part\_of relationship
- c. composed\_of relationship
- d. none of above

ANSWER: composed\_of relationship



20. If P is risk probability, L is loss, then Risk Exposure (RE) is computed as

- a.  $RE = P/L$
- b.  $RE = P + L$
- c.  $RE = P * L$
- d.  $RE = 2 * P * L$

ANSWER:  $RE = P * L$

21. Number of clauses used in ISO 9001 to specify quality system requirements are

- a. 15
- b. 20
- c. 25
- d. 28

ANSWER: 20

22. ER model shows the

- a. Static view
- b. Functional view
- c. Dynamic view
- d. All the above

ANSWER: Static view

23. IEEE 830-1993 is a IEEE recommended standard for

- a. Software Requirement Specification
- b. Software design
- c. Testing
- d. Both (A) and (B)

ANSWER: Software Requirement Specification

24. One of the fault base testing techniques is

- a. Unit Testing
- b. Beta Testing
- c. Stress Testing
- d. Mutation Testing

ANSWER: Mutation Testing

25. If the objects focus on the problem domain, then we are concerned with

- a. Object Oriented Analysis
- b. Object Oriented Design
- c. Object Oriented Analysis and Design
- d. None of the above

ANSWER: Object Oriented Analysis

26. In a risk-based approach the risks identified may be used to:

- i. Determine the test technique to be employed
  - ii. Determine the extent of testing to be carried out
  - iii. Prioritize testing in an attempt to find critical defects as early as possible.
  - iv. Determine the cost of the project
- a. ii is True; i, iii, iv and v are False
  - b. i,ii,iii are true and iv is false
  - c. ii and iii are True; i, iv are False

d. ii, iii and iv are True; i is false

ANSWER: i,ii,iii are true and iv is false

27. Which of the following is not a part of the Test Implementation and Execution Phase?

- a. Creating test suites from the test cases
- b. Executing test cases either manually or by using test execution tools
- c. Comparing actual results
- d. Designing the Tests

ANSWER: Designing the Tests

28. The Test Cases derived from use cases .

- a. Are most useful in uncovering defects in the process flows during real world use of the system.
- b. Are most useful in uncovering defects in the process flows during the testing use of the system.
- c. Are most useful in covering the defects in the process flows during real world use of the system.
- d. Are most useful in covering the defects at the Integration Level.

ANSWER: Are most useful in uncovering defects in the process flows during real world use of the system.

29. What can static analysis NOT find?

- a. The use of a variable before it has been defined.
- b. Unreachable ("dead") code.
- c. Memory leaks.
- d. Array bound violations.

ANSWER: Memory leaks.

30. Which plan describes how the skills and experience of the project team members will be developed?

- a. HR Plan
- b. Manager Plan
- c. Team Plan
- d. Staff Development Plan

ANSWER: Staff Development Plan

31. Alpha and Beta Testing are forms of

- a. Acceptance testing
- b. Integration testing
- c. System Testing
- d. Unit testing

ANSWER: Acceptance testing

32. The model in which the requirements are implemented by its category is

- a. Evolutionary Development Model
- b. Waterfall Model
- c. Prototyping
- d. Iterative Enhancement Model

ANSWER: Evolutionary Development Model

33. A COCOMO model is

- a. Common Cost Estimation Model.
- b. Constructive Cost Estimation Model.
- c. Complete Cost Estimation Model.



d. Comprehensive Cost Estimation Model

ANSWER: Constructive Cost Estimation Model.

34. SRD stands for .

- a. Software Requirements Definition
- b. Structured Requirements Definition
- c. Software Requirements Diagram
- d. Structured Requirements Diagram

ANSWER: Structured Requirements Definition

35. The tools that support different stages of software development life cycle are called\_\_

- a. CASE Tools
- b. CAME tools
- c. CAQE tools
- d. CARE tools

ANSWER: CASE Tools

36. Which defect amplification model is used to illustrate the generation and detection of errors during the preliminary steps of a software engineering process?

- a. Design
- b. Detailed design
- c. Coding
- d. All mentioned above

ANSWER: All mentioned above

37. Which method is used for evaluating the expression that passes the function as an argument?

- a. Strict evaluation
- b. Recursion
- c. Calculus
- d. Pure functions

ANSWER: Strict evaluation

38. Which factors affect the probable consequences if a risk occur?

- a. Risk avoidance
- b. Risk monitoring
- c. Risk timing
- d. Contingency planning

ANSWER: Risk timing

39. Staff turnover, poor communication with the customer are risks that are extrapolated from past experience are called\_\_\_\_\_

- a. Business risks
- b. Predictable risks
- c. Project risks
- d. Technical risks

ANSWER: Predictable risks

40. Organization can have in-house inspection, direct involvement of users and release of beta version are few of them and it also includes usability, compatibility, user acceptance etc. is called
- Task analysis
  - GUI requirement gathering
  - GUI design & implementation
  - Testing

ANSWER: Testing

41. Which project is undertaken as a consequence of a specific customer request?
- Concept development projects
  - Application enhancement projects
  - New application development projects
  - Application maintenance projects

ANSWER: New application development projects

42. Requirement engineering process includes which of these steps?
- Feasibility study
  - Requirement Gathering
  - Software Requirement specification & Validation
  - All mentioned above

ANSWER: All mentioned above

43. Software safety is a quality assurance activity that focuses on hazards that may cause an entire system to fall.
- True
  - False

ANSWER: True

44. Give the disadvantages of modularization.
- Smaller components are easier to maintain
  - Program can be divided based on functional aspects
  - Desired level of abstraction can be brought in the program
  - None of the above

ANSWER: None of the above

45. Effective software project management focuses on the four P's. What are those four P's?
- People, performance, payment, product
  - People, product, process, project
  - People, product, performance, project
  - All of the above.

ANSWER: People, product, process, project

46. Give the Real-world factors affecting maintenance Cost.
- As technology advances, it becomes costly to maintain old software.
  - The standard age of any software is considered up to 10 to 15 years.
  - Most maintenance engineers are newbie and use trial and error method to rectify problem.
  - All mentioned above

ANSWER: All mentioned above



47. Mention any two indirect measures of product.

- a. Quality
- b. Efficiency
- c. Accuracy
- d. Both A and B
- e. Both B and C

ANSWER: Both A and B

48. Which testing is the re-execution of some subset of tests that have already been conducted to ensure the changes that are not propagated?

- A. Unit testing
- B. Regression testing
- C. Integration testing
- D. Thread-based testing

ANSWER: Regression testing

49. State if the following are true for Project Management. During Project Scope management, it is necessary to –

- 1. Define the scope
- 2. Decide its verification and control
- 3. Divide the project into various smaller parts for ease of management.
- 4. Verify the scope

- a. True
- b. False

ANSWER: True

50. Software Requirement Specification (SRS) is also known as specification of .

- a. White box testing
- b. Acceptance testing
- c. Integrated testing
- d. Black box testing

ANSWER: Black box testing

## Part V: Internet Programming

### Module objective

At the end of this course, students will be able to:

- Know and understand basic web related terminologies
- Have core understanding on HTML
- Design a website with attractive interface
- Know the new features of HTML5
- Understand and use types of selectors in CSS
- Work on CSS to create responsive page
- Know Java script

### 5.1. Introduction to HTML

#### What is HTML?

HTML is the set of markup symbols or codes, which are called TAGs, inserted in a file intended for display on a World Wide Web browser page. The markup tells the Web browser how to display a Web page's words and images for the user. Each individual markup code is referred to as an element which is enclosed by tag. HTML is a formal Recommendation by the World Wide Web Consortium (W3C) and is generally adhered to by the web browsers. Simply HTML is a language for describing web pages.

- HTML stands for Hyper Text Markup Language
- HTML is a **markup** language, a **markup** language is a set of markup **tags**
- HTML documents contain HTML **tags** and plain **text**
- HTML documents are also called **web pages**

---

#### HTML Tags

---

HTML markup tags are usually called HTML tags. HTML tags are keywords surrounded by **angle brackets** like `<html>`. HTML tags are the hidden *keywords* within a web page that define how the browser must format and display the content.

Most tags must have two parts, an opening and a closing part. For example, `<html>` is the opening tag and `</html>` is the closing tag. Note that the closing tag has the same text as the opening tag, but has an additional forward-slash / character. There are some tags that are an exception to this rule, and where a closing tag is not required. The `<img>` tag for showing images is one example of this. Each HTML file must have the essential tags for it to be valid, so that web browsers can understand it and display it correctly.

- HTML tags normally **come in pairs** like `<b>` and `</b>`
- The first tag in a pair is the **start tag**, "`<b>`" the second tag is the **end tag** "`</b>`"
- The end tag is written like the start tag, with a **forward slash** before the tag name
- Start and end tags are also called **opening tags** and **closing tags**

#### HTML Elements

HTML element is everything between the start tag and the end tag, including the tags:

#### Example

- `<p>`This is paragraph`</p>` is a paragraph element.
- `<h>`this is heading`</h>` is a heading element.

The `<html>` element defines the whole document, since it is the opening and ending for every HTML document. It has a start tag `<html>` and an end tag `</html>`



The `<body>` element defines the document body. It has a start tag `<body>` and an end tag `</body>`. Every element of the `<body>` is displayed on the web browser depending on the tag's interpretation. Elements Don't forget closing tags, if the tag has. You might found unexpected output due to forgetting closing tag.

### Tag Attributes

Attributes allow you to customize a tag, and are defined within the opening tag, for example: `` or `<p align="center"> ... </p>`

On the above example, `img` is tag, `src` is attribute and `image1.jpg` is value and `p` is tag, `align` is attribute and `center` is value

Attributes are often assigned a value using the equals sign, such as `border="0"` or `width="50%"`, but there are some that only need to be declared in the tag like this: `<hr noshade>`. Most attributes are optional for most tags, and are only used when you want to change something about the default way a tag is displayed by the browser. However, some tags such as the `<img>` tag has required attributes such as `src` and `alt` which are needed in order for the browser to display the web page properly.

Attribute will be defined further with tags on coming pages

- Attributes provide additional information about an element
- Attributes are always specified in the start tag
- Attributes usually come in name/value pairs like: **name="value"**

**Example**, consider the following HTML code,

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
```

### Explanation:

The code has the following tags, `<html>`, `<head>`, `<title>`, `<body>`, `<h1>`, `<p>`

**<html>**

`<html>` tag is the main tag in which it includes all other tags in it. When you design a web, your html code should start with `<html>` tag and should end with `</html>`. The tag `<html>` has attribute `lang`, which is used to define the language used.

**<head>**

Is used to include different tags like, `<title>`, `<style>`, `<base>`, `<link>`, `<meta>`, `<script>` in which each of them have their own use. It should start immediately after the opening html tag and end before the opening body tag. Elements found in heading part are not to be displayed on the browser as the web content.

**<body>**



The content of body tag is the one which is to be displayed on your browser. Each element, according to their tag being processed and displayed on the browser.

### Web Browsers

The purpose of a web browser is to read HTML documents and display them as web pages. The browser does not display the HTML tags, but uses the tags to determine how the content of the HTML page is to be presented/ displayed to the user. The output for code written above is shown below. It has a title "page title" which is found as element with <title> under <head>. The output is displayed on a browser, and as you see, the tags can not be displayed, instead the content only be displayed as the tag orders. The text "my first heading" is found in the tag <h1>, that is why it has larger text size.

### HTML editing tools

Web pages can be created and modified by using professional HTML editors like Adobe Dreamweaver, Microsoft front page, Microsoft Expression Web and others. Just to learn html well, let us use simple text editor like Notepad (PC) or TextEdit (Mac). Notepad ++ can also be used.

### Understanding Tags

#### Heading

HTML headings are defined with the <h1>, <h2>, <h3>, <h4>, <h5> and <h6> tags. Where <h1> produce larger text and <h6> small text. When you use heading tag, the content will display alone on a single line.

#### Example

```
<h1>This is a heading</h1>
<h2>This is a heading</h2>
<h3>This is a heading</h3>
```

OUTPUT

**This is a heading**

**This is a heading**

**This is a heading**

As you can see on the above example, <h1> produce larger text than others <h2> less than h1 and continue like that with <h6> small than the other heading tags.

Heading tags have attribute "align" which is used to define the position of the heading text with left, center, right or justified alignment.

