CHAPTER-1 (REVISION TOUR)

Object Oriented Programming System (OOPS): Object Oriented Programming is usually defined in terms of several (unrelated) techniques (eg, inheritance, encapsulation, polymorphism), but there is one characteristic that all OOP programs must have -- **group related data and functions together** in a *class*. Before OOP, data was grouped in data structures and the functions where grouped together in modules. The advantages of grouping them together in a class are not immediately obvious, but the advantages have turned out to be substantial.

Advantages of OOP:

Re-use of code: Linking of code to objects and explicit specification of relations between objects allows related objects to share code. Encapsulation allows class definitions to be re-used in other applications.

Easy for maintenance: The concepts such as encapsulation, data abstraction allow for very clean design and its easy to debug.

Easy to add new module: OOP facilitate to add new modules easily.

OOPS terminology and features

1. Data Abstraction
2. Data Encapsulation
3. Modularity
4. Inheritance
5. Polymorphism

Data Abstraction: Data Abstraction refers to the act of representing essential features without including the background details or explanations.

Encapsulation: The wrapping up of data and associated functions into a single unit (class) is known as encapsulation. It is just a way to implement data abstraction.

Modularity: Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules.

Inheritance: Inheritance is the capability of a class to inherit capabilities&properties from other class.

Role of Inheritance in OOPS

1. Its capability to express the inheritance relationship which makes it ensures the closeness with the real world models.
2. Reusability:- Inheritance allows the addition of additional features to an existing class without modifying it.
3. Transitive nature:- If a class A inherits properties of another class B, then all subclasses of A will automatically inherit the properties of B.

Polymorphism:- Polymorphism is the ability for a message or data to be processed in more than one form. (One name having many form and behaving differently in different situations).

Polymorphism is a property by which the same message can be sent to objects of several different classes.

Data Types:

1. Fundamental types (atomic)
2. Derived types

Fundamental types:- Fundamental data types are those that are not composed of other data types.

Five fundamental data types are there

1. int Data type for Integers
2. char Data types for Characters
3. float Data types for Floating point numbers

4. double Data types for Double precision floating point numbers
5. void Data types for Empty set of values and non-returning functions.

Data Type Modifiers

1.signed **2.** unsigned **3.** long **4.** short

Integer Types

Types	Approximate Size (bytes)	Minimal Range
Short unsigned short singed short	2 2 2	–32768 to 32768 0 to 65,535 Same as short
int unsigned int singed int	2 2 2	–32768 to 32768 0 to 65,535 Same as int
Long unsigned long signed long	4 4 4	– 2,147,483,648 to 2,147,483,648 0 to 4,294,967,295 Same as long

Character Types

Types	Approximate Size (in bytes)	Minimal Range
char unsigned char singed char	1 1 1	–128 to 127 0 to 255 Same as char

Floating point Types

Types	Approximate Size (in bytes)	Minimal Range	Digits of precision
float unsigned float singed float	4 8 10	3.4×10^{-38} to 3.4×10^{38} 1.7×10^{-308} to 1.7×10^{308} 3.4×10^{-4932} to 3.4×10^{4932}	7 15 19

Derived Data Types

1.Arrays **2.** Functions **3**. Pointers **4**. Reference **5**. Constant

Arrays:- Arrays refer to a named list of a finite number of similar data elements.

Function:- A function is a named part of a program that can be invoked from other parts of the program as often needed.

Pointer: A pointer is a variable that holds a memory address.

A reference is an alternative name for an object.

Constant:- A keyword const can be added to the declaration of an object to make that object a constant rather than a variable. A constant must be initialized at the time of declaration.

User-Defined Derived Data Types

1.Class **2**. Structure **3**. Union **4.** Enumeration

Class:- A class represents a group of similar objects. To represent classes in C++, it offers user-defined data types called **class**. Once a *class* has been defined in C++, objects belonging to that class can easily be created. A class bears the same relationship to an object as a type bears to a variable.

Structure:- A structure is a collection of variables (of different data types) referenced under one name, providing of convenient means of keeping related information together.

Union:- A union is a memory location that is shared by two or more different variables generally of different types at different times. Defining a union is similar to defining a structure.

Enumeration:- An alternative method for naming integer constants is often more convenient than const. This can be achieved by creating enumeration used keyword enum.

I/O Operators:

1. Input Operator ‘>>’

2. Output Operator ‘<<’

Cascading of I/O operators:- The multiple use of input or output operators (‘<<’ or ‘>>’) in one statement is called cascading of I/O operators.

Operators:-

1. Arithmetic Operators
2. Increment/Decrement Operators
3. Relational Operators
4. Logical Operators
   a. AND Operator (&&)
   b. NOT Operator (!)
   c. OR Operator (||)
5. Conditional Operator (?:)

Precedence of Operators

++ (post increment), – – (post decrement)	High
++ (pre increment), – – (pre decrement), sizeof, ! (not), – (unary minus), + (unary plus)	↑
* (multiplication), / (division), % (modulus)	
+ (add), – (subtract)	
< (less than), <= (less than or equal), > (greater than), >= (greater than or equal)	
= = (equal), ! = (not equal)	
&& (logical AND)	
|| (logical OR)	
?: (conditional expression)	
= (simple assignment) and other assignment operators (arithmetic assignment operators)	
Comma Operator	Low

Type Conversion:- The process of converting one predefined type into another is called type conversion

1. Implicit type conversion
2. Explicit type conversion

Implicit type conversion:- An implicit type conversion is a conversion performed by the compiler without programmer’s intervention. An implicit conversion is applied generally whenever differing data types are intermixed in an expression, so as not to lose information.

Explicit type conversion:- An explicit type conversion is user-defined that forces an expression to be of specific type.

The explicit conversion of an operand to a specific type is called type casting.

C++ short hands:- C++ offers special short hands that simplify the coding of a certain type of assignment statement. The general form of C++ short hands is

var = **var** *operator* **expression**	is same as	**Var** *operator* = **expression**
x = *x* + 1	is same as	*x* + = 1
x = *x* – 10	is same as	*x* – = 10
x = *x* * 2	is same as	*x* * = 2
x = *x* / 2	is same as	*x* / = 2
x = *x* % *z*	is same as	*x* % = *z*

FLOW OF CONTROL

1. Sequential
2. Selection
3. Iteration
4. Jump

Sequential flow of control:- Statements are executed one by one in a linear fashion one after the another. Normally the statements are executed sequentially as written in the program.

Selection flow of control:- Sometime the operational flow of control of a program has to be altered depending upon a condition. The statements that allow to do so, are known as selection statements. C++ provides two types of selection statements:

1. If
2. Switch

Iteration Statement:- The iteration statements allow a set of instructions to be performed separately until a certain condition is fulfilled. The iteration statements are also loops or looping statements.

1. for
2. while } (Entry controlled loop)
3. do-while (Exit controlled loop)

Nested Loop:- A loop may contain another loop in its body. This form of a loop is called nested loop. But in a nested loop, the inner loop must terminate before the outer loop.

Jump Statements:- The jump statements unconditionally transfer program control within a function. C++ has four statements that perform an unconditional branch **return, goto, break and continue.**

Break Statement:- The break statement enables a program to skip over part of the code. A break statement terminates the smallest enclosing **while**, **do-while**, **for** or **switch** statement. Execution resumes at the statement immediately following the body of the terminated statement.

The continue statement:- The continue is another jump statement some what like the break statement as both the statements skip over a part of the code. But the **continue** statement is somewhat different from **break**. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between.

QUESTIONS

1. Differentiate between Structure & Arrays.
2. What is the difference between ‘a’ and “a” in C++?
3. What are data types? What all are predefined data types in C++?
4. Why is char often treated as integer data type?
5. What is a reference variable? What is its usage?
6. Identify the errors in the following code segment:

```
int main()
{cout<<”Enter two numbers”;
cin>>num>>auto;
float area = length × breadth;}
```

7. An unsigned int can be twice as large as the signed int. Explain how.
8. What will be the result of the following expressions if I = 10 initially?

a. ++I <= 10 b. I ++ <= 10

9. How a = 10 and a = = 10 are different?
10. What will be the output of the following code fragment?

```
int val, res, n=100;
cin>>val;
res = n + val > 1750 ? 400 : 200;
cout<<res;
```

a. If the input is 2000 b. If the input is 1000 c. If the input is 500

CHAPTER-2 (Structure)

Structure: Structure is a derived data type, it’s basically a combination of different types of variables, which are grouped together so to be identified as a single entity. Structure can be visualized as record and its member as fields.

Defining a Structure: Format for defining a structure is as follows:

```
Struct Tag
{
member1;
member2;
member3;
};
```

E.g.: Struct biodata

```
{
char name[20];
char address[40];
int age;
float height;
};
```

Declaration of Structure variable:

E.g.

```
struct biodata emp1;
struct biodata emp2;
```

or

```
Struct biodata
{char name[20];
char address[40];
int age;
float height;
} emp1, emp2;
```

Array of Structures:

E.g.:

```
Struct biodata male[20], female[10];
```

or

```
Struct biodata
{char name[20];
char address[40];
int age;
float height;
} male[20], female[10];
```

Accessing individual members of the structure:

Making use of dot (.) operator as follows, can access the individual members:

Variable.member

E.g.: male.name

male.address

CHAPTER-4 (Class & Objects)

Introduction : The most important features of C++ are the classes and objects. The initial name of C++ was *"C with classes"*. The class is a single most important C++ feature that implements OOP concepts and ties them together. It implements data hiding, abstraction and encapsulation.

Definition: A class is a group of similar objects. A class is a way to bind the data describing an entity and its associated functions together. In C++ class makes a data type that is used to create objects of this type.

Example:-

```
class Account  {
                int accountno;
                char type;
                float balance;
                float deposit (float amount)
                        {       balance + = amount;
                                return balance;
                        }
                float withdraw (float amount)
                        {       balance – = amount;
                                return balance;
                        }
        };
```

Classes are needed to represent real-world entities that not only have data type properties (their characteristics) but also associated operations (their behaviours).

Declaration of Classes:-

The declaration of a class involves declaration of its four associated attributes:

1. *Data Members* are the data type properties that describe the characteristics of a class. There may be zero or more data members of any type in a class.
2. *Member Functions* **(Methods)** are the set of operations that may be applied to objects of that class. There may be zero or more member function for a class. They are referred to as the *class interface.*
3. *Program Access Levels* that controls access to members from within the program. These access levels are *private, public or protected*. Depending upon the access level of a class member, access to it is allowed or denied.
4. *Class Tagname* that serves as a type specifier for the class using which objects of this class type can be created.