```
<h2 align="center">This is a heading</h2>
```

There is also "title" attribute that can be used with heading tags. The value of the title attribute will be displayed as a tooltip when you mouse over the paragraph.

```
<h2 title="heading">This is a heading</h2>
```

After the output is displayed, when you mouse over the text "this is heading", "heading" will appear as a tooltip.

### HTML Paragraphs

HTML paragraphs are defined with the <p> tag. Paragraph tag is used to enclose a text or paragraph so that you can apply changes like alignment. A sentence which is enclosed with <p> tag will be displayed as a paragraph.

#### Example

```
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
```

The above elements displays two line of sentence since each of them are under <p> tag.

#### Example

```
<p>this is paragraph
This is sentence </p>
```

This paragraph element displays a single line output, because the text is enclosed by only one <p>.



If you are not using tags like heading, paragraph, break the text that you write on several lines will be displayed as a single line on the browser tags.

### Horizontal rule

The `<hr>` tag defines a thematic break in an HTML page, and is most often displayed as a horizontal rule. It is used to separate content (or define a change) in an HTML page.

Attributes like "width", "align", "color" used to add extra feature to the `<hr>` output. If width is not stated, then the horizontal rule will fill from left to right and also it is center aligned by default.

```
<p>text before line</p>
<hr color="red" width="180px"
align="left">
<p>text after line</p>
```

OUTPUT

text before line

text after line

Based on the attribute applied, the line has color red, it is aligned to the left and has a width of 180 pixels.

Width and height of outputs are measured through pixel or percent. Pixel is represented by px and if it is percent use %.

### HTML Links

HTML links are defined with the `<a>` tag. `<a>` is called anchor tag and used to create link between pages.

#### Example

```
<a href="www.metitech.com"
target="_blank"> click hear </a>
```

OUTPUT

[click hear](http://www.metitech.com)

On this example, href is an attribute of anchor tag which is used to show the link which is called URL. "click hear" is the text to be displayed, as you see the output, on the page and when user clicks on it, "metitech.com" will open. Target is another attribute that can be used with anchor tag. It is used to determine where the link has to be opened. Its values can be "blank" which opens the link in new window, "self" which opens the link on that page by overwriting the previous, and others like "parent", "new" are also used. Target's value could be name of a frame if frame is used. Using anchor tag, href must be there.

### HTML Images

HTML images are defined with the `<img>` tag. `<img>` tag has no closing, but it has attributes which are used to show the location of the image, used to manage the size and look of the image.

#### Example

```

```

OUTPUT



As you see the above example, it is empty element, since it has no closing tag. But there are 4 attributes used. src is used to show the location of the image. "class.jpg" is the name of the image which is located on same space with the html file. alt used to display text description when the user hover the mouse on the image, width will manage the horizontal size of the image. 144 and 102 have no measurement so that they use pixel by default. If the image is found in a folder called "sample" which is other than the html file then the scr looks like:

```

```

Border is another attribute for specifying border for the image displayed.

```

```

 this image has border/frame when it displayed on browser.

### HTML Table

<table> tag is used to insert table to webpage. Table has various uses especially in previous time, it has been used for keeping alignment and layout. Tabular data represented using the <table> tag. It works together with <tr>, <td> and other additional tags. Tables are divided into table rows with the <tr> tag. Table rows are divided into table data with the <td> tag. A table row can also be divided into table headings with the <th> tag.

#### Cell padding and cell spacing

Cell in table is the intersection between raw and column. The table below has 6 cells. Cell spacing specifies the space between the cells.

Cell padding specifies the space between the cell content and its borders. If you do not specify padding, the table cells will be displayed without padding. Cell padding and spacing will be discussed later on CSS.

```
<table border="1">
<tr>
<th> Name</th>
<th> F.Name</th>
<th> Age</th>
</tr>
<tr>
<td>meron</td>
<td>zalalem</td>
<td>20</td>
</tr>
<tr>
<td>eve</td>
<td>solomon</td>
<td>19</td>
</tr>
</table>
```

OUTPUT

Name	F.Name	Age
meron	zalalem	20
eve	solomon	19

it has attribute "border" with value 1 which is why the table has border line. The table has three rows and that is because there is a <tr> tag opened and closed three times. Under each opened <tr>, first there are three <th> which are used to classify the data with column. And then there are <td>s under each of the last



two <tr>. Which has the same purpose with <th>. <th> is different from <td> in that <th> is used to define headers, and the text will become bold and somehow larger.

colspan and rowspan

Attribute values should always be enclosed in quotes. Double style quotes are the most

common  
Tables should  
and rows


is allowed.  
Merging cell in


Attribute colspan

```
<table border="1" >
  <tr>
<td colspan="2">&nbsp;</td>
  <td>&nbsp;</td>
</tr>
  <tr>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
  </tr>
  <tr>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
  </tr>
</table>
```

```
<table border="1">
  <tr>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
  <td rowspan="2">&nbsp;</td>
  </tr>
  <tr>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
  </tr>
  <tr>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
  </tr>
</table>
```

colspan is used to merge multiple cells on column side. rowspan used to merge cells on row side.

The left and right side codes deliver same output as the tables on left and right side consecutively. colspan="2" merges the two column cells, and rowspan="2" merges two row cells. If the number of cells merged are 3 then value of attribute colspan=3 or rowspan=3.

### HTML list

List is used to display listed items, objects or anything. A list could be displayed in ordered way, which is using 1,2,3...or unordered way. HTML offers three ways for specifying lists of information. All lists must contain one or more list elements.

<ul> - An unordered list. This will list items using plain bullets.

<ol> - An ordered list. This will use different schemes of numbers to list your items.

<dl> - A definition list. This arranges your items with its description.

&nbsp; is an HTML entity which is used to give space. Since multiple space on editor cannot be displayed on the browser, if multiple space is needed then &nbsp; can be used. 1 &nbsp; is used for single space.

### Unordered list

An unordered list is a collection of related items that have no special order or sequence. An unordered list starts with the `<ul>` tag. Each list item starts with the `<li>` tag. By default, the list items will be marked with bullets (small black circles)

### Example

```
<ul >
```

```
  <li>laptop</li>
```

```
  <li>desktop</li>
```

```
  <li>palmtop</li>
```

```
</ul>
```

OUTPUT

- laptop
- desktop
- palmtop

As the example above shows, `<ul>` gives us an output with no order or number. Instead it uses bullets. Every content enclosed with `<li>` ..... `</li>` under the `<ul>` tag was displayed with a bullet. If you want to change bullet style, then using type attribute helps.

### Example

```
<ul type="square">
```

```
  <li>laptop</li>
```

```
  <li>desktop</li>
```

```
  <li>palmtop</li>
```

```
</ul>
```

OUTPUT

- laptop
- desktop
- palmtop

The bullets on the above example are changed from disc to square. Attribute values like square, disc, circle and none can be used.

### Ordered list

If you are required to put your items in a numbered list instead of bulleted then HTML ordered list will be used. An ordered list starts with the `<ol>` tag. Each list item starts with the `<li>` tag. By default, the list items will be marked with numbers:

### Example

```
<ol >
```

```
  <li>laptop</li>
```

```
  <li>desktop</li>
```

```
  <li>palmtop</li>
```

OUTPUT

1. laptop
2. desktop

```
<ol type="A">
```

```
  <li>laptop</li>
```

```
  <li>desktop</li>
```

```
  <li>palmtop</li>
```

```
</ol>
```

OUTPUT

- A. laptop
- B. desktop
- C. palmtop

There is also start attribute that can be used with `<ol>`. It helps to indicate where the list counting has to start from.



```
<ol type="I" start="3">
  <li>laptop</li>
  <li>desktop</li>
  <li>palmtop</li>
</ol>
```

OUTPUT

III. laptop  
IV. desktop  
V. palmtop

The example shows us lists which are ordered in roman number and numbering starts from 3. Because attribute start has value 3.

#### Description list

A description list also called definition list is a list of terms, with a description of each term. The **<dl>** tag defines the description list, the **<dt>** tag defines the term (name), and the **<dd>** tag describes each term. It is used to list terms like in a dictionary or encyclopedia.

```
<dl>
  <dt>Laptop</dt>
  <dd>Movable computer</dd>
  <dt>Desktop</dt>
  <dd>Large in physical
size</dd>
</dl>
```

OUTPUT

Laptop  
Movable computer  
Desktop  
Large in physical size

In order to display special text on the web, formatting tags are must. There are lots of tags with different purpose.

Formatting Tag	Name of tag	Description
<a href="#"><b>&lt;b&gt;</b></a>	Bold	Defines bold text
<a href="#"><b>&lt;em&gt;</b></a>	Emphasized	Defines emphasized text/ has same output with <b>&lt;i&gt;</b>
<a href="#"><b>&lt;i&gt;</b></a>	Italics	Defines a part of text in an alternate voice or mood
<a href="#"><b>&lt;small&gt;</b></a>	Small	Defines smaller text
<a href="#"><b>&lt;strong&gt;</b></a>	Strong	Defines important text/ has same output with <b>&lt;b&gt;</b>
<a href="#"><b>&lt;sub&gt;</b></a>	Subscript	Defines subscripted text
<a href="#"><b>&lt;sup&gt;</b></a>	Superscript	Defines superscripted text
<a href="#"><b>&lt;ins&gt;</b></a>	Inserted	Defines inserted text
<a href="#"><b>&lt;del&gt;</b></a>	Deleted	Defines deleted text
<a href="#"><b>&lt;mark&gt;</b></a>	Marked	Defines marked/highlighted text
<a href="#"><b>&lt;strike&gt;</b></a>	Strike-out	Puts a line through the center of the text, just like <b>&lt;del&gt;</b>
<a href="#"><b>&lt;pre&gt;</b></a>	Preformatted text	Any number of space and punctuations between this tag will appear as it is



<code>&lt;center&gt;</code>	Center	It makes the text center aligned
<code>&lt;tt&gt;</code>	Type writer	Displays text with type writer mode
<code>&lt;q&gt;</code>	Quotation	Used to display text in double quotation

- HTML code does not allow multiple spaces to display on browser. Or browsers cannot interpret multiple spaces found in HTML code.
- Even if you have multiple spaces while writing html code, only one is displayed.
- Preformatted tag is used to display things which cannot be interpreted by browser
- Whatever number of lines written on html code editor, browser displays it on single line if not using tags like (not only) break, heading, paragraph.
- Any number of lines counts as one line, and any number of spaces count as one space.

### HTML break

`<br>` is break tag which is used to break a line. In HTML, if you are not using tags like `<br>`, `<p>` or heading tags, the text that you are writing with several lines displayed as a single line on the browser. So using `<br>` is must on the place that you want to break your line. `<br>` has no closing tag. It is an empty element.

### HTML comment

There are some notes or descriptions about the code that we don't want to display on the browser but to keep them on the editor to have clear idea about what is written there. Such types of text should be commented so that it won't be displayed. Comment is used to do such things. In html `<!-- -->` is used to comment do comment.

### HTML Font

Font is one of formatting tag used in html. Since CSS is doing much of its activity, font tag is become deprecate. But until you have knowledge about CSS, better to understand how it works. `<font>` has multiple attributes which can be used to change the color, size, face of text enclosed by the tag.