The Class Member Functions (Methods) Definition:-

a. Outside the class definition
b. Inside the class definition

Outside the class definition:-

It is same as definition a normal function except that the name of the function is the full name *(qualified name)* of the function. The full name of the function *(qualified name)* is written as :

class name :: function name

(i.e. first we give the name of the class to which the function belong to then a scope resolution operator (::) followed by the name of the function)

The general form of a member function definition outside the class definition is :

```
return type class name :: function name (parameter list)
        {
                function body;
        }
```

Inside the Class Definition: It is just same as definition of a normal function without any difference. *Function defined inside the class specification are automatically* ***inline****.*

Objects: Objects are the instances of the class. To make use of class specified, objects have to be declared. For example to declare the objects of the above class Accounts:

Accounts saving, current;

Here saving and current are the two objects of the class Accounts.

Class Vs Structure:

A Class declaration looks exactly like the declaration of a Structure except

- ➢ The struct keyword has been substituted by the keyword class
- ➢ Member functions can be declared in a class but not in structure.
- ➢ It divides its members into segments : *public and private*
- ➢ By default members are private in class but public in structure.

Referencing Class Members:

The members of a class are referenced using objects of the class. For instance to access the member functions deposit and withdrawal of the class Account :

saving . deposit(2000);
saving . withdraw(1000);
current . deposit (10000);
current . withdraw(2000);

here saving and current are the objects of the class Account. So the general format for calling a member function is:

object – name . function – name (actual – arguments);

Scope of Class and Its Members:-

1. *Public Members* are the members (data members and function members) that can be used by any function (member as well as non member). It can also be used by the objects of the class.
2. *Private Members* are the members that are hidden from outside world. The private members implement the OOP concept of data hiding. The private members of a class can be used only by the member function and friend function of the class in which it is declared. It can't be used by the objects of the class.
3. *Protected members* are similar to private members that they cannot be accessed by non-member functions. The difference between protected and private members is that protected members get inherited while private members are not. (the difference between protected and private becomes more clear when we study inheritance)

Global Class: A class is said to global class if its definition occurs outside the bodies of all functions in a program, which mean that objects, of this class type can be declared from anywhere in the program.

Local Class: A class is said to be local class it its definition occurs inside a function body, which means that objects of this class type can be declared only within the function that defines this class type.

Global Object:-

An object is said to be a global object if it is declared outside all the function bodies and it means that this object is globally available to all functions in the program i.e. this object can be used anywhere in the program.

Local Object:-

An object is said to be a local object if it is declared within a function, which means that this object is locally available to the function that declares it and it cannot be used outside the function declaring it.

Nested classes:-

A class may be declared within another class. A class declared within another is called a nested class. The outer class is known as enclosing class and the inner class is known as nested class.

Inline Functions:-

The inline functions are a C++ enhancement designed to speed up programs. The coding of normal function and inline function is similar except that inline functions definitions starts with the keyword inline.

Working of normal functions:-

After loading the executable program in the computer memory, these instructions are executed step by step. When a function call instruction is encountered, the program stores the memory address of the instruction immediately following the function call statement, loads the function being called into the memory, copies argument values, jumps to the memory location of the called function, executes the function code, stores the return value of the function, and then jumps back to the address of the instruction that was saved just before executing the called function.

Working of inline functions:-

In case of inline function call, the compiler replaces the function call statement with the function code itself and then compiles the entire code. Thus with inline function the compiler does not have to jump to another location to execute the function, and then jump back as the code of the called function is already available to the calling program. Inline functions run a little faster than the normal function as function – calling – overheads are saved, however there is memory penalty. *An inline function definition should be placed above all the functions that call it.*

The functions should be inlined only when they are small. **The inlining does not work with the following situations:**

1. For functions that return values and are having a **loop** or **a switch** or **a goto**.
2. For functions not returning values, if a return statement exists.
3. If function contains **static** variables.
4. If function is *recursive* (a function that calls itself)

Friend Functions:-

A friend function is a non member function that's granted access to a class's private and protected members and a friend class is a class whose member functions can access another class's private and protected members.

Need for Friend Functions:-

The private and protected data of a class might need to be shared with some non member of the class. In such a situation, that non member may be declared a friend of the class, and thus get the privilege of accessing private and protected members of the class.

*When declaring a friend function, the keyword **friend** appears only in the prototype declaration of the function inside the class definition, but not in the actual function definition, which occurs outside the class definition.*

Facts about Friend Functions:-

1. A function may be declared friend of more than one class.
2. It does not have the class scope, rather it depends upon its original declaration and definition.
3. It does not require an object (of the class that declares it a friend) for invoking it. It can be invoked like a normal function.
4. Since it is not a member function (It is a friend function) it cannot access the members of the class directly and has to use an object name and membership operator (.) with each member name.
5. It can be declared anywhere (in any of the three section) in the class without affecting the meaning.
6. A member function of the class operates upon the members of the object used to invoke it, while a friend function operates upon the object(s) passed to it as arguments.

The Scope Resolution Operator (::) :-

The :: (Scope Resolution Operator) will uncover a hidden scope of global item.

If there are multiple variables with the same name, defined in separate blocks then the :: operator always refers to the file scope variable.

Memory Allocation of Objects:-

Member functions are created and placed in the memory space only once when the class is defined. The memory space is allocated for objects data members only when the objects are declared. No separate space is allocated for member functions when the objects are created.

Static Data Members:-

A static data member of a class is just like a global variable for its class. That it, this data member is globally available for all the objects of that class type. The static data members are usually maintained to store values common to the entire class. For instance, a class may have a static data member keeping track of its number of existing objects.

How it differs from Ordinary Data Members:-

1. There is only one copy of this data member maintained for the entire class which is shared by all the objects of that class.
2. It is visible only within the class, however, its lifetime (the time for which it remains in the memory) is entire program.

Steps for making a data member static:-

1. Declaration within the class definition.
2. Definition outside the class definition.

Example:-

```
class sample
        {
                :
                static int x;       //declaration
        };
int sample::x = 100;                //definition
```

Static Member Function:-

A member function that accesses only the static members of a class may be declared as static. This can be done by putting keyword static before the function declaration in the class definition as shown below:

```
class sample
        {
                :
                static int x;       //declaration
                static void show(void)
                        {
                                cout<<x<<endl;
                        }
};
int sample::x = 100;
```

How it differs from Ordinary Members Functions:-

1. A static member function can access only static members (functions/variables) of the same class.
2. A static member function is invoked by using the class name instead of its objects. To call static function show() we have to write sample::show() i.e. class - name :: function – name.

CHAPTER-5 (CONSTRUCTOR & DESTRUCTORS)

Definition :- A member function with the same name as its class is called constructor and it is used to initialize the objects of that class type with a legal initial value.

Example :-

```
class Student
        {
                int rollno;
                float marks;
        public:
                void input();
                void output();
                student()       //constructor
                {
                        rollno=0;
                        marks=0.0;
                }
        };
```

Need for Constructor: A constructor for a class is needed so that the compiler automatically initializes an object as soon as it is created. A class constructor, if defined, is called whenever a program creates an object of that class.

Definition of Constructor inside the class:-

```
class Student
        {
                int rollno;
                float marks;
        public:
                void input();
                void output();
                student();      //constructor
                {
                        rollno=0;
                        marks=0.0;
                }
        };
```

Definition of Constructor outside the class:-

```
class Student
        {       int rollno;
                float marks;
        public:
                void input();
                void output();
                student();      //constructor
        };

Student::Student()

                {       rollno=0;
                        marks=0.0;
                }
```

Note :- Generally a constructor should be defined under the public section of a class, so that its objects can be created in any function.

Constructor with arguments

```
class Student
        {
                int rollno;
                float marks;
        public:
                void input();
                void output();
                student(int, float);      //constructor
        };
Student::Student(int x, float y)
                {
                        rollno=x;
                        marks=y;
                }
```

when the constructor has argument then the object will be declared as follows:
student obj(5, 99.99);
Now the value of rollno will be 5 and value of marks will be 99.99 for obj. To check it you can call the function output() as follows
obj.output();

Default Constructor :-

If a class has no explicit constructor defined, the compiler will supply a default constructor.
X:X() is a default constructor (A constructor which does not accept arguments or parameters is called default constructor). With the default constructor objects are created just the same way as variables of other data types are created. (for example x obj1)
Note:- *The default constructor provided by the compiler does not do anything specific. It initializes the data members by any dummy values.*
Note :- *A constructor with default arguments is equivalent to a default constructor.*
X::X(int a=10, int b=20) is also a default constructor as we can create objects like x obj1.

Parameterized Constructors

A constructor with arguments is known as parameterized constructor. The parameterized constructors allow us to initialize the various data elements of different objects with different values when they are created. This is achieved by passing different values as arguments to the constructor function when the objects are created.

```
class Student
        {       int rollno;
                float marks;

        public:
                void input();

                void output();
                student(int, float);      //constructor
        };
```

```
Student::Student(int x, float y)
                {
                        rollno=x;
                        marks=y;
                }
Student New;  //  invalid
```

The above object declaration is invalid as Student constructor expects two arguments and no argument value has been supplied at the time of object declaration. Therefore, with parameterized constructor, the initial values must be passed at the time of object creation.

This can be done in two manners :

(a) by calling constructor implicitly (implicit call)

(b) by calling constructor explicitly (explicit call)

Implicit call to the constructor

Student New(5, 99.99);

This statement will create an object New of type Student and invoke the constructor (implicitly) of Students to initialize New with values 5 and 99.99.

Explicit call to the constructor

Student New = Student(5, 99.99);

This statement will create an object New of type Student and invoke the constructor (explicitly) of Student to initialize New with values 5, 99.

Temporary Instance: The explicit call to a constructor also allows you to create a temporary instance or temporary object. A temporary instance is the one that lives in the memory as long as it is being used or referenced in an expression and after this it dies. The temporary instances are anonymous i.e. they do not bear a name.

Note :- *An explicit call to the constructor lets you create a temporary instance.*

Constructor for fundamental data types:-

Even fundamental data types have constructor. For example int x(5) will assign value of 5 to x, float y(50.9) will assign 50.9 to y.

Copy Constructor :-

A copy constructor is a constructor of the form classname (classname &). The copy constructor gets invoked automatically at the following different situations.

(a) When an object is passed by value.

(b) When a function returns an object

(c) When a object is initialized dynamically. Ex Student Old = New.

Characteristics of Constructors:-

(1) Constructor function is invoked automatically when the objects are created.

(2) Constructor functions obey the usual access rules. That is, private and protected constructors are available only for member and friend functions, however, public constructors are available for all the functions that have access to the constructor of a class, can create an object of the calss.

(3) No return type (not even void) can be specified for a constructor.

(4) They cannot be inherited, though a derived class can call the base class constructor.

(5) A constructor may not be static.

(6) It is not possible to take the address of a destructor.
(7) Default constructors and copy constructors are generated (by the compiler) where needed. Generated constructors are public.
(8) Like other functions constructors can also have default arguments.
(9) An object of a class with a destructor cannot be a member of a union.
(10) A constructor can be used explicitly to create new objects of its class type. For example Student New = Student (5, 99.99);
(11) The default constructor and the copy constructor are generated by the compiler only if these have not been declared for a class.

Destructors :-
A destructor is also a member function whose name is the same as the class name but is preceded by tilde ('~') sign. For example ~Student() is a destructor for class Student.
Need for Destructor:-
The allocated resources must be deallocated before the object is destroyed. Now this is done by the destructor.

***Note :-** Only the function having access to a constructor and destructor of a class, can define objects of this class types otherwise the compiler reports an error.*
Generally a destructor should be defined under the public section of a class so that its objects can be destroyed in any function.

Characteristics of a Destructor
(1) Destructor functions are invoked automatically when the objects are destroyed.
(2) If a class has a destructor, each object of that class will be deinitialized before the object goes out of scope. (Local objects at the end of the block defining them and global and static objects at the end of the program)
(3) Destructor functions also, obey the usual access rules as other member function do.
(4) No argument can be provided to a destructor, neither does it return any value.
(5) They cannot be inherited.
(6) A destructor may not be static.
(7) It is not possible to take the address of a destructor.
(8) Member functions may be called from within a destructor.
(9) An object of a class with a destructor cannot be a member of a union.

CHAPTER-6 (INHERITANCE)

Definition: Ability of one class to derive (get) properties form the existing class (old class known as base class) is known as inheritance.
The class inheritance lets you derive new classes (derived classes) from old ones, with the derived class inheriting the properties(data members,function members) of the old class, known as base class.
Need for Inheritance: **a.**Closeness to the real world(capability to express the inheritance relationship)
b.Reusability. (We can reuse some of the properties of the existing class)
c.Transitive in nature. (A $\rightarrow$ B $\rightarrow$ C here C will have some of the properties

of A though C is inherited from B but not from A)

Different forms of Inheritance:

1. Single Inheritance

$$\begin{tikzcd} A \arrow[d] \\ B \end{tikzcd}$$

2. Multiple Inheritance

$$\begin{tikzcd} A \arrow[dr] & & B \arrow[dl] \\ & C & \end{tikzcd}$$

3. Hierarchical Inheritance

$$\begin{tikzcd} A \arrow[dr] & B \arrow[d] & C \arrow[dl] \\ & D & \end{tikzcd}$$

4. Multilevel Inheritance

$$\begin{tikzcd} A \arrow[d] \\ B \arrow[d] \\ C \end{tikzcd}$$

5. Hybrid Inheritance

$$\begin{tikzcd} & A \arrow[dl] \arrow[dr] & \\ B \arrow[dr] & & C \arrow[dl] \\ & D & \end{tikzcd}$$

Syntax of defining a derived class:

```
class derived-class-name : visibility-mode base-class name
{
        :
        :       //definition of a class normally
};
```

The visibility mode may be either private or public or protected. If no visibility mode is specified, then by default the visibility-mode is considered as private.

Visibility Modes:

The visibility mode controls the visibility of inherited base class members in the derived class.

A derived class inherits all the non-private members of the base class, i.e. the derived class has access privilege only to the non-private members of the base class.

Private:

If the derived class is inherited privately form the base class elements then

Private data members	Not inheritable
Protected data members	Inheritable (protected members from the base class will be available under the private section of the derived class)

Public data members Inheritable (public members from the base class will be available under the private section of the derived class)

Protected:

If the derived class is inherited privately form the base class elements then

Private data members Not inheritable

Protected data members Inheritable (protected members from the base class will be available under the protected section of the derived class)

Public data members Inheritable (public members from the base class will be available under the protected section of the derived class)

Public:

If the derived class is inherited privately form the base class elements then

Private data members Not inheritable

Protected data members Inheritable (protected members from the base class will be available under the protected section of the derived class)

Public data members Inheritable (public members from the base class will be available under the public section of the derived class)

Solved Questions:

What does inheritance mean? Or ***(CBSE Question Bank 1998)***

What is the inheritance mechanism in C++ ? ***(CBSE Question Bank 1998)***

Ans. Inheritance is the capability of one class of things to inherit capabilities or properties from another class. In C++, a class inheriting its properties from another class, is known as subclass or derived class. The class, whose properties are inherited, is known as base class or super class.

What is single inheritance ? What is the multiple inheritance ? ***(CBSE Question Bank 1998)***

Ans. When a subclass inherits only from one base class, it is known as single inheritance.

$$\begin{tikzcd} X \arrow[d] & \text{Base class} \\ Y & \text{Sub class} \end{tikzcd} \qquad \begin{tikzcd} X \arrow[dr] & & Y \arrow[dl] & \text{Base Classes} \\ & Z & & \text{Sub Class} \end{tikzcd}$$

(a) Single inheritance (b) Multiple inheritance.

When a subclass inherits from multiple base classes, it is known as multiple inheritance.

State very briefly how protected members are inherited publicly and privately?*(CBSE Question Bank 1998)*

Ans. When a protected member is inherited publicly, it remains protected in the derived class. On the other hand when a protected member is inherited privately, it becomes private in the derived class.

Difference between privately and publicly derived visibility modes? ***(CBSEQuestionBank1998)***

Ans. With publicly derived class, the public and protected members of the base class remain public and protected in the derived class also. Private members of the base class are not inherited at all. With privately derived class, the public and protected members of the base class become private members of the derived class. Private members of the base class are not inherited at all.

<u>CHAPTER-7 (File Handling)</u>

Stream :- A sequence of bytes

File: A collection of data or information stored on some media with some specific name is called file.

Text File :- The file which stores data or information in ASCII characters. In text file each line is terminated with a special character “\n” or endl (end of line) character also known as delimiter character.

Binary File :- It is a file that contains information in the same format as it is stored in the memory. In binary file no delimiter character is used. The records are separated by record size.

Which library would you use for file I/O?

Fstream.h

What types of streams would you declare for the following purposes?

(i) for input from file		**(ii) for output to a file**	**(iii) for both Input and Output**
Ans	(i) ifstream	(ii) ofstream	(iii) fstream

The classes defined under fstream.h is derived from which classes.
The fstream.h classes are derived from istream.h classes.
What is the purpose of getline function.
It reads characters from the input stream till delimiter is encountered or desired no of characters.
Differentiate between get and getline function of istream class.
The get() and getline() functions are identical except getline() removes the delimiter from the buffer where as get() does not.
How is binary file is different from text file?
In text mode character translation takes place line "\n" gets translated to new line where as no such character translation takes place in binary file. In text mode reading and writing is done character by character where as in binary file the structure variables or objects gets written in one go.
Differentiate between ios::ate and ios::app modes.
Initially when the file is opened the file pointer gets placed at end of file in both the cases but ios::app allows writing one at the end of file (EOF) but ios::ate allows writing any where within the file.
Differentiate between get(), put() and read() , write() functions.
In get() or put() function I/O is done byte by byte. On the other hand read() or write() you can read or write one record in one go. (one structure or object).
Differentiate between read() and write() function
Read() reads one record from the file where as write() writes one record to the file.
Write name of two member functions belonging to fstream class.
seekp(), seekg(), tellp(), tellg()
Distinguish between serial and sequential file.
In serial file records are arranged in the manner they are inserted in the file. There is no significance of primary key. In sequential file record can be arranged in the order of their primary key.
What are file modes? What role do they play in file I/O?
The filemode describes how a file is to be used in the program.

ios::in	opens file for reading (i.e input mode)
ios::out	opens file for output (i.e output mode)
ios::ate	writing can be done in middle of file
ios::app	writing can be done only at the end of file.
ios::trunc	opens in output mode & erases the previous content of file
ios::nocreate	open fails if file doesn't exist. (output mode)
ios::noreplace	open fails if file exist. (output mode)
ios::binary	opens the file in binary mode.

Chapter-8 (Pointers)

Pointers are variables that hold addresses in C and C++. They provide much power and utility for the programmer to access and manipulate data in ways not seen in some other languages. They are also useful for passing parameters into functions in a manner that allows a function to modify and return values to the calling routine. When used incorrectly, they also are a frequent source of both program bugs and programmer frustration.

Introduction: As a program is executing all variables are stored in memory, each at its own unique address or location. Typically, a variable and its associated memory address contain data values. For instance, when you declare:

```
int count = 5;
```

The value "5" is stored in memory and can be accessed by using the variable "count". A pointer is a special type of variable that contains a memory address rather than a data value. Just as data is modified when a normal variable is used, the value of the address stored in a pointer is modified as a pointer variable is manipulated.
Usually, the address stored in the pointer is the address of some other variable.

```
int *ptr;
ptr = &count    // Stores the address of count in ptr
    // The unary operator & returns the address of a variable
```

To get the value that is stored at the memory location in the pointer it is necessary to dereference the pointer. Dereferencing is done with the unary operator "*".

```
int total;
total = *ptr;
    // The value in the address stored in ptr is assigned to total
```

The best way to learn how to use pointers is by example. There are examples of the types of operations already discussed below. Pointers are a difficult topic. Don't worry if everything isn't clear yet.
Declaration and Initialization: Declaring and initializing pointers is fairly easy.

```
int main()
{
    int j;
    int k;
    int l;
    int *pt1;    // Declares an integer pointer
    int *pt2;    // Declares an integer pointer/
    float values[100];
    float results[100];
    float *pt3;    // Declares a float pointer
    float *pt4;    // Declares a float pointer
    j = 1;
    k = 2;
    pt1 = &j;    // pt1 contains the address of the variable j
    pt2 = &k;    // pt2 contains the address of variable k
    pt3 = values;
        // pt3 contains the address of the first element of values
     pt3 = &values[0];
        // This is the equivalent of the above statement
    return 0;
}
```

Pointer Dereferencing/Value Assignment: Dereferencing allows manipulation of the data contained at the memory address stored in the pointer. The pointer stores a memory address. Dereferencing allows the data at that memory address to be modified. The unary operator "*" is used to dereference. For instance:

```
*pt1 =*pt1 + 2;
```

This adds two to the value "pointer to" by pt1. That is, this statement adds 2 to the contents of the memory address contained in the pointer pt1. So, from the main program, pt1 contains the address of j. The variable "j" was initialized to 1. The effect of the above statement is to add 2 to j.
The contents of the address contained in a pointer may be assigned to another pointer or to a variable.

```
*pt2 = *pt1;
   // Assigns the contents of the memory pointed to by pt1
   // to the contents of the memory pointer to by pt2;
k = *pt2;
   // Assigns the contents of the address pointer to by pt2 to k.
```