```
<font size="+1" color="red"
face="Arial"> formatted
text</font>
```

OUTPUT  
T

formatted text

Due to font tag is used; the text displayed has color red, face Arial and size larger than normal (+1). Size, color and face are attributes of font and used to change how displayed text looks like.

### HTML `<head>` element

HTML `<head>` tag is a Container Tag. All Header elements contains like general information about page, meta-information, style sheet URL and document type information. Head tag elements does not display in body part on web browser. The `<head>` element is a container for metadata (data about the page) and is placed between the `<html>` tag and the `<body>` tag.

- Head should be used just once.
- It should start immediately after the opening html tag and end directly before the opening `body` tag.
- HTML metadata is data about the HTML document. Metadata is not displayed.
- Metadata typically define the document title, character set, styles, links, scripts, and other meta information.
- It is container for the following tags: `<title>`, `<style>`, `<meta>`, `<link>`, `<script>`, and `<base>`.

`<title>`



The <title> element defines the title of the document. It is shown on the title bar of browser. In addition provides a title for the page when it is added to favorites and displays a title for the page in search engine results

```
<head>
<title> My official site</title>
</head>
```

#### <link>

The HTML <link> tag is used to specify relationships between the current document and external resource.

```
<head>
<link rel="stylesheet" type="text/css" href="/css/style.css">
</head>
```

This example shows how to insert icon on the title bar of the web page. Image called, logo.png displayed on the title bar. Many more linkages processed using link tag.

```
<head>
<link rel="shortcut icon" href="images/LOGO.png" />
</head>
```

Meta tag is used to specify additional important information about a document in a variety of ways. Meta tag is an empty element without a closing tag but it carries information within its attributes. You can include one or more meta tags in your document based on what information you want to keep in your document but in general, meta tags do not impact physical appearance of the document so from appearance point of view, it does not matter if you include them or not. Metadata is used by browsers (how to display content), by search engines (keywords), and other web services.

The first Meta is about keywords, means that if someone searches a page with either of those keywords, then your page will be within the list. The second Meta is used to refresh your page by itself. As it is stated on content attribute, the page refreshes every 30 second.

```
<head>
<meta name="keywords" content="HTML,CSS,XML,JavaScript">
<meta http-equiv="refresh" content="30">
</head>
```

Meta tags are a great way for webmasters to provide search engines with information about their sites.

This tag is used to enclose a client side scripting language which is java script. It can also link external java script page to the current page using attribute src.

#### <style>

The <style> tag is used to define style information for an HTML document. Inside the <style> element you specify how HTML elements should render in a browser; CSS is defined in style tag. Each HTML document can contain multiple <style> tags.

#### HTML block and inline elements

Most HTML elements are defined as **block level** elements or as **inline** elements. Block level elements normally start (and end) with a new line when displayed in a browser.



Inline elements, on the other hand, are normally displayed without starting a new line.

### HTML div

Div tag is the very important block level tag which plays a big role in grouping various other HTML tags and applying CSS on group of elements. This time it is used to create webpage layout where we define different parts (Left, Right, Top etc) of the page which was done using table concept in previous time. The <div> element is often used as a container for other HTML elements. It has no required attributes, but both **style** and **class** are common. When used together with CSS, the <div> element can be used to style blocks of content.

The <div> element has no special meaning. Except that, because it is a block level element, the browser will display a line break before and after it.

### HTML span

The HTML <span> is an inline element and it can be used to group inline-elements in an HTML document. This tag does not provide any visual change on the block but has more meaning when it is used with CSS. It is often used as a container for some text.

The difference between the <span> tag and the <div> tag is that the <span> tag is used with inline elements where as the <div> tag is used with block-level elements.

### Iframe

<iframe> is used to define inline frame on your page. An inline frame is used to embed another document within the current HTML document.

```
<iframe src="example1.html" scrolling="no" frameborder="0"
name="sample" ></iframe>
```

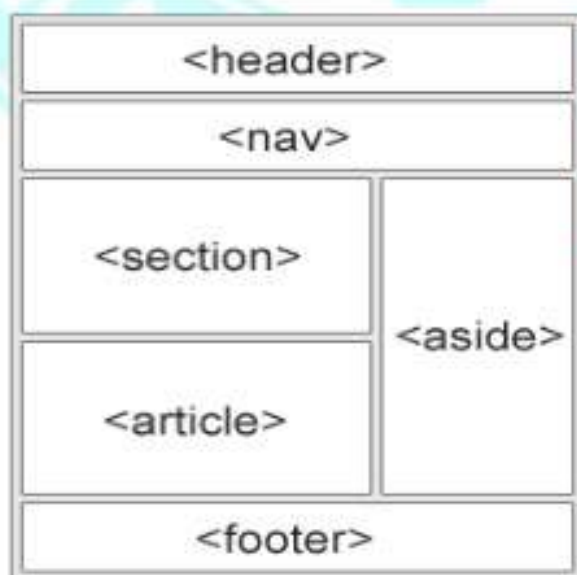
Attribute src is required because it defines the page to include inside the iframe. src="example1.html" shows that the iframe will have a content of example1.html. Scrolling is another attribute which used to determine whether the iframe is fixed or scrollable, so if it is no, then it won't scroll and if it is yes, it will. Frame border defines border of the iframe, since it is 0, then the border is not visible. Name is another attribute which is very essential in which if you want to display another page on iframe created, then your link target will use this name instead of "blank", "self" or others.

The following link will open on the above iframe. Because on target attribute, the name of the iframe is given. ("sample")

```
<a href="www.metitech.com" target="sample">open here</a>
```

### HTML Layout Elements

HTML has several semantic elements that define the different parts of a web page. A semantic element clearly describes its meaning to both the browser and the developer.





**Non-semantic** elements like: `<div>` and `<span>` - tells nothing about its content. However, **semantic** elements like listed below, are used to clearly define its content. List of semantic elements in HTML are listed below.

- `<header>` - Defines a header for a document or a section
- `<nav>` - Defines a set of navigation links
- `<section>` - Defines a section in a document
- `<article>` - Defines an independent, self-contained content
- `<aside>` - Defines content aside from the content (like a sidebar)
- `<footer>` - Defines a footer for a document or a section
- `<details>` - Defines additional details that the user can open and close on demand
- `<summary>` - Defines a heading for the `<details>` element

### HTML Entities

Entities in HTML are used to replace reserved characters. Characters, not present on your keyboard, can also be replaced by entities. Some characters are reserved in HTML. If you use the less than (`<`) or greater than (`>`) signs in your text, the browser might mix them with tags. A character entity looks like this:

`&entity_name;` OR `&#entity_number;`

- 
- Advantage of using an entity name is an entity name is easy to remember.
  - Disadvantage of using an entity name is browsers may not support all entity names, but the support for numbers is good.
  - Entity names are case sensitive
- 

Below are some of entities used in HTML

Result	Description	Entity Name	Entity Number
	non-breaking space	<code>&amp;nbsp;</code>	<code>&amp;#160;</code>
<code>&lt;</code>	less than	<code>&amp;lt;</code>	<code>&amp;#60;</code>
<code>&gt;</code>	greater than	<code>&amp;gt;</code>	<code>&amp;#62;</code>
<code>&amp;</code>	Ampersand	<code>&amp;amp;</code>	<code>&amp;#38;</code>
€	Euro	<code>&amp;euro;</code>	<code>&amp;#8364;</code>
©	copyright	<code>&amp;copy;</code>	<code>&amp;#169;</code>
®	registered trademark	<code>&amp;reg;</code>	<code>&amp;#174;</code>

### HTML Forms

HTML Forms are one of the main points of interaction between a user and a web site or application. They allow users to send data to the web site. Most of the time that data is sent to the web server, but the web page can also intercept it to use it on its own.

Html form contains form elements and Form elements can be in different types of input elements, checkboxes, radio buttons, submit buttons, and others. A form can also contain select lists, textarea, fieldset, legend, and label elements.

## The Input Element

The most important form element is the `<input>` element. The `<input>` element is used to select user information. An `<input>` element can vary in many ways, depending on the type attribute. An `<input>` element can be of type text field, checkbox, password, radio button, submit button, and more. The most common input types are described below.

```
<form name="formname" method="post" action="">
  ID:    <input type="text" name="id"><br><br>
  Name:  <input type="text" name="name"><br><br>
  Password: <input type="password" name="pass">
<p>Sex</p>
  <input type="radio" name="gen" checked="checked" >  Male
  <input type="radio" name="gen">      Female
<p>Hobby</p>
  <input type="checkbox" name="ho">  Movie
  <br><br>
  <input type="checkbox" name="ho">  Swimming
  <br><br>
  <input type="checkbox" name="ho">  Football<br>
  <input type="submit" name="button" value="Submit">
</form>
```

OU  
TPU



## Colors

HTML colors are specified with predefined color names, or with RGB, HEX, HSL, RGBA, or HSLA values. HSL stands for hue, saturation, and lightness. HSLA color values are an extension of HSL with an Alpha channel (opacity).

### **hsl(hue, saturation, lightness)**

- Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, and 240 is blue.
- Saturation is a percentage value, 0% means a shade of gray, and 100% is the full color.
- Lightness is also a percentage value, 0% is black, and 100% is white.

### **hsla(hue, saturation, lightness, alpha)**

- The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all)

## RGB Color Values

In HTML, a color can be specified as an RGB value, using this formula:

**rgb(red, green, blue)**



Each parameter (red, green, and blue) defines the intensity of the color with a value between 0 and 255. This means that there are  $256 \times 256 \times 256 = 16777216$  possible colors! For example,

- `rgb(255, 0, 0)` is displayed as red, because red is set to its highest value (255), and the other two (green and blue) are set to 0.
- `rgb(0, 255, 0)` is displayed as green, because green is set to its highest value (255), and the other two (red and blue) are set to 0.
- To display black, set all color parameters to 0, like this: `rgb(0, 0, 0)`.
- To display white, set all color parameters to 255, like this: `rgb(255, 255, 255)`.

### RGBA Color Values

RGBA color values are an extension of RGB color values with an Alpha channel - which specifies the opacity for a color.

**`rgba(red, green, blue, alpha)`**

- The alpha parameter is a number between 0.0 (fully transparent) and 1.0 (not transparent at all)

### HEX Color Values

A hexadecimal color is specified with: `#RRGGBB`, where the RR (red), GG (green) and BB (blue) hexadecimal integers specify the components of the color. In HTML, a color can be specified using a hexadecimal value in the form: `#rrggbb` where rr (red), gg (green) and bb (blue) are hexadecimal values between 00 and ff (same as decimal 0-255).

- For example, `#ff0000` is displayed as red, because red is set to its highest value (ff), and the other two (green and blue) are set to 00.
- Another example, `#00ff00` is displayed as green, because green is set to its highest value (ff), and the other two (red and blue) are set to 00.
- To display black, set all color parameters to 00, like this: `#000000`.
- To display white, set all color parameters to ff, like this: `#ffffff`.

### HTML Emojis

Emojis look like images, or icons, but they are not. They are letters (characters) from the UTF-8 (Unicode) character set. UTF-8 covers almost all of the characters and symbols in the world. To display an HTML page correctly, a web browser must know the character set used in the page. This is specified in the `<meta>` tag:

`<meta charset="UTF-8">`

If not specified, UTF-8 is the default character set in HTML. Many UTF-8 characters cannot be typed on a keyboard, but they can always be displayed using numbers (called entity numbers):

The `<meta charset="UTF-8">` element defines the character set. The characters A, B, and C, are displayed by the numbers 65, 66, and 67. To let the browser understand that you are displaying a character, you must start the entity number with `&#` and end it with `;` (semicolon).

### Emoji Characters

Emojis are also characters from the UTF-8 alphabet. Since Emojis are characters, they can be copied, displayed, and sized just like any other character in HTML. Let us see some examples about displaying emoji.

### HTML Video

The HTML `<video>` element is used to show a video on a web page. To show a video in HTML, use the `<video>` element:



```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogv" type="video/ogg">
  Your browser does not support the video tag.
</video>
```

The controls attribute adds video controls, like play, pause, and volume. It is a good idea to always include width and height attributes. If height and width are not set, the page might flicker while the video loads. The <source> element allows you to specify alternative video files which the browser may choose from. The browser will use the first recognized format. The text between the <video> and </video> tags will only be displayed in browsers that do not support the <video> element. To start a video automatically, use the autoplay attribute:

```
<video width="320" height="240" autoplay>
```

**Note:** Chromium browsers do not allow autoplay in most cases. However, muted autoplay is always allowed. Add muted after autoplay to let your video start playing automatically (but muted)

```
<video width="320" height="240" autoplay muted>
```

### HTML Audio

The HTML <audio> element is used to play an audio file on a web page. To play an audio file in HTML, use the <audio> element:

```
<audio controls>
  <source src="horse.ogg" type="audio/ogg">
  <source src="horse.mp3" type="audio/mpeg">
  Your browser does not support the audio element.
</audio>
```

The controls attribute adds audio controls, like play, pause, and volume. The <source> element allows you to specify alternative audio files which the browser may choose from. The browser will use the first recognized format. The text between the <audio> and </audio> tags will only be displayed in browsers that do not support the <audio> element. To start an audio file automatically, use the autoplay attribute:

```
<audio controls autoplay>
```

**Note:** Chromium browsers do not allow autoplay in most cases. However, muted autoplay is always allowed. Add muted after autoplay to let your audio file start playing automatically (but muted):

```
<audio controls autoplay muted>
```

## 5.2. Cascading Style Sheet (CSS)

CSS (Cascading Style Sheets) is a style sheet language used for describing the presentation of a document written in a markup language. It defines **how to display** HTML elements.

CSS has lots of advantages on your web like, saves time, loads faster, easy to maintain, platform independence, multi device compatibility and others.

### CSS Syntax

A CSS comprises of style rules that are interpreted by the browser and then applied to the corresponding elements in your document. A style rule is made of three parts namely selector, property and value.



- On this code, p is a selector.
- Background-color and border are properties
- and #B9B9B9 and dashed are values.

```
p
{
background-color:#B9B9B9;
border:dashed;
}
```

Selector and property must separate with { and you have to close it at the end. Property and value must separate with : . And you have to put ; at the end of each line (property : value;). Property has a value that will be interpreted by the browser to be applied on the html in which the selector is called. In the above code the selector is p, which is paragraph tag, means that in the html code every text which is enclosed by <p> will have a background color and border.

Don't forget a CSS declaration always ends with a semicolon, and declaration groups are surrounded by curly brackets.

### Example

```
p {
color:red;
text-align:center;
}
```

In this case every text enclosed by <p> will have red font color and it will align to center.

### CSS Comments

Comments are used to explain your code, and may help you when you edit the source code at a later date. Comments are ignored by browsers. A CSS comment starts with /\* and ends with \*/. Comments can also address multiple lines:

### Example

```
/*This is a multiple
lines comment*/
p
{
color:red;
/*This is another comment*/
text-align:center;
}
```

Text within comment symbol will not be considered by the browser.

### Ways to Insert CSS

There are three ways of inserting a style sheet:

- External style sheet
- Internal style sheet
- Inline style

### External Style Sheet

External style sheet is one way of inserting CSS to your page. In this case CSS code will be saved as a file with file extension of .css. An external style sheet is ideal when the style is applied to many pages. With an external style sheet, you can change the look of an entire Web site by changing one file. Each page must link to the style sheet using the <link> tag. The <link> tag goes inside the head section:

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css">
</head>
```

Do not add a space between the property value and the unit (such as margin-left:20 px). The correct way is: margin-left:20px tag.

### Internal Style Sheet

This type of CSS is defined within the page itself, so no need of linking. An internal style sheet should be used when a single document has a unique style. You define internal styles in the head section of an HTML page, by using the <style> tag, like this:

```
<head>
<style>
hr{color:purple;}
p{margin-left:20px;}
body{background-image:url("images/background.gif");}
</style>
</head>
```

It is possible to have multiple <style> defined in the heading part.

### Inline Styles

Inline styles are defined very near to the element in which the style is to be applied. It is defined within the HTML file in the body part. To use inline styles you use the style attribute in the relevant tag. The style attribute can contain any CSS property. The example shows how to change the color and the left margin of a paragraph:

```
<p style="color:red;margin-left:20px;">This is paragraph.</p>
```

While defining inline style sheet, the attribute style is used and the value should be enclosed with " ". You are not to use { }, unlike the other two styles.

- Better to use External style sheet if you have multiple pages that needs styling.
- External styles should be saved with extension .css, and you have to use <link> tag in heading part to link it with the current page.



- External css styles are applicable only on the page which they are linked.
- Internal style sheet styles are defined within the HTML page itself. It is must to define them in heading part using the tag <style>.

### **CSS Selectors**

CSS selectors allow you to select and manipulate HTML element(s). CSS selectors are used to "find" (or select) HTML elements based on their id, classes, types, attributes, values of attributes and much more.

#### **The Universal Selectors**

Rather than selecting elements of a specific type, the universal selector quite simply matches the name of any element type and it is defined using "\*". If you define style with universal selector, then it will apply on every element.

```
* {
  color: #000000;
}
```

This rule renders the content of every element in our document in black.

#### **The Descendant Selectors**

Suppose you want to apply a style rule to a particular element only when it lies inside a particular element. As given in the following example, style rule will apply to <b> element only when it lies inside <td> tag.

```
td b {
  color: #000000;
}
```

In this case the style will apply to elements in the <td> tag and enclosed with <b>.

#### **The element Selector**

The element selector selects elements based on the element name. You can select all <p> elements on a page like this: (all <p> elements will be center-aligned, with a red text color)

```
p {
  text-align:center;
  color:red;
}
```

#### **The Class Selectors**

The class selector finds elements with the specific class. The class selector uses the HTML class attribute. To find elements with a specific class, write a period character, followed by the name of the class. In the example below, all HTML elements with class="center" will be center-aligned:

```
.center
{
  text-align:center;
  color:red;
}
```

This class should be called like this:

```
<h1 class="center"> centered text</h1>
```

The text will be center aligned and also red color, since class with such style is called. This class can be called within any tag using attribute "class".

You can also specify that only specific HTML elements should be affected by a class. In the example below, all p elements with class="center" will be center-aligned:

```
p.center
{
  text-align:center;
```

```
color:red;
}
```

On this example, class "center" cannot be called by any tag because it is specifically defined for <p> as it is shown on the selector (p.center).

### The id Selector

The id selector uses the id attribute of an HTML tag to find the specific element. An id should be unique within a page, so you should use the id selector when you want to find a single, unique element.

To find an element with a specific id, write hash character #, followed by the id of the element. The style rule below will be applied to the HTML element with id="id1":

```
#id1
{
text-align:center;
color:red;
}
```

This id should be called like this:

```
<h1 id="id1"> centered text</h1>
```

You can make it a bit more particular:

```
h1#black {
color: #000000;
}
```

Or

```
#black h1 {
color: #000000;
}
```

This defines the content in black for only <h1> elements with *id* attribute set to *black*.

Do not start id name and class name with a number and don't forget that class has to start with . and id has to start with #.

### The Attribute Selectors

You can also apply styles to HTML elements with particular attributes. The style rule below will match all the input elements having a type attribute with a value of *text*

```
input[type = "text"]{
color: #000000;
}
```

The advantage to this method is that the <input type = "submit"> element is unaffected, and the color applied only to the desired text fields.

### Grouping Selectors

In style sheets there are often elements with the same style:

```
h1
{
text-align:center;
color:red;
}
h2
{
text-align:center;
```



```
color:red;
}
p
{
text-align:center;
color:red;
}
```

To minimize the code, you can group selectors. To group selectors, separate each selector with a comma. In the example below we have grouped the selectors from the code above:

```
h1,h2,p
{
text-align:center;
color:red;
}
```

### CSS Links

Link is one of the tags that need CSS to be applied. As you see on previous lesson, html links are not attractive. In order to make it user attractive, you have to apply CSS. Links can be styled with any CSS property (e.g. color, font-family, background, etc.). In addition, links can be styled differently depending on what state they are in. The four links states are:

- a:link - a normal and any link but not visited
- a:visited - a link the user has visited
- a:hover - a link when the user's mouse over it
- a:active - a link the moment it is clicked

### CSS Margin and padding

Padding property allows you to specify how much space should appear between the content of an element and its border (inside the border). The padding property sets the padding space on all sides of an element. The [padding area](#) is the space between the content of the element and its border. Negative values are not allowed. The value of this attribute should be a length, a percentage, or the word "inherit". If the value is "inherit", it will have the same padding as its parent element. If a percentage is used, the percentage is of the containing box. The padding property is a shorthand property for the following individual padding properties:

- padding-top
- padding-right
- padding-bottom
- padding-left

### Example

```
h1 {
padding: 10px 20px 30px 15px;
}
```

This means, top padding is 10px, right padding is 20px, bottom padding is 30px and left padding is 15px. Or you can specify specifically using the above four padding properties. If padding has three values, then the first is for top, the second is for right and left and the third will be for bottom. If it has two values, the first is for top and bottom and the second will be for right and left. If it has one value, then all the four will take that value.



```
div {  
  width: 300px;  
  padding: 25px;  
  box-sizing: border-box;  
}
```

Use the box-sizing property to keep the width at 300px, no matter the amount of padding is. The example above shows that the width of the box will remain at 300px even if there is padding of 25px on all sides.

The CSS **margin** properties are used to generate space around elements (outside the border). The margin does not have a background color, and is completely transparent. It is possible to use negative values to overlap content. The value of this attribute should be a length or a percentage. Percentage specifies a margin in % of the width of the containing element. The margin property allows you set all of the properties for the four margins (margin-top, margin-right, margin-bottom and margin-left) in one declaration. It works as the same fashion as padding does.

Negative values are not allowed for padding but possible in the case of margin.

The border properties allow you to specify how the border of the box representing an element should look. There are three properties of a border you can change, **border-color** specifies the color of a border, **border-style** specifies what kind of border to display (the values are listed below) and **border-width** specifies the width of a border.

Border style has to be set to see the value of others (color and width). Some of values of border-style are:

- none: Defines no border
- dotted: Defines a dotted border
- dashed: Defines a dashed border
- solid: Defines a solid border
- double: Defines two borders.

The border-width property is used to set the width of the border. The width is set in pixels, or by using one of the three pre-defined values: thin, medium, or thick. The "border-width" property does not work if it is used alone. Use the "border-style" property to set the borders first.

The border-color is used to specify the color of border to be displayed. Color can be assigned using hexadecimal, RGB or by its name. You can also set the border color to "transparent".

### Border - Individual sides

It is possible to have different borders for different sides using:

- border-top-style
- border-right-style
- border-bottom-style
- border-left-style

Or you can also use "border-style" only with a value from one to four. If it has four values like:

**border-style:dotted solid double dashed;**

It means: top border is dotted, right border is solid, bottom border is double and left border is dashed. Or if it has three values:



**border-style:dotted solid double;**

It means: top border is dotted, right and left borders are solid and bottom border is double

**border-style:dotted solid;**

In this case, top and bottom borders are dotted and right and left borders are solid. And if it is like:

**border-style:dotted;**

Then all four borders are dotted.

To shorten the code, it is also possible to specify all the individual border properties in one property. The "border" property is a shorthand property for the border-width, border-style (required) and border-color.

**Example**

```
p {  
  border: thin solid red;  
}
```

After setting border for the element, sometimes we may need to have non sharp edge on every angle or some of them. "border-radius" is used for that.

```
p {  
  border: thin solid red;  
  border-radius:20px;  
}
```

Four sides of the border are rounded with 20px. As the pixel increases the roundness of the edges increases.

```
p {  
  border: thin solid red;  
  border-bottom-left-radius:20px;  
  border-top-right-radius:20px;  
}
```

On the above example "border-bottom-left-radius" is set with 20px and the left bottom side is rounded as you can see it. The same for "border-top-right-radius".

**CSS table**

Table is one of the HTML elements that can improve using CSS. There are lots of futures that we can see. "border" property is used to specify table's border.

```
table {  
  border: 1px solid black;  
}
```

OUTPUT

cell 11	cell 12	cell 13
cell 21	cell 22	cell 23

The CSS code above gives border for outer table, not for each cell. Check the following code:

```
table td {  
  border: 1px solid black;  
}  
table {  
  border-collapse:collapse;  
}
```

OUTPUT

cell 11	cell 12	cell 13
cell 21	cell 22	cell 23

Here both table and td have border with 1px solid and black color, so that the table will look like as the output. "border-collapse" property helps the table to have single border, if it was not set or if its value is "separate" then the border looks like each cell has its own rectangle border.

Other properties like "vertical-align", "text-align", "padding", "height", "width", "color", "background-color" can also be used.

```
td {  
    border-bottom: 1px solid black;  
}  
table{  
    border-collapse:collapse;  
}  
tr:hover{  
    background-color:#CFF;  
}  
tr:nth-child(odd)  
{  
    background-color: #d2d2d2;}
```

OUTPUT

cell 11	cell 12	cell 13
cell 21	cell 22	cell 23
cell 31	cell 32	cell 33
cell 41	cell 42	cell 43

"border-bottom" allows having border only on the bottom part. "tr:hover" is used to apply change when the user takes mouse over the row. In this case the background color will change when mouse over through the row. "tr:nth-child(odd)" will apply change according to the value on every odd rows. You may change "odd" with "even".

### CSS shadow

Having shadow to text, elements and boxes using CSS is possible through CSS3. There are two major shadow properties "text-shadow" and "box-shadow".

```
h1 {  
    text-shadow: 3px 3px 2px blue;  
}
```

OUTPUT

**Text with shadow**

Values for "text-shadow", the first 3px is for horizontal shadow, the next one is for vertical shadow. 2px is called blur effect, how much should the color distribute as a shadow. And "blue" is the color of the shadow.

```
div {  
    border:solid thin;  
    position:absolute;  
    width:300px;  
    height:300px;  
    box-shadow: 10px 10px 5px 2px grey;  
}
```

OUTPUT






The first 10px shows horizontal shadow, the next is for vertical shadow. 5px is for blur effect and 2px for shadow spread radius. "grey" is color for shadow.

### CSS List

We can apply different style with properties that we know still on lists. In addition there are some properties that will help us specifically. "list-style-type" used to determine the list type like, square or disk or letter or number. "list-style-position" has a value of either inside or outside which displays lists with some padding or without. "list-style-image" specifies an image as the list item marker.

```
ul {  
  background: #666;  
  padding: 20px;  
}  
ul li {  
  background: #96F;  
  margin: 5px;  
  color:white;  
}
```

OUTPUT

- 
- Coffee
  - Tea
  - Coca Cola

"ul" has its own background set and padding value. "li" under every ul has another background color, text color and padding set.

### CSS Background

CSS background is used to define background for html elements. Properties used for background effects:

- background-color
- background-image
- background-repeat
- background-attachment
- background-position

#### Background Color

The background-color property specifies the background color of an element. The background color of a page is defined in the body selector. You may have background color for different elements like for h1, p or others.

#### Background Image

The background-image property specifies an image to use as the background of an element. By default, the background-image property repeats an image both horizontally and vertically so it covers the entire element. The background image for the whole page can be set like this:

```
body {  
  background-image:url("paper.gif");  
}
```

### CSS Text

#### Text Color

The color property is used to set the color of the text. With CSS, a color is most often specified by:

- a HEX value - like "#ff0000"
- an RGB value - like "rgb(255,0,0)"
- a color name - like "red"

```
body{Color:red;}
```

#### Text Alignment



The text-align property is used to set the horizontal alignment of a text. Text can be centered, or aligned to the left or right, or justified. When text-align is set to "justify", each line is stretched so that every line has equal width, and the left and right margins are straight (like in magazines and newspapers).

```
h1{text-align:center;}
```

### Text Transformation

The text-transform property is used to specify uppercase and lowercase letters in a text. It can be used to turn everything into uppercase or lowercase letters, or capitalize the first letter of each word.

```
p.uppercase{text-transform:uppercase;}
```

```
p.lowercase{text-transform:lowercase;}
```

Since p.uppercase and p.lowercase are class selectors, they can only be applied if you call them in <p> as <p class="uppercase"> or <p class="lowercase">

### CSS Font

CSS font properties define the font family, boldness, size, and the style of a text.

#### Font Family

The font family of a text is set with the font-family property. The font-family property should hold several font names as a "fallback" system. If the browser does not support the first font, it tries the next font. Start with the font you want, and end with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.

**Note:** If the name of a font family is more than one word, it must be in quotation marks, like: "Times New Roman".

```
p{font-family:"Times New Roman";}
```

#### Font Size

The font-size property sets the size of the text. Being able to manage the text size is important in web design. However, you should not use font size adjustments to make paragraphs look like headings, or headings look like paragraphs.

Always use the proper HTML tags, like <h1> - <h6> for headings and <p> for paragraphs. The font-size value can be an absolute or relative size. Font size can be set with px or em. Where 1em=16px.

```
h1{font-size:40px;}
```

```
h1{font-size:2em;}
```

### Responsive Web Design - The Viewport

#### What is The Viewport?

The viewport is the user's visible area of a web page. The viewport varies with the device, and will be smaller on a mobile phone than on a computer screen. Before tablets and mobile phones, web pages were designed only for computer screens, and it was common for web pages to have a static design and a fixed size. Then, when we started surfing the internet using tablets and mobile phones, fixed size web pages were too large to fit the viewport. To fix this, browsers on those devices scaled down the entire web page to fit the screen. This was not perfect!! But a quick fix.

#### Setting The Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the <meta> tag. You should include the following <meta> viewport element in all your web pages:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

This gives the browser instructions on how to control the page's dimensions and scaling.

The width=device-width part sets the width of the page to follow the screen-width of the device (which will vary depending on the device).

The initial-scale=1.0 part sets the initial zoom level when the page is first loaded by the browser.

#### What is a Media Query?



Media query is a CSS technique introduced in CSS3. It uses the @media rule to include a block of CSS properties only if a certain condition is true.

On the code defined below, If the browser window is 600px or smaller, the background color will be lightblue:

```
@media only screen and (max-width: 600px)
{
  body {
    background-color: lightblue;
  }
}
```

Using the "only" word on the syntax has no effect on modern browsers. You may write it like @media screen and (max-width: 600px). Don't forget to enclose the @media code with a curly brace.

- You have to set viewport to make @media work
- @media has to be enclosed within a curly brace
- You can set @media in <style> (internal CSS) or external CSS
- Do NOT use large fixed width elements. It may lead for horizontal scroll
- Pixel points like 992px and 600px are what we call "typical breakpoints"

### 5.3. JavaScript Introduction

JavaScript is the most popular scripting language in the world. JavaScript is the language for the web, for HTML, for servers, PCs, laptops, tablets, cell phones, and more. A scripting language is a lightweight programming language. JavaScript code can be inserted into any HTML page, and it can be executed by all types of web browsers. JavaScript is easy to learn.

The HTML DOM (Document Object Model) is the official W3C standard for accessing HTML elements. It is very common to use JavaScript to manipulate the DOM (to change the content of HTML elements).

#### Example

```
x = document.getElementById("demo"); //Find HTML element with id="demo"
```

```
x.innerHTML = "Hello JavaScript"; //Change the content of HTML element
```

**document.getElementById()** is one of the most commonly used HTML DOM methods.

You can also use JavaScript to:

- Delete HTML elements
- Create new HTML elements
- Copy HTML elements
- And more ...

#### Where?

You can write java script code either in heading part or body part. Before writing java script code you have to open <script> tag and close it at the end. You can place any number of scripts in an HTML document. And those can be both in <head> and <body>. But it is recommended to keep all your java script code (in a single page) together.

Java script code can be found alone as external file saved with extension file .js. If your code is found externally then you have to call (link) it with the current page like:

```
<script src="myScript.js"></script>
```

You are not expected to use <script> on external java script code.

- Java script is case sensitive (lower and upper case letters have different meaning)
- Java script code can be used either in head or body part.

- Use semicolon at the end of single line code.

### JS comment

JavaScript comments can be used to explain JavaScript code, just like other language's comment, text under comment will not be executed. So that it makes your code more readable and prevent execution, when testing alternative code. Single line and multi line comment are used.

#### Single Line Comments

Single line comments start with `//`. Any text starting with `//` up to the end of the line will be ignored by JavaScript (will not be executed).

#### Multi-line Comments

Multi-line comments start with `/*` and end with `*/`. Any text between `/*` and `*/` will be ignored by JavaScript.

```
<script>
//document.write("My Script");
document.write("Second line");
/*document.write("third line");
document.write("fourth line");*/
</script>
```

OUTPUT

Second line

The third and fourth lines are commented using multi line comment, which prevents all of them from being executed. So second line is the only to be displayed.

#### To display output

Java script can display output in different ways. "innerHTML", "document.write()" and "alert" are used to display output.

```
<script>
alert("Text with Box");
</script>
```

OUTPUT

Text with Box

OK

"alert" is used to display message or text on a message box.

```
<script>
document.write("text on screen");
</script>
```

OUTPUT

text on screen

"document.write()" is used to display text on your browser. Do not use this function except testing java script.



```
<body>
<h1 id="try"></h1>
<script>
document.getElementById("try").innerHTML="Text in heading";
</script>
</body>
```

OUT  
PUT

## Text in heading

On the above code <h1> has no element and it has id with the name "try". In the java script code, document.getElementById() is used to access any HTML element with its id. And we use the id "try" to access <h1>. innerHTML is used to manage the content of tag with that id. And "text in heading" is assigned for <h1> and it is displayed.

```
<body>
<h1 id="try">this is heading text</h1>
<script>
var x;
x=document.getElementById("try").innerHTML;
</script>
</body>
```

led "try" to a

A JavaScript code under a function will execute only when some event calls it. Function is defined with the "function" keyword, followed by a name, followed by parentheses ().

function try1()

Function names can contain letters, digits, underscores, and dollar signs. The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...)

The code to be executed, by the function, is placed inside curly brackets: {} Most often, JavaScript code is written to be executed when an event occurs, like when the user clicks a button. If we put JavaScript code inside a function, we can call that function when an event occurs.