Pointer Arithmetic: Part of the power of pointers comes from the ability to perform arithmetic on the pointers themselves. Pointers can be incremented, decremented and manipulated using arithmetic expressions. Recall the float pointer "pt3" and the float array "values" declared above in the main program.

```
pt3 = &values[0];
   // The address of the first element of "values" is stored in pt3
pt3++;
   // pt3 now contains the address of the second element of values
*pt3 = 3.1415927;
   // The second element of values now has pie (actually pi)
pt3 += 25;
   // pt3 now points to the 27th element of values
*pt3 = 2.22222;
   // The 27th element of values is now 2.22222

pt3 = values;
   // pt3 points to the start of values, now

for (ii = 0; ii < 100; ii++)
{
    *pt3++ = 37.0;    // This sets the entire array to 37.0
}

pt3 = &values[0];
   // pt3 contains the address of the first element of values
pt4 = &results[0];
   // pt4 contains the address of the first element of results

for (ii=0; ii < 100; ii++;)
{
    *pt4 = *pt3;
          // The contents of the address contained in pt3 are assigned to
          // the contents of the address contained in pt4
    pt4++;
    pt3++;
}
```

Pointer and Function

Returning Multiple Values From Functions:

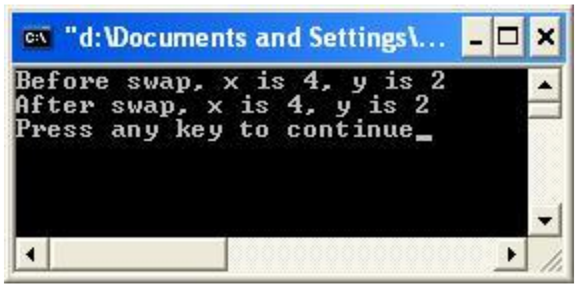
So far, all the examples shown have either not returned any values, or have returned a single value using the return statement. In practice, it will be common to need to return multiple values from a function. Let's see how this can be done by developing a function to swap two integer values. Here's a first attempt.

```
#include <stdio.h>

void swap(int x, int y);
        /* Note that the variable names in the
           prototype and function
           definition need not match.
           Only the types and number of
           variables must match */

int main()
{
    int x = 4;
    int y = 2;
    cout<<"Before swap x is “<<x<<”y is ”<< y<<endl;
    swap(&x,&y);
    cout<<"After swap x is “<<x<<”y is ”<< y<<endl;
}
void swap(int first, int second)
{
    int temp;
    temp = second;
    second = first;
    first = temp;
}
```

Results:



```
Before swap, x is 4, y is 2
After swap, x is 4, y is 2
Press any key to continue_
```

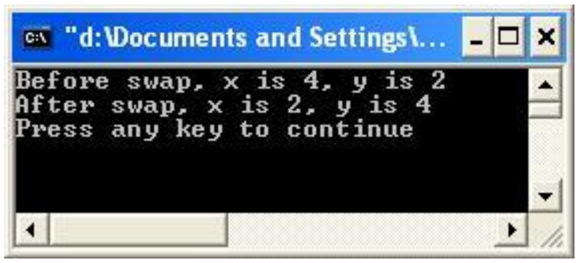
What happened? The values weren't swapped. The function didn't work as expected. In C, all arguments are passed into functions by value. This means that the function receives a local copy of the argument. Any modifications to the local copy do not change the original variable in the calling program.

If a variable is to be modified within a function, and the modified value is desired in the calling routine, a pointer to the variable should be passed to the function. The pointer can then be manipulated to change the value of the variable in the calling routine. It is interesting to note that the pointer itself is passed by value. The function cannot change the pointer itself since it gets a local copy of the pointer. However, the function can change the contents of memory, the variable, to which the pointer refers. The advantages of passing by pointer are that any changes to variables will be passed back to the calling routine and that multiple variables can be changed. Here is the same example with pointers being passed into the function.

```
#include <stdio.h>
void swap(int *x, int *y);
int main()
{
   int x = 4;
   int y = 2;

   cout<<"Before swap x is “<<x<<”y is ”<< y<<endl;
   swap(&x,&y);
   cout<<"After swap x is “<<x<<”y is ”<< y<<endl;
   return 0;
}
void swap(int *first, int *second)
{
   int temp;
   temp = *second;
   *second = *first;
   *first = temp;
}
```

Results:



```
Before swap, x is 4, y is 2
After swap, x is 2, y is 4
Press any key to continue
```

Passing Arrays to Functions

When an array is passed into a function, the function receives not a copy the array, but instead the address of the first element of the array. The function receives a pointer to the start of the array. Any modifications made via this pointer will be seen in the calling program. Let's see how this works. Suppose the main program has an array of 10 integers and that a function has been written to double each of these.

```
void doubleThem(int a[], int size);
int main()
{
   int myInts[10] = {1,2,3,4,5,6,7,8,9,10};

   doubleThem(myInts, 10);

   return 0;
}
void doubleThem(int a[], int size)
{
   int i;
```

```
    for (i = 0; i < size; i++)
    {
        a[i] = 2 * a[i];
    }
}
```

Note that due to relationship between pointers and arrays in C, this function could also be written as follows and the rest of the program could be used without modification.

```
void doubleThem(int *pt, int size)
{
    int i;
    for (i = 0; i < size; i++)
    {
        *pt = 2 * *pt;
        pt++;
    }
}
```

Strings are stored in C as null terminated character arrays. To conclude this lesson, let's see how to pass strings into and out of functions by implementing a string copy function. Note that a string copy function, strcpy, is part of the C standard library.

```
#include <stdio.h>
void strcpy(char *dest, char *source);
int main()
{
    char a[100] = "Hello From About C/C++";
    char b[100];

    strcpy(b,a);
    printf("%s\n",b);
}
void strcpy(char d[], char s[])
{
    int i = 0;
    for (i = 0; s[i] != '\0'; i++)
    {
        d[i] = s[i];
    }
    d[i] = s[i]; /* Copy null terminator to dest */
}
```

Relationship Between Pointers and Arrays:

In some cases, a pointer can be used as a convenient way to access or manipulate the data in an array. Suppose the following declarations are made.

```
float temperatures[31];
    /* An array of 31 float values, the daily temperatures in a month */

float *temp;    /* A pointer to type float */
```

Since temp is a float pointer, it can hold the address of a float variable. The address of the first element of the array temperatures can be assigned to temp in two ways.

```
temp = &temperatures[0];
temp = temperatures;
   /* This is an alternate notation for the first
   element of the array. Same as temperatures = &temperatures[0]. */
```

The temperature of the first day can be assigned in two ways.

```
temperatures[0] = 29.3;    /* brrrrrrr */
*temp = 15.2;    /* BRRRRRRR */
```

Other elements can be updated via the pointer, as well.

```
temp = &temperatures[0];

*(temp + 1) = 19.0;
   /* Assigns 19.0 to the second element of temperatures */

temp = temp + 9;
   /* temp now has the address of the 10th element of the array */

*temp = 25.0;
   /* temperatures[9] = 25, remember that arrays are zero based,
   so the tenth elementis at index 9. */

temp++;    /* temp now points at the 11th element */

*temp = 40.9;    /* temperatures[10] = 40.9 */
```

Pointers are particularly useful for manipulating strings, which are stored as null terminated character arrays in C

This pointer

Introduction: This lesson covers the purpose and use of the "this" pointer. In addition to the explicit parameters in their argument lists, every class member function (method) receives an additional hidden parameter, the "this" pointer. The "this" pointer addresses the object on which the method was called. There are several cases when the "this" pointer is absolutely required to implement desired functionality within class methods.

Purpose: To understand why the "this" pointer is necessary and useful let's review how class members and methods are stored. Each object maintains its own set of data members, but all objects of a class share a single set of methods. This is, a single copy of each method exists within the machine code that is the output of compilation and linking. A natural question is then if only a single copy of each method exists, and its used by multiple objects, how are the proper data members accessed and updated. The compiler uses the "this" pointer to internally reference the data members of a particular object. Suppose, we have an Employee class that contains a salary member and a setSalary method to update it. Now, suppose that two Employees are instantiated.

```
class Employee {
public:
   ....
   void setSalary(double sal)
   {
      salary = sal;
   }
private:
   ....
   double salary;
   ....
}
int main()
{
   ....
   Employee programmer;
   Employee janitor;
   janitor.setSalary(60000.0);
   programmer.setSalary(40000.0);
   ....
}
```

If only one setSalary method exists within the binary that is running, how is the correct Employee's salary updated? The compiler uses the "this" pointer to correctly identify the object and its members. During compilation, the compiler inserts code for the "this" pointer into the function. The setSalary method that actually runs is similar to the following pseudo-code.

```
void setSalary(Employee *this, float sal)
{
   this->salary = sal;
}
```

The correct object is identified via the "this" pointer. So what, you say. Interesting, maybe. But if this implicit use of the "this" pointer were all there was, this topic would mainly be of interest to compiler designers. As we will see, explicit use of the "this" pointer also has great utility.

Chapter-9,10 (DATA STRUCTURES)

1. Introduction-

A computer system is mainly used as data management system, where ‘data’ is very important. Data may be organized in different ways, which increase or decrease the efficiency of processing. Means data must be organized in a proper manner. Data structure is a way to achieve the same.

2. What is data type & data structure?

Data can be represented as raw data, data item and data structure. Raw data is simply set of values given to computer system as input. Data item represents single unit of values of some specific type.

Now we must know about data type. Same type of data which has same properties & characteristics can be grouped to form an atomic unit. For example an object name can contain only alphabets, quantity can have only digits, and so on. Such type of data can be represented by fundamental data types .

In C++ following data types are available-

Data Structure:

Before we know about data structures, we must know about representation of data, which can be in form of *data items* and *data structures*.

Data Structure is an organized collection of various data items related to one another in different ways that can be used as a single unit.

Now we can say:

Data structure =organized data + allowed operations

Data type=data value + operations

3 <u>Operations on data structure</u>

Some common operations which are normally performed on a linear structure are :

1. insertions-process to add new data item
2. traversal-it is a method of accessing each element individually for information or processing
3. search-it is an activity in which whole data structure is searched to find the location of specific data item
4. deletion-it is a process to remove a particular data item from existing data structure
5. sorting-it is a technique of arranging the data items of a data structure in ascending/descending order
6. merging-is a process of combining two or more list into a single list

4. <u>Arrays</u>

An array is a collection of related data items, which can be accessed by a common name. Arrays are the simplest data structures where elements of arrays require contiguous memory locations. An array in C++ an be declared as follow:

type arrayname[size];

Where type is the data type of variables or which array is forming, array name is the common name by which all the variables can be accessed and size is the number of variables(i.e. 0 to size –1) it is also called index or subscript of the array. In an array all the variables are distinguished identified with the help of subscript only.

for e.g : int arr[5];

In the above example **arr** is the name of the array variable which is a group of 5 integer variables. Further arr[0] is the first variable, arr[1] is the second variable ------ and arr[4] is the last variable (i.e 5 - 1)

NOTE : In C++ array size must be constant and positive value because the memory allocation for an array is always done at the time of compilation and called static Data Structure.