Don't forget to use () while calling a function without parameter. And use () with variables or values to call function with parameter.

The following events can be used to call the function.

Event	Use
onClick	Function called when clicked on element
onDbClick	Function called when double clicked on element
onKeyPress	Function called when enter key pressed on element
onKeyUp	Function called when any key pressed on element
onMouseOver	Function called when mouse moves over element

```

<body>
<h1 id="sample" ></h1>
<button onClick="try1()" >click</button>
<script>
function try1()
{
    document.getElementById("sample").innerHTML="Heading text";
}
</script>
</body>

```

On the code above, a button called "click" will display on the browser. Then when you click on the button, "Heading text" will display above the button with <h1> property. This happens because, when the user clicks on the button, the function "try1" is called and code under try1 will execute. The code orders to change the content of tag with id "sample" which is <h1>, so the text will display with <h1>.

As we discussed earlier, code under function executes only if they are called. Unless they won't execute and their output cannot be shown on browser. In order to call functions, you have to use events, as placed on the above table (they are not all).

```

<body>
<h1 id="sample" ></h1>
<button onClick='try1("Heading text")'>click</button>
<script>
function try1(a)
{
    document.getElementById("sample").innerHTML=a;
}
</script>
</body>

```

expressions (z=x+y). In java script, variables should declare using the key word "var". Variable can have short names (like x, y and z) or more descriptive names (age, ID, FullName).

#### Example

```

var
var
var a=x + y1;

```

```

x=2;
y1=3;

```

While using variables in java script, we have to consider the following:

- Variable names must begin with a letter or \$ or \_
- Variable names are case sensitive (y and Y are different variables)
- Variable names can contain letters, digits, underscores, and dollar signs.
- Reserved words (like JavaScript keywords) cannot be used as names

It is possible to declare and assign value to a variable at different line. Or you can do both at a time.

```

var x; //this is declaration
x=5;    // this is assignment
or
var x=5    // both together

```

You can declare many variables in one statement. Just start the statement with var and separate the variables by comma:

```

var name="abc", age=30, id="1232";

```



In computer programs, variables are often declared without a value. The value can be something that has to be calculated, or something that will be provided later, like user input. Variable declared without a value will have the value **undefined**. The variable *x* in the previous declaration will have value *undefined* until it assigned new value.

If you re-declare a JavaScript variable, it will not lose its value.

### Operator

Operators are used to do same activity as you know in other programming languages. There are different operators used in programming and scripting languages. Arithmetic, assignment, string and comparison operators are discussed here.

#### Arithmetic operators

```
<script>
var x=4;
var y=5;
document.write(x + y);      //9
document.write(x++);        //4
document.write(x);          //5
document.write(x % y);      //0
document.write(++y);        //6
</script>
```

Second line display 4 because the operator used is post increment, means it increases after that line of code is executed. The fourth line displays 0 because it returns the remainder of the two numbers division.

#### Assignment operators

```
<script>
var x=4;
var y=5;
document.write(x);          //4
document.write(x+=y);       //9
document.write(x);          //9
</script>
```

Third line display 9 because on the second line, the value of *x* is changed (*x += y* means *x=x + y*) so that *x* and *y* are added and assigned for *x*. it is the same for all other arithmetic operators like */*, *\**, *%* and *-*. Means, */=*, *\*=*, *%=* and *-=* can be used as we see on the above example for *+=*.

#### Comparison operators

```

<script>
var x=4;
var y="4";
document.write(x==y);           //true
document.write(x===y);          //false
</script>

```

Here the first output displays "true" because "=" is used to check if the two values are equal and it is true that the two values are equal. Second line is false because it checks if the two variables have same value and type.

In java script, the type of data that you are using is determined after value is assigned for the variable. Since there are no different data types before value is assigned, the type of data is unknown until it is assigned. So there are two things "type" and "value" for every variable result.

The example shows that x has value 4 and its type is integer. And y has value 5 and type is string because it is inside double quotation. So if we compare them using "=", since it compares only their value, the result is true. To the reverse if they are compared using "===", since it compares both value and type, the result becomes false.

### String operator

```

<script>
var x=4,y="4";
var txt1="My", txt2=" Note";
document.write(x + y);           //44
txt1 += txt2;
document.write(txt1);             //My Note
</script>

```

Values inside " are considered as string, so that if they are with integer value, the value will be concatenated result. "44" is displayed because y has string value and "+" is used as concatenate for string values. The second line displays the result of "txt1" which is changed by "+" operator before it displays.

In java script, if number and string are together, then they all will treat as a string. And java script evaluates operation from left to right.

```

<script>
document.write(4 + 5 + "love");   //9love
document.write("love" + 4 + 5);   //love45

```



Since execution starts from left to right, and 4 and 5 are numbers, they will be added and concatenated with the string "love". On the second line it starts with string so all treat as string and concatenate to each other.

### String methods

Methods	Use	How?
length	Returns string length	var x=txt.length;
indexOf	Returns the position of word	var y=txt.indexOf("home");
search	Same with indexOf	var y=txt.search("home");
replace	Used to replace word with other	var y=txt.replace("home","house")
toUpperCase	Changes lower case to upper	var z=txt.toUpperCase( );

```
<script>
var txt="this is my home";
document.write(txt.length);           //15
document.write(txt.indexOf("home"));  //11
document.write(txt.replace("home","house");//this is my house
document.write(txt.toUpperCase());    //THIS IS MY HOME
</script>
```

for

example if indexOf("home") is given and if there is no "home" in the given string then -1 will be returned.

### Change HTML Styles (CSS)

Changing the style of an HTML element is a variant of changing an HTML attribute. You can change HTML elements using java script code.

### Example

```
<body>
<h1 id="sample" >sample text</h1>
<button onClick='try1()'>click</button>
<script>
function try1()
{
var txt=document.getElementById("sample");
txt.style.fontSize="3em";
txt.style.color="red";
}
</script>
</body>
```

When you click the button, on the above code, the function "try1" will be called and it will change the font size and color of a text with id "sample".

## Conditional Statements

Conditional statements are used to perform different actions based on different conditions. Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

### If Statement

```
<body>
<button onClick='try1()'>click</button>
<script>
function try1()
{
var x=prompt("enter your age");
if(x>20) alert("you are legal");
else if(x>18) alert("you have to wait");
else alert("you are kid");
}
</script>
</body>
```

When you are using conditional statement, you have to know that only one of those conditions will work. If there is no "else" you may have no condition to run. On the above example the user is asked to enter his/her age using the "prompt" function. After the user enters age, value will assign for variable "x" and with the conditional statement, using "if", "else if" and "else", x is checked whether it is legal age or not. The first condition checks "if(x>20)" this will work only if x has value greater than 20. If this is true, then the others won't be checked. But if it is false, then the next condition "else if(x>18)" will be checked. Just as the previous case if it is true then code under it will run and no other condition be checked. If the above two are not true then the "else" case will run.

On the example above "alert" is used to display the message using message box.

Please do not forget that java script is case sensitive and do not use upper case for if, else if and else. And do not put semicolon (;) at the end of conditional statement (if, else if and else).

## JavaScript Loops

In programming loop is used to do activities repeatedly. While you interpret the real world to the computer, activities which are done repeatedly will be managed by the use of loop. It will help us to iterate things as long as we want them.

### For Loop

for loop has the following syntax:

- Semicolon (;) is used to separate the three statements in for loop.
- Initiation is used to tell the loop where to start from.
- Condition is checked always before loop's code is executed. If condition is false, code won't be executed.
- Increment or decrement is used to increase or decrease the initial value. Like x++ or x--

On the code above, "i=0" is initial state, means "i" will start from 0. The condition checks if "i" is less t



```
<body>
<script>
var i;
for(i=0;i<5;i++)
document.write(i); </script>
</body>
```

han 5, if yes it will go to execute the code under for loop. When "i" less than 5 is false then, the code will stop to execute and go to another line after for loop. The increment orders value to increase with 1 every time. Increment is executed every time after the block code (code under for loop) executes.

### JavaScript While Loop

The while loop loops through a block of code as long as a specified condition is true. When using while loop, initial value has to be set before the while loop. And increment/decrement must be there within the block code.

```
<body>
<script>
var i=0;
while(i<5)
{
document.write(i);
i++;
}
</script>
</body>
```

Code above shows while loop. Initial value is set before while loop "i=0". The block code (which is enclosed by { }) executes only if the condition (i<5) is true. When the block code execute, increment will execute also and iteration continue to condition until it becomes false.

### The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, and then it will repeat the loop as long as the condition is true.

### The Break Statement

The break statement can be used to jump out of a loop. The break statement breaks the loop and continues executing the code after the loop (if any):

### The Continue Statement

The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

```
<script>
var i=0;
while(i<5)
{
document.write(i);
    if(i==2)
        break;
i++;
}
</script>
```

OUTPUT

012

If there is no break on this code, 01234 was display. As you see code written above in the while loop, it checks if i is equal to 2 and if that satisfies break will execute. When break executes the iteration will stop executing and execution will continue to the next code (after the loop). That is the reason why 3 and 4 are not displayed on the output.

```
<script>
var i;
for(i=0;i<5;i++)
{
if(i==2)
continue;
document.write(i);
}
</script>
```

OUTPUT

0134

Here continue is used and the loop doesn't break at all but it backs to increment without executing code when "i" is equal to 2. For that matter 2 is not displayed on the output. Since there is continue after "if(i==2)", when i becomes 2 continue executes and that means the code will immediately go to increment instead of "document.write(i)" so 2 will not be displayed.

### Lab 1: Java script and HTML

```
<html>
<body>
<p>Click the button to display your name</p>
<button onclick="myFunction()">Try it</button>
<p id="demo"></p>
<script>
function myFunction()
{
var r=prompt("enter your name");
document.getElementById("demo").innerHTML=r;
}
</script>
</body>
</html>
```

OUT  
PUT

Click the button to display your name

Try it



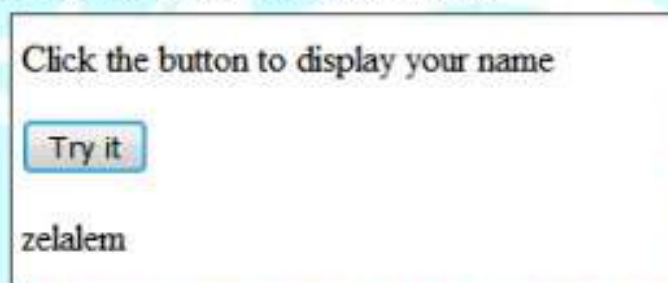


enter your name

zelalem

OK Cancel

Enter your name and say ok, then this will display.



Click the button to display your name

Try it

zelalem

"prompt" is used to accept input from the user. After having name from the user the code displays that on the HTML tag with the id "demo". `<p id="demo">` has no element before name assigned for it.

### Lab 2: Confirm box

```
<html>
<body>
<button onclick="myFunction()">close</button>
<script>
function myFunction()
{
var r=confirm("are you sure");
if(r==true)
window.close();
else
alert("window is \n not closing");
}
</script>
</body>
</html>
```

"confirm" is used to take input from the user as yes (true) or cancel (false). The code above takes user confirmation and if the user says ok it will close the current window opened. Using "confirm" box, if the

user press on yes, then it returns "true" else it returns "false". On the code above when the user press on "close" button, confirm box appears and asks if the user is sure, then if the user click on "yes", which is "true" for the code, the window will close by the code "window.close()". If the user press on "cancel" which is "false" for the code, "window is not closing" message will appear by alert box. When this message appears, it displays in two lines because "\n" is used in between.

## 5.4. What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open-source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use

PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages. PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

### PHP is an amazing and popular language!

It is powerful enough to be at the core of the biggest blogging system on the web (WordPress)! It is deep enough to run the largest social network (Facebook)! It is also easy enough to be a beginner's first server-side language!

### What is a PHP File?

- PHP files can contain text, HTML, CSS, JavaScript, and PHP code
- PHP code are executed on the server, and the result is returned to the browser as plain HTML
- PHP files have extension ".php"

### What Can PHP Do?

- PHP can generate dynamic page content
- PHP can create, open, read, write, delete, and close files on the server
- PHP can collect form data
- PHP can send and receive cookies
- PHP can add, delete, modify data in your database
- PHP can be used to control user-access
- PHP can encrypt data

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.



## Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free.
- PHP is easy to learn and runs efficiently on the server side

### Basic PHP Syntax

A PHP script can be placed anywhere in the document. A PHP script starts with `<?php` and ends with `?>`:

```
<?php
// PHP code goes here
?>
```

The default file extension for PHP files is ".php". A PHP file normally contains HTML tags, and some PHP scripting code. Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

#### Example

```
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

**Note:** PHP statements end with a semicolon (;).

### Comments in PHP

A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is looking at the code.

#### Example

```
<html>
<body>
<?php
// This is a single-line comment
# This is also a single-line comment
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
```

```
// You can also use comments to leave out parts of a code line
```

```
$x = 5 /* + 15 */ + 5;  
echo $x;  
>  
</body>  
</html>
```

### PHP Case Sensitivity

In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive. In the example below, all three echo statements below are legal (and equal):

#### Example

```
<html>  
<body>  
  
<?php  
ECHO "Hello World!<br>";  
echo "Hello World!<br>";  
EcHo "Hello World!<br>";  
>  
</body>  
</html>
```

However, all variable names are case-sensitive. In the example below, only the first statement will display the value of the \$color variable (this is because \$color, \$COLOR, and \$coLOR are treated as three different variables):

#### Example

```
<html>  
<body>  
  
<?php  
$color = "red";  
echo "My car is " . $color . "<br>";  
echo "My house is " . $COLOR . "<br>";  
echo "My boat is " . $coLOR . "<br>";  
>  
</body>  
</html>
```

### Creating (Declaring) PHP Variables

In PHP, a variable starts with the \$ sign, followed by the name of the variable:

#### Example

```
<?php  
$txt = "Hello world!";  
$x = 5;  
$y = 10.5;  
>
```



After the execution of the statements above, the variable \$txt will hold the value **Hello world!**, the variable \$x will hold the value **5**, and the variable \$y will hold the value **10.5**.

**Note:** Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

## PHP Variables

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total\_volume).

Rules for PHP variables:

- A variable starts with the \$ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_)
- Variable names are case-sensitive (\$age and \$AGE are two different variables)

Remember that PHP variable names are case-sensitive!

## Output Variables

The PHP echo statement is often used to output data to the screen. The following example will show how to output text and a variable:

### Example

```
<?php
$txt = "becom.com";
echo "I love $txt!";
?>
```

The following example will produce the same output as the example above:

### Example

```
<?php
$txt = "metitech.com";
echo "I love " . $txt . "!";
?>
```

Note: "." Indicates the concatenation

The output is = I love metitech.com

### Example

```
<?php
$x = 5;
$y = 4;
```

```
echo $x + $y;  
?>
```

### PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value. In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

### PHP Variables Scope

In PHP, variables can be declared anywhere in the script. The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

### Global and Local Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

#### Example

```
<?php  
$x = 5; // global scope  
  
function myTest() {  
    // using x inside this function will generate an error  
    echo "<p>Variable x inside function is: $x</p>";  
}  
myTest();  
  
echo "<p>Variable x outside function is: $x</p>";  
?>
```

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

#### Example

```
<?php  
function myTest() {  
    $x = 5; // local scope  
    echo "<p>Variable x inside function is: $x</p>";  
}  
myTest();
```



```
// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

## PHP The global Keyword

The global keyword is used to access a global variable from within a function. To do this, use the global keyword before the variables (inside the function):

### Example

```
<?php
$x = 5;
$y = 10;

function myTest() {
    global $x, $y;
    $y = $x + $y;
}
myTest();
echo $y; // outputs 15
?>
```

PHP also stores all global variables in an array called `$GLOBALS[index]`. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

### Example

```
<?php
$x = 5;
$y = 10;

function myTest() {
    $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}
myTest();
echo $y; // outputs 15
?>
```

## The static Keyword

Normally, when a function is completed/ executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job. To do this, use the **static** keyword when you first declare the variable:

### Example

```
<?php
function myTest() {
    static $x = 0;
    echo $x;
    $x++;
}

myTest();
myTest();
myTest();
?>
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

### PHP echo and print Statements

echo and print are more or less the same. They are both used to output data to the screen.

The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

### Get the Length of a String

The PHP strlen() function returns the length of a string. The example below returns the length of the string "Hello world!":

#### Example

```
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

The output of the code above will be: 12.

### Count the Number of Words in a String

The PHP str\_word\_count() function counts the number of words in a string:

#### Example

```
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

The output of the code above will be: 2.

### Reverse a String



The PHP `strrev()` function reverses a string:

#### Example

```
<?php
echo strrev("Hello world!"); // outputs !dlrow olleH
?>
```

The output of the code above will be: !dlrow olleH.

#### Search for a Specific Text within a String

The PHP `strpos()` function searches for a specific text within a string. If a match is found, the function returns the character position of the first match. If no match is found, it will return `FALSE`. The example below searches for the text "world" in the string "Hello world!":

#### Example

```
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

The output of the code above will be: 6.

**Note:** The first character position in a string is 0 (not 1).

#### Replace Text within a String

The PHP `str_replace()` function replaces some characters with some other characters in a string.

The example below replaces the text "world" with "Dolly":

#### Example

```
<?php
echo str_replace("world", "Dolly", "Hello world!"); // outputs Hello
Dolly!
?>
```

The output of the code above will be: Hello Dolly!

### PHP Operators

Operators are used to perform operations on variables and values. PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators

- Logical operators
- String operators

## PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result
<code>==</code>	Equal	<code>\$x == \$y</code>	Returns true if \$x is equal to \$y
<code>===</code>	Identical	<code>\$x === \$y</code>	Returns true if \$x is equal to \$y, and they are of the same type
<code>!=</code>	Not equal	<code>\$x != \$y</code>	Returns true if \$x is not equal to \$y
<code>!==</code>	Not identical	<code>\$x !== \$y</code>	Returns true if \$x is not equal to \$y, or they are not of the same type

## PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
<code>.</code>	Concatenation	<code>\$txt1 . \$txt2</code>	Concatenation of \$txt1 and \$txt2
<code>.=</code>	Concatenation assignment	<code>\$txt1 .= \$txt2</code>	Appends \$txt2 to \$txt1

## PHP Conditional Statements

### Example

```
<?php
$t = date("H");

if ($t < "20") {
    echo "Have a good day!";
}
?>
```

### Example

```
<?php
$t = date("H");
if ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```



### Example

```
<?php
$t = date("H");

if ($t < "10") {
    echo "Have a good morning!";
} elseif ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

### PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this. In PHP, we have the following looping statements:

- **while** - loops through a block of code as long as the specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

### Example

```
<?php
$x = 1;
while($x <= 5) {
    echo "The number is: $x <br>";
    $x++;
}
?>
```

### Example

```
<?php
$x = 1;
do {
    echo "The number is: $x <br>";
    $x++;
} while ($x <= 5);
?>
```

### Example

```
<?php
$x = 6;
do {
    echo "The number is: $x <br>";
    $x++;
} while ($x <= 5);
?>
```

### The PHP for Loop

The for loop is used when you know in advance how many times the script should run.

#### Example

```
<?php
for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x <br>";
}
?>
```

### PHP foreach Loop

#### Example

```
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
    echo "$value <br>";
}
?>
```

### Function in PHP

A user defined function declaration starts with the word "function":

#### Syntax

```
function functionName() {
    code to be executed;
}
```

**Note:** A function name can start with a letter or underscore (not a number).

**Tip:** Give the function a name that reflects what the function does!

Function names are NOT case-sensitive.

In the example below, we create a function named "writeMsg()". The opening curly brace ( { ) indicates the beginning of the function code and the closing curly brace ( } ) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:



## Example

```
<?php
function writeMsg() {
    echo "Hello world!";
}

writeMsg(); // call the function
?>
```

## PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (\$fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

## Example

```
<?php
function familyName($fname) {
    echo "$fname Refsnes.<br>";
}

familyName("Hani");
familyName("Hege");
familyName("Tame");
familyName("Kai Jim");
familyName("Borge");
?>
```

The following example has a function with two arguments (\$fname and \$year):

## Example

```
<?php
function familyName($fname, $year) {
    echo "$fname Refsnes. Born in $year <br>";
}

familyName("Hege", "1975");
familyName("Tame", "1978");
familyName("Kai Jim", "1983");
?>
```

## PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

### Example

```
<?php
function setHeight($minheight = 50) {
    echo "The height is : $minheight <br>";
}
setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

### PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

### Example

```
<html>
<body>
<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method. To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
</body>
</html>
```

The output could be something like this:

```
Welcome John
Your email address is john.doe@example.com
```

The same result could also be achieved using the HTTP GET method:

### Example

```
<html>
<body>
```



```
<form action="welcome_get.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>

</body>
</html>
```

and "welcome\_get.php" looks like this:

```
<html>
<body>

Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>

</body>
</html>
```

The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.

## GET vs. POST

Both GET and POST are treated as `$_GET` and `$_POST`. These are super global, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special. `$_GET` is an array of variables passed to the current script via the URL parameters. `$_POST` is an array of variables passed to the current script via the HTTP POST method.

### When to use GET?

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases. GET may be used for sending non-sensitive data. **Note:** GET should NEVER be used for sending passwords or other sensitive information!

### When to use POST?

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send. Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server. However, because the variables are not displayed in the URL, it is not possible to bookmark the page.



## PHP include and require Statements

The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement. Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement. The include and require statements are identical, except upon failure:

- require will produce a fatal error (E\_COMPILE\_ERROR) and stop the script
- include will only produce a warning (E\_WARNING) and the script will continue

So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement. Otherwise, in case of Frame Work, CMS, or a complex PHP application coding, always use the require statement to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.

Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

### Syntax

include *'filename';*

or  
require *'filename';*

The require statement is also used to include a file into the PHP code. However, there is one big difference between include and require; when a file is included with the include statement and PHP cannot find it, the script will continue to execute. If we do the same example using the **require** statement, the echo statement will not be executed because the script execution dies after the require statement returned a fatal error.

## PHP Session

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser. So, Session variables hold information about one single user, and are available to all pages in one application. A session is a way to store information (in variables) to be used across multiple pages. Unlike a cookie, the information is not stored on the user's computer.



### Example

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

**Note:** The `session_start()` function must be the very first thing in your document. Before any HTML tags.

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (`session_start()`). Also notice that all session variable values are stored in the global `$_SESSION` variable:

### Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] . "<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] . ".";
?>

</body>
</html>
```

Favorite color is green.  
Favorite animal is cat.

The output shows us that session variable can be accessed on another page.

**How does it work? How does it know it's me?**

Most sessions set a user-key on the user's computer that looks something like this: 765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another page, it scans the computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

### Destroy a PHP Session

To remove all global session variables and destroy the session, use `session_unset()` and `session_destroy()`:

#### Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```

#### Note:

`session_unset()` is not used to destroy session, but to remove all assigned session variables.