5. <u>One Dimensional Arrays:-</u>

A one-dimensional array is the simplest form of an array. An array which represents a list of values and has only one subscript is called one dimensional array. An individual variable in the array is called an array aliment. Array allows to access the group of related data in easier way.

For example: - WAP to input 10 and print the total and the average of these numbers.

// Complete the following program given below uses to input 5 integer values into an array val and print them on screen including the some and average.

```
# include<iostream.h>
#include<conio.h>
void man( )
{
int val[5]; //declare an array of 5 integer space
int avg, sm;
------
------
}
```

Basic Operations on One Dimensional Array

A. Traversal:- Traversal in one dimension array means accessing each and every element of the array one by one for some manipulations. Suppose an array arr has N elements and data array will be accessed. As N elements will be traveled so let us take a variable I imiliatizing value 1 to N (i.e –

lower limit to upper limit) at the timeof traveling.I is increamented one by one till N. Here value of I willindicate which element isbeing accessed.

Alogorithm for the same is given below :-
Let : ARR[ ] is the array of size N

```
Start:
        Step 1 : set I:=1
        Step 2: While I<= N repeat step 3 and step 4.
        Step 3 : Process ARR[ I ]
        Step 4 : Set I : = I+1
                [ end of while ]
Stop:
```

B. <u>Searching :</u> - Searching of a data Structure is a process of finding a specific data and its location. There one various searching algorithms but according to our syllabus we have to learn two very common technique

= Linear Search
= Binary Search.

Linear Search:- It is the simplest method of searching. In this process each elements of the array(starting from beginning) is examined sequentially until the given data is found**.**

***Algorithm* for Linear Search**
Let : ARR[ ] is an array of size N and VAL is the data item to be searched pos is the variable which will hold the location of VAL. I is initialized value 1 and it will be incremented at every step until VAL is found.

```
Start :
        Step1: Set I: = 1, pos : = -1
        Step2: While I < + N repeat Step 3 & Step 4
        Step3: IF ARR[ I ] = VAL Then
            {
3.1.1               set pos : = I
3.1.2               Go To step
            }  [end of IF ]
        Step 4: set : = I + 1
                [end of while ]
        Step 5: IF pos < > -1 Then
                        Print " Data Present at location",pos
                ELSE
                        Print "Data Not present"
                [end of if]
Stop:
```

In the above process if the data to be searched is at the last position of the array then it will becomes worst, because no. of comparisons is same as no. of elements in the list which is time consuming. Solution of such problem is Binary Search.

<u>Binary Search :</u>

This popular search process finds the data in minimum comparisons. But there is one pre-condition for this method i.e. ‘**the list must be in sorted order** (either ascending or descending)’**.** Before discussing algorithm we would like to indicate the general idea of this technique.

Suppose one wants to find a page no. 50 in a book. Obviously one does not perform linear search, rather one opens the almost in the middle to check which half contains the page no. 50. Then again opens that half in middle to check which quarter contains the page 50. Then again opens that quarter in middle to check which eighth of book contains page 50. And so on until, one finds the page no 50.

Here we compare the given data (to be searched) with the middle element of the array that may three outcomes. First, is the data is less than the middle element of the array i.e. value may present in the first half of the array and the data is to be searched within this part only. Second, is data is greater than the middle element i.e. data may present in the last half of the array and data is to be searched within this part of the array. Third, is data is equals to the middle element i.e. data is at middle position and program will be back with location of data. In first two cases again same process is repeated for the new part of the array.

***Algorithm for binary search*:**

Let: ARR[ ] is the array of **N** elements stored in ascending order. The DATA is to be searched in ARR[ ]

```
        Start:
Step 1 :        Set left := 1, right := N
Step 2 :        WHILE left <= right REPEAT step 3 & 4
Step 3 :        mid = (left + right)/2
Step 4 :        If ARR[mid] == DATA then
                        Print “Seach Successful”
                        Print DATA, “found at ”, mid
                        Break
                Else if ARR[mid]< DATA then
                        right=mid+1
                Else
                        left=mid-1
                End if
                Step 5 :        Print “data is not in list”
        Stop:
```

SORTING OF ARRAY

There are a large number of methods for sorting arrays. We will discuss the following techniques for sorting arrays.

A. Selection sort
B. Bubble sort
C. Insertion sort.

We will assume that it is required to sort the array in ascending order. The programs require only minor modification for sorting the array in descending order.

A. SELECTION SORT:

In selection sort the array is divided into two parts, the sorted part and the unsorted part. To start with, the entire array is assumed to be unsorted.

The algorithm for selection sort is given below.

Let : The array x[] is assumed to have N elements.

```
START:
STEP 1      For I = 1 to N – 1 repeat steps 2 to 5
STEP 2          small = x[I],  L = I
STEP 3          for j = I + 1 to n repeat step 4
STEP 4              if small > x[j]
                        Small = x[j] ; L = j
                    {End of loop j}
STEP 5          Swap elements I and L
            {End loop I}
STOP
```

Let us write a Function in C++ for selection sort

```
void selection_sort ( )
{
        int I, j ;
        int L ;
        int small ;
        for (I = 0 ; I<n ; I++)
        {
        small = x[I] ;
        L = I ;
        For(j = I + 1 ; j<n ; j++)
                {
                        if small  x[j])
                                {
                                        small = x[j] ;          //smallest element
                                        L = j ;                 //and its locations
                                        }
                                }
                        int temp=x[I];                          //interchange elements I and L
                        x[I] = x[L];
                        x[L] = temp;
                }
}
```

B. BUBBLE SORT

In bubble sort, two consecutive elements are compared and are interchanged immediately if required. Suppose, it is required to sort an array x[] with \n elements. Element x[I] is compared with x[I + 1]. Elements x[I] and x[I + 1] are interchanged immediately if x[I] happens to be greater than x[I + 1]. When the process is repeated for I = 0, 1, 2…n-1, the largest element in the array moves to the end of the array. This completes one pass. The process is repeated in case some interchange is made and the program terminates otherwise.

The algorithm for bubble sort of an x[] with n integer number is given below :

```
START:
STEP 1      changed = 0
STEP 2      for j = 0 to j = N – 1
STEP 3          if x[j] > x[j + 1]
```

```
                              Swap x[j] and x[j + 1]
STEP 4                        changed = 1
                          {end if}
                         {end for loop j}
STEP 5          if changed = 0 go to step 2
STOP:
```

Let us write a Function in C++ for bubble sort

```
void bubble_sort ( )
{
        int j ;
        int changed ;
        do
                {
                    changed = 0 ;
                    for (j = 0 ; j < N ; j++)
                         if (x[j] > x[j + 1])
                           {
                                 swap (j, j + 1) ;
                                 changed = 1 ;
                           }
                } while changed == 1)
}
```

3. INSERTION SORT:
In insertion sort, the array is decomposed into two parts, sorted and unsorted. The unsorted pat of the array follows the sorted part. To start with, only the first element is considered to belong to the sorted part and rest to unsorted part. The first element of the unsorted part is selected and inserted into the appropriate location in the sorted part.

```
void insertion_sort( )
{
        int I, j ;
        int temp ;
        for I = 1; I < N ; I++)
                {
                        temp = x[I] ;
                        j = I – 1 ;
                        while (temp < x[j] && j > = 0)
                {
                        x[j + I] = x[j] ;
                        j = j – 1 ;
                }
        x[j + 1] = temp ;
        }}
```

MERGING OF TWO ARRAYS
Merging is defined as the operation in which the elements of two arrays are combined to form a single array.

If arrays A and B, the input to the program, are sorted. Third array is to be formed which must be in sorted order. Sorting the final array is an overhead, which can be avoided if the final array is generated by appending numbers in sorted order to the final array so that they are in ascending order. The algorithm for merging two arrays in the above manner is given below : -

```
START:
STEP 1      p1 = 0 p2 = 0 i = 0
STEP 2      repeat Step 3 to Step 6 while p1 < n1 and p2 < n2
STEP 3      if A[p1] < B[p2] go to step 4 else go to step 5
STEP 4      C[I] = A[p1], I = I + 1, p1 = p1 + 1
STEP 5      C[I] = B[p2] I = I+1, p2 = p2 + 1
            { end of if     }
STEP 6      If  p1 < N then Repeatedly add all remaining elements of array A[ ] into C[ ]
STEP 7      If  p2 < M then Repeatedly add all remaining elements of array B[ ] into C[ ]
STOP:
```

Let us write a C++ function to merge two arrays

```
data_type * merge_sorted (data_type A[ ], data_type B[ ], int n, int m)
{
        int I ;
int pa pb ;
array C ;
        pa = 0 ;
        pb = 0 ;
int k = n + m;
I = 0 ;
while ( pa < n  && pb < m)
{
        if (A[pa] < B[pb])
        C[I] = A[pa++ ];
        else
                C[I] = B[pb++] ;
        I++ ;
}
for (    ; pa < n ; I++)
C[I] = A[pa++] ;
for (   ; pb < m ; I ++)
C[I] = B[pb++] ;

return (C);
}
```

TWO DIMENSIONAL ARRAY

A two dimensional array may be represented as a matrix with a number of columns. It can be pictorially represented as given in Fig

I/j	0	1	2	3

0	x[0][0]	x[0][1]	x[0][2]	x[0][3]
1	x[1][0]	x[1][1]	x[1][2]	x[1][3]
2	x[2][0]	x[2][1]	x[2][2]	x[2][3]
3	x[3][0]	x[3][1]	x[3][2]	x[3][3]

However, Fig shows a two dimensional array as the programmer views the data. The data is not stored in the memory of the computer in this manner. The elements of an array, two or one dimensional, occupy continuous memory locations. The elements of the two dimensional aray may be stored in the memory in two feasible ways – Row major order. in the former, starting from the first row, all the elements of a row are stored sequentially in consecutive memory locations. In the later, elements of the columns, starting from the first and increasing in sequence occupy continuous memory locations. The way the array is stored in by both the mechanisms is shown in Fig.

Row major Order	Column major Order
x[0][0]	x[0][0]
x[0][1]	x[1][0]
x[0][2]	x[2][0]
x[0][3]	x[3][0]
x[1][0]	x[0][1]
x[1][1]	x[1][1]
x[1][2]	x[2][1]
x[1][3]	x[3][1]
……. …….	…… ……
X[n][m-1]	x[n-1][m]
X[n][m] .	x[n][m] .

ADDRESS OF INDIVIDUAL ELEMENTS OF TWO DIMENSIONAL ARRAY

The base address of a two dimensional array x[] [] is the memory location of the first element of the array x[0] [0]. From fig it can be seen that the address of any element x[I] [j] will depend upon the way the array is stored.

Suppose, x[M] [N] is an array of real numbers. The base address of x[] [] is x. let us find the address of element x[2] [3]. In row major order element x[2] [3] will be in the third row and 4t^{h,} column of array x[M] [N]. Before this element there will be two full rows (row 0 and 1) and three elements of the third row. Since there are N elements in each row, the number of elements which require memory assignment before x[2] [3] are N*2+3. Since each float variable requires four bytes for its storage, the memory requirement for storing all the elements preceding element x[2] [3] is 4(N*2+3). Therefore the address of element x[2] [3] is X + 4 (N * 2 + 3)

In general, the address of element x[I] [j] will be given by

$$A = X + K * (N * I + J)$$

Where,

X is the base address of array x[ ][ ]
N is the number of columns in the array
K is the number of bytes reqired for storage of each element of the array

Similarly, in column major order the address of element x[I][J] will be given by

$$A = X + K (M * J + I)$$

Where,

X is the base address of array x[ ][ ]
M is the number of rows in the array
K is the number of bytes required for storage of each element of the array

If address of any element x[p][q] is B, the address of any element x[I] [J] can be expressed as

$$A = B + K [N * (j – q) + (I – p)]$$

STACKS AND QUEUES

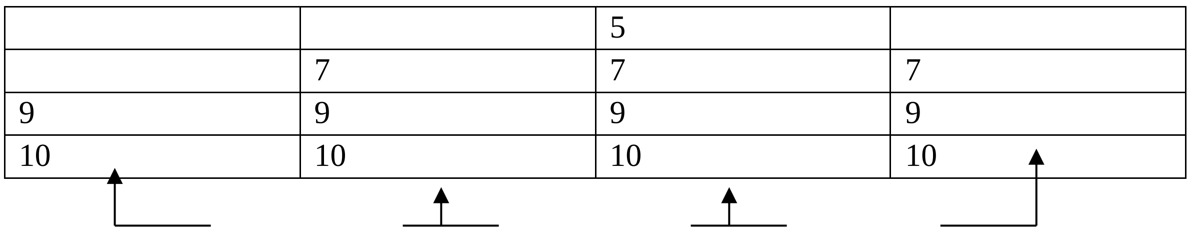
INTRODUCTION

In array, insertion and deletion can be done randomly at any location, be it at the start, some where at the middle or at the end. But in some special data structure insertions or deletions are possible at specific locations. Stacks and queues are such special data structures. In stacks insertion and deletion is possible at one end of the list, known as the top. In queues insertion is possible only at one end known as the rear end and deletion is possible from the other end, known as the front end. Here, we will discuss operations in stacks and queues and some of their practical applications.

THE STACK

Physically, stack is a LIFO(Last in First Out) type of data structure. The size of a stack can increase or shrink in a dynamic manner as items are added to the stack or removed from it. But when an item is to be deleted from the stack, the one, which was last inserted into the system, is the first to be deleted. This is known as LIFO discipline. In a stack the process of insertion and deletion of elements are known as push and pop respectively.

Fig : explains the push and pop operations in a stack.



		5	
	7	7	7
9	9	9	9
10	10	10	10

Push (7) *Push (5)* *Pop (5)*

A stack can be implemented as an array or as a linked list.

BASIC OPERATIONS AND IMPLEMENTATION OF STACK

The basic operations that can be performed on stack are:

- Creation of stack

- Check for empty stack
- Check for full stack
- Add element in stack
- Delete element from stack
- Print stack

A stack is a list, any list implementation can be implemented by using stack. We can implement a stack by using following data structures:

- Array implementation
- Linked list implementation

Array implementation:

The first implementation of stack by using array let ST[max] with **max** no of element in the stack. The first element in the stack will be ST[0], the second ST[1] and so on. To push an element on the stack top, we increment the top and set the value at ST[top] = value. To pop the element from the stack first we obtains the popped value and then decrement the top.

A general definition for linear stack:

```
Data_type Stack[size];
int top= -1;
```

to check whether a stack is empty or full we have to check only the value of top. If top is greater than or equals to the maximum size then the stack is full i.e. call ‘STACK OVERFLOW’ and if top is less than zero then stack is empty i.e. called ‘STACK IS UNDERFLOW’.

ALGORITHM FOR PUS:

In this algorithm we push an element called data on the top of the linear stack called **stack[max]** which can hold maximum **max** elements. The variable **top** is pointing to topmost element of stack.

```
Start:
        Step1   :       if top >= max   then
                                Print “ STACK OVERFLOW”
                                exit
                        { end of if       }
        Step 2  :       top = top + 1
        Step 3  :       stack[top] = data
Stop:
```

Let us write a general function in C++ called push, to insert elements on the top of the stack.

```
        void Push( data_type stack[ ], int &top, data_type data)
                {
                        if((top+1)>= max)
                                {
cout<<”STACK IS OVERFLOW”;
                                return(-1);
                                }
                        top++;
                        stack[top] = data;
                return(1);
}
```

Algorithm for Pop:

In this algorithm we pop an element from the stack and copy the contents of topmost element into a variable let **data**.

```
Start:
        Step 1  :       if top < 0 then
                                Print " STACK IS UNDERFLOW"
                                exit
                        { end of  if }
        Step 2  :       data = stack[top]
        Step 3  :       top = top - 1
Stop:
```

Let us write a general function to pop elements from the stack.

```
data_type Pop(data_type stack[ ], int &top)
                {
                        data_type data;
                        if(top < 0)
                        {
                                cout<<"STACK IS UNDERFLOW";
                                return(-1);
                        }
                data = stack[top];
                top = top – 1;
                return(data);
                }
```

LINKED LIST IMPLEMENTATION OF STACK:

We can also implement stack using linked list. We perform a push by inserting at the front of the list and perform a pop by deleting at front of the list.

Declaration for linked Stack:

```
        struct node
        {
                data_type       data;
                node * next;
        };
        node *top = NULL;
        node *tepm = NULL;
```

top is the pointer that points to toppest node.

Algorithm for Push

```
        Start:
                Step 1  :       tepm = new node
Step 2  :       if tepm = NULL  then
                                        Print " STACK IS OVERFLOW"
                                        Exit
                                { end of  if  }
                Step 3  :       input val
                Step 4  :       temp->data = val
                Step 5  :       temp->next = top
        Step 6  :       top = temp
    Stop:
```

Let us write a general function in C++ to push an element on the top of the stack.

```
        node *push(node *top, data_type val)
        {
                node temp = NULL;
                temp = new node;
                if(temp == NULL)
                {
                        cout<<"STACK IS OVERFLOW";
                }
else
{
                        temp->data = val;
                temp->next = top;
                        top = temp;
                }
                return(top);
        }
```

Eleventh Period: [Plus One Practical Period]

Algorithm for Pop

In this algorithm we delete a node from the stack after checking whether the stack is empty or not.

```
    Start:
        Step 1  :       if top = NULL  then
                                Print "STACK IS UNDERFLOW"
                        {       end of if       }
        Step 2  :       temp = top;
        Step 3  :       top = top->next;
        Step 4 :        Print "Popped data is ", temp->data
        Step 5  :       delete temp
    Stop:
```

Let us write a general function in C++ to pop elements from the top of the stack.

```
    node *Pop(node * top)
    {
        node *temp;
            if(top == NULL)
                {
                    cout<<"STACK IS UNDERFLOW";
                }
            else
                {
                temp = top;
                top = top->next;
```

```
                    cout<<"Popped element is "<<temp->data;
                    temp->next = NULL;
                    delete temp;
                    }
          return(top);
     }
```

APPLICATIONS OF STACKS:
One of the most important applications of stacks is the evaluation of arithmetic expressions by the computer. An arithmetic expression is a combination of variables and/or constants connected by arithmetic operators and parenthesis. Parenthesis is used to specify the sequence in which the operations are to be performed. The operators used in arithmetic expressions, symbols for their representations, and their hierarchy is listed in Table.
Arithmetic Operator and their symbols.

Sr. No.	Operator	Symbol	Hierarchy
1.	Exponentiation	↑	5
2.	Division	/	4
3.	Multiplication	*	3
4.	Addition	+	2
5.	Subtraction	-	1

An arithmetic expression can be written in any of the following forms:-
A. Infix Notation
B. Postfix or Reverse Polish Notation
C. Prefix or Polish Notation.
A. Infix Notation
The operators given in tale are binary operators. Each binary operator operates on two operands. Suppose, we want to add two operands A and B and assign the result to operand E. here, operand '+' (plus) will operate on operands A and B. In infix notation the expression will be written as

E = A + B. (1)

B. Postfix or Reverse Polish Notation
Here, the arithmetic operator is embedded between the operands on which the operator is to operate. However, in postfix notation the operator is placed after the operands. Equation (1), in postfix notaiton

E = AB +. (2)

C. Prefix or Polish Notation.
In prefix notation the operator will be placed before the operands. The above expression, in prefix notation will be written as

E = +AB (3)

Postfix notation is used for evaluation of arithmetic expressions by the computer. When arithmetic expressions are written in postfix expression, the temporary results evaluated are put in a stack. For evaluation of the final result, it will be required to pup values from top of the stack. In the next section, we will discuss how an infix arithmetic expression can be converted into postfix from. In the subsequent sections, we will describe how a stack can be created and how the final result can be evaluated by popping temporary results stored in the stack.
Any expression from infix form can be converted into postfix or prefix form. The procedure is simple. At any time only operator operates on two operands. If A and B are the operands and + is the

operator, in infix form the expression is written as A + B. in postfix form the same would be written as AB+. In prefix the same would be written as +AB. The same is true for conversion of any complex expression. We will explain with some examples.

EXAMPLE

Convert the following expression into postfix and prefix form

Z = (A + B) * C – D

SOLUTION

The steps for finding the post-fix expression for Z are

(a) Evaluate A + B and store it as T1

T1 = AB+

(b) Evaluate T1*C and store it as T1

T2 = (AB+)*C
  = AB + C*

(c) Evaluate T2-D and assign the result to Z

Z = T2- D
  AB + C * D –

For quick evaluation the same may be written as :-

Z = (A+B)*C-D
= AB+*C-D
= AB+C*-D
= AB+C*D-

The prefix from for expression Z may be found as under :-

Z = (A+B)*C-D
= +AB*C-D
=*+ABC-D
= -*+ABCD

Algorithm for infix to postfix conversion:

Suppose, EXP is expression in infix form which is to be converted in to its equivalent postfix form PX. Now add ';' at end of EXP and push '(' into stack.