## 5.5. PHP MySQL Database

With PHP, you can connect to and manipulate databases. MySQL is the most popular database system used with PHP.

### What is MySQL?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.



## Database Queries

A query is a question or a request. We can query a database for specific information and have a recordset returned. Look at the following query (using standard SQL):

```
SELECT LastName FROM Employees
```

The query above selects all the data in the "LastName" column from the "Employees" table.

The following table shows us how to connect to the server "localhost" using php code.

### Example

```
<?php  
  
$conn = mysqli_connect("localhost", "root", "", "dbname");  
  
// Check connection  
if (!$conn) {  
    die("Connection failed: " . mysqli_connect_error());  
}  
echo "Connected successfully";  
?>
```

### Description

The above connection is for MySQLi (improved).

Mysqli\_connect is used to connect with the server

Localhost is the name of server

Root is default username of your server

"" (which is empty) is the password of the server

Dbname is the name of database that is found in the server and i am to use

After you connect to your server and finalize activities, it is recommended to close the connection. To do that use the following code:

```
mysqli_close($conn);
```

## Insert Data

After a database and a table have been created, we can start adding data in them. Here are some syntax rules to follow:

- The SQL query must be quoted in PHP
- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

The INSERT INTO statement is used to add new records to a MySQL table:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

Let us assume that we have a database with the name "dbreg". And table named "Student" with four columns: "id", "firstname", "lastname" and "age". Now, let us fill the table with data.

The following examples add a new record to the "Student" table:

### Example (for MySQLi)

```
<?php
$conn = mysqli_connect("localhost", "root", "", "dbreg");
if (!$conn) { // Check connection
    die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO Student (id,firstname, lastname, age)
VALUES ('1123','John', 'Doe', 21)";

if (mysqli_query($conn, $sql)) {
    echo "<script>alert('New record created successfully');</script>";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```



## Select Data

The SELECT statement is used to select data from one or more tables:

```
SELECT column_name(s) FROM table_name
```

Or we can use the \* character to select ALL columns from a table:

```
SELECT * FROM table_name
```

The following example selects the id, firstname and lastname columns from the Student table and displays it on the page:

### Example (MySQLi)

```
<?php
$conn = mysqli_connect("localhost", "root", "", "dbreg");
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
$sql = "SELECT id, firstname, lastname FROM Student";
$result = mysqli_query($conn, $sql);
while($row = mysqli_fetch_assoc($result))
{
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
    $row["lastname"]. "<br>";
}
mysqli_close($conn);
?>
```

Code lines to explain from the example above:

First, we set up an SQL query that selects the id, firstname and lastname columns from the Student table. The next line of code runs the query and puts the resulting data into a variable called \$result. Dont forget that mysqli\_query is used to process the query written.

The function fetch\_assoc() puts all the results into an associative array that we can loop through. The while() loop loops through the result set and outputs the data from the id, firstname and lastname columns.

You can also put the result in an HTML table:

### Example

```

<?php
$conn = mysqli_connect("localhost", "root", "", "dbreg");
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
$sql = "SELECT id, firstname, lastname FROM Student";
$result = mysqli_query($conn, $sql);
echo "<table><tr><th>ID</th><th>Name</th></tr>";
while($row = mysqli_fetch_assoc($result))
{
    echo "<tr><td>".$row["id"]."</td><td>".$row["firstname"]."
    ".$row["lastname"]."</td></tr>";
}
echo "</table>";

mysqli_close($conn);
?>

```

### Delete Data

The DELETE statement is used to delete records from a table:

```

DELETE FROM table_name
WHERE some_column = some_value

```

**Notice the WHERE clause in the DELETE syntax:** The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

Let's look at the "Student" table:

Id	Firstname	lastname	age
1122	John	Doe	21
2213	Mary	Moe	20
1321	Julie	Dooley	21

The following examples delete the record with id=1321 in the "Student" table:

### Example (MySQLi Procedural)

```

<?php
$conn = mysqli_connect("localhost", "root", "", "dbreg");

```



```

if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=1321";

if (mysqli_query($conn, $sql))
    echo "<script>alert('Record deleted successfully');</script>";
else
    echo "Error deleting record: " . mysqli_error($conn);
mysqli_close($conn);
?>

```

After the record is deleted, the table will look like this:

Id	Firstname	lastname	age
1122	John	Doe	21
2213	Mary	Moe	20

## Update Data

The UPDATE statement is used to update existing records in a table:

```

UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value

```

**Notice the WHERE clause in the UPDATE syntax:** The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

Let's look at the "Student" table:

Id	firstname	lastname	age
1122	John	Doe	21
2213	Mary	Moe	20

The following examples update the record with id=1122 in the "Student" table:

### Example (MySQLi)

```
<?php
$conn = mysqli_connect("localhost", "root", "", "dbreg");
// Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
$sql = "UPDATE Student SET lastname='Moe' WHERE id=1122";

if (mysqli_query($conn, $sql))
    echo "<script>alert('Record updated successfully');</script>";
else
    echo "Error updating record: " . mysqli_error($conn);
mysqli_close($conn);
?>
```

After the record is updated, the table will look like this:

Id	Firstname	lastname	age
1122	John	Moe	21
2213	Mary	Moe	20

### Limit Data Selections from a MySQL Database

MySQL provides a LIMIT clause that is used to specify the number of records to return. The LIMIT clause makes it easy to code multi page results or pagination with SQL, and is very useful on large tables. Returning a large number of records can impact on performance. Assume we wish to select all records from 1 - 30 (inclusive) from a table called "Orders". The SQL query would then look like this:

```
$sql = "SELECT * FROM Orders LIMIT 30";
```

When the SQL query above is run, it will return the first 30 records. What if we want to select records 16 - 25 (inclusive)? Mysql also provides a way to handle this: by using OFFSET. The SQL query below says "return only 10 records, start on record 16 (OFFSET 15)":

```
$sql = "SELECT * FROM Orders LIMIT 10 OFFSET 15";
```

You could also use a shorter syntax to achieve the same result:



```
$sql = "SELECT * FROM Orders LIMIT 15, 10";
```

Notice that the numbers are reversed when you use a comma.

### Review questions

1. What is HTML?
  - a) HTML describes the structure of a webpage
  - b) HTML is the standard markup language mainly used to create web pages
  - c) HTML consists of a set of elements that helps the browser how to view the content
  - d) All of the mentioned
2. What is the correct syntax of doctype in HTML5?
  - a) </doctype html>
  - b) <doctype html>
  - c) <doctype html!>
  - d) <!doctype html>
3. Which of the following tag is used for inserting the largest heading in HTML?
  - a) head
  - b) <h1>
  - c) <h6>
  - d) heading
4. In which part of the HTML metadata is contained?
  - a) head tag
  - b) title tag
  - c) html tag
  - d) body tag
5. Which element is used to get highlighted text in HTML5?
  - a) <u>
  - b) <mark>
  - c) <highlight>
  - d) <b>
6. Which element is used for or styling HTML5 layout?
  - a) CSS
  - b) jQuery
  - c) JavaScript
  - d) PHP
7. Which HTML tag is used to insert an image?
  - a) <img url="htmllogo.jpg" />
  - b) <img alt="htmllogo.jpg" />
  - c) 
  - d) <img link="htmllogo.jpg" />
8. In HTML, which attribute is used to create a link that opens in a new window tab?
  - a) src="\_blank"
  - b) alt="\_blank"
  - c) target="\_self"
  - d) target="\_blank"
9. Which of the following HTML tag is used to create an unordered list?
  - a) <ol>

- b) <ul>
  - c) <li>
  - d) <ll>
10. What is the work of <address> element in HTML5?
- a) contains IP address
  - b) contains home address
  - c) contains url
  - d) contains contact details for author
11. Which of the following tag is used to create a text area in HTML Form?
- a) <textarea> </textarea>
  - b) <text></text>
  - c) <input type="text" />
  - d) <input type="textarea" />
12. Which attribute is not essential under <iframe>?
- a) frameborder
  - b) width
  - c) height
  - d) src
13. Which element is used to define a discrete unit of content such as a blogpost, comment, and so on?
- a) section
  - b) class
  - c) article
  - d) media
14. Which of the following tag is used to embed css in html page?
- a) <css>
  - b) <!DOCTYPE html>
  - c) <script>
  - d) <style>
15. Which of the following CSS property is used to make the text bold?
- a) text-decoration: bold
  - b) font-weight: bold
  - c) font-style: bold
  - d) text-align: bold
16. Which of the following CSS style property is used to specify an italic text?
- a) style
  - b) font
  - c) font-style
  - d) @font-face
17. Which of the following function defines a linear gradient as a CSS image?
- a) gradient()
  - b) linear-gradient()
  - c) grayscale()
  - d) image()
18. Which of the following CSS property defines the different properties of all four sides of an element's border in a single declaration?
- a) border-collapse
  - b) border-width



- c) padding
- d) border
- 19. Which of the following CSS property specifies the look and design of an outline?
  - a) outline-style
  - b) outline-format
  - c) outline-font
  - d) none of the mentioned
- 20. Which of the following CSS property sets the shadow for a box element?
  - a) set-shadow
  - b) box-shadow
  - c) shadow
  - d) canvas-shadow

**Answers**

- 1. D
- 2. D
- 3. B
- 4. A
- 5. B
- 6. A
- 7. C
- 8. D
- 9. B
- 10. D
- 11. A
- 12. A
- 13. C
- 14. D
- 15. B
- 16. C
- 17. B
- 18. B
- 19. A
- 20. B

## Reference

1. Walter Savitch, Problem Solving with C++, 10th Edition, 2018
2. Richard L. Halterman: "Fundamentals of Computing with C++", 2019.
3. Richard L. Halterman, "Fundamentals of C++ Programming", School of Computing Southern
4. Narasimha Karumanchi: "Data Structures And Algorithms Made Easy", 5<sup>th</sup> Edition Nara, 2017
5. Data Structures and Algorithm Analysis in C++, 2<sup>nd</sup> edition Mark Allen Weiss, Pearson Education, 2014
6. Goodrich (Author), Tamassia (Author), Goldwasser, "Data Structures & Algorithms in Java, 6th Ed.", Wiley, 2014
7. Anany Levitin: "Introduction to the Design and Analysis of Algorithms", 3rd Edition, Cognella Academic Publishing, January 2012.
8. Anany Levitin, Introduction to the Design and Analysis of Algorithms, 3rd ed. Pearson Education , February 2017
9. Vishwajit K Barbudhe (Author), Shraddha N Zanjat (Author), Bhavana S Karmore, Computer Algorithms, Introduction to Design and Analysis , LAP Lambert Academic Publishing, 2020
- 10.
11. Elmasri, R., & Navathe, S. (2017). *Fundamentals of database systems* (7<sup>th</sup> Edition). Pearson.
12. Osama Mustafa, Robert P. Lockard. (2019). Oracle Database Application Security, Apress, Berkeley, CA.
13. Raghu Ramakrishnan, Johannes Gehrke. Database Management Systems, McGraw-Hill; 3rd edition, 2002
14. Ian Sommerville , SOFTWARE ENGINEERING, Ninth Edition, 2011.
15. Martina Seidl, Marion Scholz, Christian Huemer, Gerti Kappel. UML @ Classroom: An
16. Introduction to Object-Oriented Modeling. Springer International Publishing AG, 2012.
17. HTML, CSS, and JavaScript All in One, Sams Teach Yourself (3rd Edition), 2012
18. Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics 5th Edition, 2018
19. Shelly et al., HTML5 and CSS: Comprehensive 7th Edition, Thomson Course Tech. Jennifer Niederst,