Start:
Step 1: repeat Step 2 to Step 5 while ';' is encounter
Step 2: read next symbol
Step 3: if it is an operand, place it on to the expression
Step 4: if it is an operator, repeatedly pop the operators from stack and place it to the expression
till higher priority operator presents Finally place this operator in to stack.
Step 5: if it is right parenthesis, pop all the operators from stack and place it to expression till left parenthesis exists.
Step 6: pop all operators from stack and place it to expression.
Stop:

To see how above algorithm work, let us convert the following infix exp. into postfix exp.

Example : (A + B) * C – D

Symbol (EXP)	Action	Stack	Expression (PX)
(	Push '(' to stack	(, (	NULL
A	Append to Expression	(, (	A

+	Push ‘+’ to Stack	(, (, +	A
B	Append to Expression	(, (, +	A, B
)	Pop ‘+’ form stack and Append to	(,	A, B, +
*	Push ‘*’ to Stack	(, *	A, B, +
C	Append to Expression	(, *	A, B, +, C
–	Pop ‘+’ form stack and place to Exp, then Push ‘-’ to Stack	(, -	A, B, +, C, *
D	Append to Expression	(, -	A, B, +, C, *, D
;	Pop ‘-’ form stack and Append to		A, B, +, C, *, D,

= A, B, +, C, *, D, -

Same can be done by short cut method which is as following:

(A + B) * C – D

Sol :

(A + B) * C – D

= [A, B, +], * C – D

= [A, B, +, C, *] – D

= [A, B, +, C, *, D, –]

Postfix expression : A, B, +, C, *, D, –

Evaluation of Postfix Expression:

The algorithm to evaluate the postfix expression is given below:

Start:

Step 1 : Repeat Step 2 to Step 4 while there are more symbol in EXP.

Step 2 : Read the symbol into S.

Step 3 : If S is operand push it to stack

Step 4 : If S is an operator pop two value from stack into P & Q respectively. Apply S on value P & Q and push result into stack.

Step 5 : Pop the final value which is the result.

Stop:

Example:

Evaluate the postfix expression : 5, 4, +, 10, *, 20, -

Solution:

To solve the above post expression, following steps are taken place. The First two symbols are operands, so push it into stack.

5	4

The next symbol is ‘+’ which is an operator. So pop two operands i.e. 4 and 5 push its result (sum) 9 into stack.

9

Next symbol is 10, push it into stack.

9	10

Next symbol is ‘*’ which is an operator. So pop two operands i.e. 10 and 9 push its result (product) 90 into stack.

90

Next symbol is 20, push it into stack.

90	20	

Next symbol is '-' which is an operator. So pop two operands i.e. 20 and 90 push its result (difference) 70 into stack.

70	

Since no more symbol is there so pop from stack which is the result of expression. i.e. 70.

QUEUE AS DATA STRUCTURES:

The Queue as a data structure is very similar that you can see at various place where many people are waiting in a line for their turn to get the services such as cinema hall, railway reservation counter, ATM, Bank Counters etc . In all the above places discipline is First-cum-first serve basis that is technically called FIFO (First in first out). The Queue size also changes in a dynamic manner that is it can be increased or decrease according to the requirement. in this process insertion is only possible from one end that is called 'rear' and deletion is only possible only at the other end of the Queue that is called 'front'.

However, when a Queue is considered as an array the size of Queue may be fixed and when the queue is considered as a linked list the size of the Queue is variable.

BASIC OPERATIONS AND IMPLEMENTATION OF QUEUE

The basic operations that can be performed on stack are:

- Creation of queue
- Check for empty queue
- Check for full queue
- Add element in queue
- Delete element from queue
- Print stack

A queue is a list, any list implementation can be implemented by using queue. We can implement a queue by using following data structures:

- Array implementation
- Linked list implementation

<u>Array Implementation of Queue:</u>

A. <u>Linear Queue:</u>

The simple way to implement a queue is by using one dimensional array let QUEUE[max] with **max** elements. The first element in the queue will be QUEUE[0], the second at QUEUE[1] and so on. In the array implementation we uses two counters front and rear. To insert an element on the queue is at rear, we increment the rear and set the value at QUEUE[rear] = value. To remove the element from the queue first we obtains the removed value and then increment the front. Definition for sequentially allocated Queue is :

```
data_type  Queue[size];
int front = -1, rear = -1;
```

To check whether a queue is empty or not, we have to check the value of front, if it is less than zero or greater than the value of rear then Queue is empty i.e. called 'QUEUE IS UNDERFLOW'. And to check whether the Queue is full or not we have to check the value of rear, if it is greater than or equals to the maximum size of the Queue then the Queue is full i.e. called 'QUEUE IS OVERFLOW'.

Algorithm To Insert Into Queue:

In this algorithm an element is added in a Queue let QUE[max] which can hold maximum of **max** values.

```
Start:
        Step 1  :       if rear >= MAX  then
                                Print " QUEUE IS OVERFLOW'
                                Exit
                        {       end of  if      }
        Step 2  :       rear = rear +1
        Step 3  :       QUE[rear] = data
Stop:
```

Let us write a general function in C++ to insert an element at the rear of the Queue.

```
        Void insertQ(data_type QUE[ ], int &rear, int data)
        {       if(rear == MAX)
                {       cout<<"QUEUE IS OVERFLOW";
                        return;
}
                rear++;
                QU[rear] = data;
        }
```

Algorithm to delete elements from Queue:

In this algorithm we delete an element from the Queue after checking whether the Queue is empty or not.

```
        Start:
        Step 1  :       if front > rear OR front < 1 then
                                Print " QUEUE IS UNDERFLOW"
                                Exit
                        {       end of if       }
        Step 2  :       data = QU[front]
        Step 3  :       if  front = rear OR front = MAX   than
                                rear = front = 0
                        else
                                front = front + 1
                        {  end of  if     }
Stop:
```

Note: - In step 3 we check either front = rear or front = MAX that means all the elements are deleted from the Queue. So reinitializes the front and rear.

Let us implement a general function in C++ to delete elements from the Queue.

```
        data_type deleteQ(data_type QUE[ ], int &rear)
        {       data_type  data;
                if( front >rear || front < 0)
                {
                        cout<< "QUEUE IS UNDERFLOW";
                        return(-1);
                }
                data = QUE[rear];
                if(front == rear || front == MAX)
                        front = rear = -1;
                else
                        front++;
        return(data);
        }
```

B. <u>Circular Queue:</u>

When queue is implemented using array a situation arises when overflow occurs whenever though the free cells are available. For example, the queue given below can hold 6 elements.

Now, the rear reaches the end. So, queue is full but still there are three free cells available in queue. To avoid this drawback, we can arrange these elements in a circular array. The queue implemented using circular array are called circular queue. In circular queue as soon as a pointer exceeds maximum number of elements it should be reset to 1. So, if you have to add orange in above queue you can insert as a first element:

Algorithm for Add Queue

In this algorithm an element is added into a circular queue, which can hold MAX number of elements:

```
Start
Step 1  if ((rear +1) mod MAX = front then
        Print "Queue overflow"
        Exit
Step 2  rear = (rear +1) mod MAX
Step 3  Queue[rear] = Data
Stop
```

Note: the mod operator gives the reminder.

Let us write a general function in C++ to add an element in the circular queue.

```
        Void add_Q(data type queue[] int front, data_type val, int & rear)
        {       if ((rear + 1) % MAX == front)
                {       cout << "Queue overflow";
                        return;
                }
                        rear = (rear + 1) % MAX;
                        queue[rear] = val;
        }
```

Algorithm to delete from Queue:

In this algorithm we delete an element from queue from front end.

```
Start:
Step 1. If front > rear
        Print "Queue Underflow"
        Exit
        {End of if}
Step 2. Front  = (front + 1) mod MAX
Step 3. Data = Queue[front]s
Stop:
```

Let us write a general function in C++ to add an element in the circular queue.

```
        Void add_Q(data type queue[] int front, data_type val, int & rear)
        {
                if (front>rear)
                {
                        cout << "Queue Underflow";
                        return;
                }
                front = (front + 1) % MAX;
                data=queue[front];
                cout<<"Deleting element is"<<data;
                return;
}
```

Linked List Implementation of Queue

As we see if we implement the queue in array, some times number of memory location remains unused. To overcome this problem, it is batter to implement queue using link list.

In linked list implementation two pointer variables front and rear keep track of the front and rear end of the queue. The linked queue is given below:

Declaration of Linked Queue:

```
        struct node
                        {
                        data_type data;
                node *  next;
                        };
                node *front=NULL, *rear=NULL, *temp=NULL;
```

An element is added at the end of linked list and deleted from the front of linked list.

Algorithm to Add into Queue:

In this algorithm we add a node called **temp** with data value var in to a linked queue. Front points to the first node of the queue and rear points to the last node of the queue.

```
        Statrt:
                Step 1: temp=new node
Step 2: if temp=NULL than
                Print "Queue is over flow"
                Exit
        { end of if}
step 3: temp->data=var
step 4: temp->next=NULL
step 5: if rear=NULL than
                rear=temp
                front=rear
        else
                rear->next=temp
                rear=temp
        { end of loop}
        Stop:
```

Let us write a function in C++ called insertQ, which, add new node in linked Queue having element of type data_type.

```
        node *insertQ( node *rear, data_type var)
        {
                node *temp=NULL;
                temp=new node;
                if (temp==NULL)
                {
                        cout<<"Queue is over flow";
                        return (rear);
                }
                temp->data=var;
                temp->next=NULL;
                if (rear==NULL)
                {
                        rear=temp;
                        front=rear;
                }
                else
                {
                        rear->next=temp;
                        rear=temp;
                }
        return (rear);
        }
```

Algorithm to delete Queue:
In the following algorithm we will delete a node from linked Queue:

```
        Start:
                Step 1: if front=NULL than
                                Print"Queue is under flow"
                                Exit
                        {end of if}
                step 2: temp=front
                step 3: temp->next=NULL
                step 4: front=front->next
                step 5: print "deleting element is ", temp->data
                step 6: delete temp
        Stop
```

Let us write of function in C++ called deleteQ, which delete a node in linked Queue having elements of type data_type.

```
        node *deleteQ ( node *front, data_type &var)
        {
                node *temp;
                if (front==NULL)
                {
                        cout<<"Queue is under flow";
                        var=-1;
                }
                else
                {
                        temp=front;
                        temp->next=NULL;
                front=front->next;
                        cout<<"Deleting element is "<<temp->data;
                        var=temp->data;
                        delete temp;
                }
```

```
return (front);
}
```

Chapter-11 (Database Concepts)

Database : - It is a organized collection of inter-related data.
Database Management System is a software which is used to store, retrieve & update the data from the database.

Important features of DBMS

1. Databases reduces data redundancy i.e data repetition
2. Sharing of data among several different users
3. Data Integrity
4. Data Security i.e protection of data against unauthorized user
5. Data Independence
6. Data Abstraction

Three Levels of Data base Implementation

1. **INTERNAL LEVEL(PHYSICAL LEVEL) :-**This level describes how the data is actually stored on the storage medium. At this level, complex low-level data structures are described in details.
2. **CONCEPTUAL LEVEL**:**-** This level describes what data are actually stored in the database and their relationships
3. **EXTERNAL LEVEL** :**-** This level is concerned with the way in which the data is viewed by the user.

DATABASE MODELS

A data models is a collection of conceptual tools for describing data, data relationships, data semantics etc. There are three models

1. RELATIONAL MODEL
2. NETWORK MODEL
3. HIERARCHICAL MODEL

RELATIONAL MODEL

The relational model represents data and relationships among data by a collection of tables called relation.

NETWORK MODEL

The network model represents data by collections records and relationships among data are represented by links called pointers. The records in the database are organized as collection of arbitrary graphs

HIERARCHICAL MODEL

It is like network model but records are organized as trees rather than arbitrary graphs

DIFFERENT TERMINOLOGY IN RELATIONAL MODEL

TUPLE : The rows of table is known as tuple
ATTRIBUTES: The columns of tables called so
DEGREE : The number of columns in a table is called so
CARDINALITY : The number of rows in table is called so
VIEW / VIRTUAL TABLE : A view is a table that does not really exist but derived from one or more base tables.
Primary Key : It is a set of one or more attributes that can uniquely identify tuples with in the relation.
Candidate Key : All attribute combination inside a relation that can serve as primary key are candidate keys
Alternate key: It is a candidate key which is not primary key is called so.

Normalization

It is the process of transformation of the conceptual schema of the database into a computer representable form. It is decomposition of more complex data structures into relations.

First Normalisation

A relation is in First Normalisation if it contains atomic values and no repeating groups.

Second Normalisation

A relation is in Second Normalisation if it is in 1NF and every non key attributes is fully functionally dependent on each candidate key.

Third Normalisation

A relation is in Third Normalisation if it is in 2NF and every non key attributes is non-transitively dependent on each candidate key.

Chapter-12 (Structured Query Language)

Structured query language (SQL) is a language that enables you to create and operate on relational databases –

Following are the processing capabilities of SQL –

1. data Definition Language
2. Interactive Data manipulation Language
3. Embedded data Manipulation Language
4. Authorization
5. etc.

To cover up various types of data, we have following data types in SQL.
CHAR,
DEC,
INT ,
DATE,
FLOAT etc.

While working with foxpro based SQL we have following generally-used data types-
N for Numeric
C for Character
D for Date

A relational database system deals with huge amount of data. Number of commands are required to create and manipulate the table

We have following commands in SQL-
CREATE TABLE – It is a command to create the structure of a table. Now what is a table ? A table is a collection of data arranged in a row-column form i.e. tabular form.
Suppose we have following table empl
Table : EMPL

Empno	**Name**	**Desig**	**Salary**
1	A	Officer	15000
2	B	Clerk	4000
3	C	Manager	18000
4	D	Officer	12000
5	E	Clerk	7000
6	F	Officer	11000

Following command is used to create the structure of above-mentioned table-

```
CREATE TABLE empl (empno INTEGER, name CHAR(20), desig CHAR(15), salary DECIMAL);
```

After giving this command structure is ready for the above table and it is also ready to fill in the records.

It is a good practice to enter correct and valid data. To ensure it we have to put same conditions over a table and these conditions are called CONSTRAINTS. So a CONSTRAINT is a condition or check applicable on a column(field) or set of columns.

1. **NOT NULL** : If we write a keyword NOT NULL immediately after the data types of a column, this means column can never have empty values.

Eg. CREATE TABLE empl (empno INTEGER, name CHAR(20) NOT NULL , desig CHAR(15), salary DECIMAL);

Now it is necessary to give a name while entering a name, you can not leave name as empty.

2. **UNIQUE** : This constraint ensures no two rows have the same value in the specified column(s)

Eg. CREATE TABLE empl (empno INTEGER NOT NULL UNIQUE, name CHAR(20) NOT NULL , DESIG CHAR(15), SALARY DECIMAL);

Now 2 or more rows of the table can not have same values in empno column

3. **PRIMARY KEY** : This way a column can be declared as the primary key of the table means no 2 rows have the same values for the column defined as primary key. So how it is different from UNIQUE ? Answer is only one column in a table can be declared as primary key whereas in UNIQUE more than one column can also be considered.

Eg. CREATE TABLE empl (empno INTEGER NOT NULL PRIMARY KEY, name CHAR(20) NOT NULL , DESIG CHAR(15), SALARY DECIMAL);

Now empno column can not have same values.

4. **DEFAULT** : A column which contains DEFAULT as a constraint will enjoy a default value in case when no value is provided. Only one value can be set as default.

Eg. CREATE TABLE empl (empno INTEGER , name CHAR(20) NOT NULL , DESIG CHAR(15) DEFAULT = "OFFICER", SALARY DECIMAL);

Now in case of when no value is specified for designation it automatically sets to OFFICER.

6. **CHECK** : This puts condition over the columns.

Eg. CREATE TABLE empl (empno INTEGER, name CHAR(20) , desig CHAR(15), salary DECIMAL CHECK (salary>10000));

It restricts salary column to accept a value more than 10000.

Eg. CREATE TABLE empl (empno INTEGER, name CHAR(20) , desig CHAR(15), salary DECIMAL CHECK (salary>10000));

It restricts salary column to accept a value more than 10000.

Eg. CREATE TABLE empl (empno INTEGER, name CHAR(20) , desig CHAR(15), salary DECIMAL CHECK (salary BETWEEN 10000 AND 20000));

It restricts salary column to accept a value between 10000 and 20000.

Eg. CREATE TABLE empl (empno INTEGER, name CHAR(20) , desig CHAR(15) CHECK (desig IN (“OFFICER”,”CLERK”,”MANAGER”)), salary DECIMAL);

It restricts desig column to accept only one value specified in IN list.So it will accept only one designation from OFFICER, CLERK and MANAGER.

Table constraint : A table constraint works on group of columns.

Eg. CREATE TABLE empl (empno INTEGER, name CHAR(20) , desig CHAR(15) , salary DECIMAL , bonus DECIMAL, CHECK (bonus<salary));

The CHECK constraint given at the end of the table is a table constraint and is responsible to allow only lesser values in the comparison of salary.

SELECT : This command helps us to make queries.
Eg. SELECT * FROM empl;
This displays all the rows of the table with all the columns. * means all the columns get displayed.
Output looks like -

Empno	**Name**	**Desig**	**Salary**
1	A	Officer	15000
2	B	Clerk	4000
3	C	Manager	18000
4	D	Officer	12000
5	E	Clerk	7000
6	F	Officer	11000

Eg. SELECT name, salary FROM empl;
This displays all the rows of the table with name and salary columns only. Output looks like –

Name	**Salary**
A	15000
B	4000
C	18000
D	12000
E	7000
F	11000

Eg. SELCT DISTINCT desig FROM empl;
This displays one particular designation only one time. Output looks like :

Desig
Officer
Clerk
Manager

Removing DISTINCT in above Eg. displays the following output :

Desig
Officer
Clerk
Manager
Officer

Clerk
Officer

WHERE clause : To display particular rows which follow particular condition(s) can be displayed using WHERE clause.

EG. SELECT * FROM empl WHERE salary>12000;
It display all the columns but only those rows which match the condition salary must be greater than 12000. Output is as follows –

Empno	**Name**	**Desig**	**Salary**
1	A	Officer	15000
3	C	Manager	18000

EG. SELECT * FROM empl WHERE salary BETWEEN   10000 and 17000;
It display all the columns but only those rows which match the condition salary must between 10000 and 17000. Output is as follows –

Empno	**Name**	**Desig**	**Salary**
1	A	Officer	15000
4	D	Officer	12000
6	F	Officer	11000

EG. SELECT * FROM empl WHERE desig IN (“Manager”,”Clerk”);
It displays all the columns but only those rows which match the condition designation must be Manager or Clerk. Output is as follows –

Empno	**Name**	**Desig**	**Salary**
2	B	Clerk	4000
3	C	Manager	18000
5	E	Clerk	7000

EG. SELECT * FROM empl WHERE desig LIKE “C”
It displays nothing as output as there is no designation C exactly –

EG. SELECT * FROM empl WHERE desig LIKE “C%”
It displays all those designations which start with letter “C” –

Empno	**Name**	**Desig**	**Salary**
2	B	Clerk	4000
5	E	Clerk	7000

EG. SELECT * FROM empl WHERE desig IS NULL;
It displays nothing as there is no designation NULL.

ORDER BY Clause : Whenever a SELECT Query is executed, the resulting rows emerge in predecided order. We can sort the results or a query in the specific order using ORDER BY clause either in ascending order or descending order.

Eg. SELECT * FROM empl ORDER BY salary;
It displays every record in the ascending order of salary. Output is as follows –

Empno	**Name**	**Desig**	**Salary**
2	B	Clerk	4000
5	E	Clerk	7000

6	F	Officer	11000
4	D	Officer	12000
1	A	Officer	15000
3	C	Manager	18000

Eg. SELECT * FROM empl ORDER BY salary DESC;
It displays every record in the descending order of salary. Output is as follows –

Empno	**Name**	**Desig**	**Salary**
3	C	Manager	18000
1	A	Officer	15000
4	D	Officer	12000
6	F	Officer	11000
5	E	Clerk	7000
2	B	Clerk	4000

EG. SELECT * FROM empl WHERE salary>9000 ORDER BY salary;
It display all the columns but only those rows which match the condition salary must be greater than 9000 and in the ascending order of salary. Output is as follows –

Empno	**Name**	**Desig**	**Salary**
6	F	Officer	11000
4	D	Officer	12000
1	A	Officer	15000
3	C	Manager	18000

Eg. SELECT * FROM empl ORDER BY name DESC;
This displays all the rows of the table with all the columns in the descending order of name . Output looks like –

Empno	**Name**	**Desig**	**Salary**
6	F	Officer	11000
5	E	Clerk	7000
4	D	Officer	12000
3	C	Manager	18000
2	B	Clerk	4000
1	A	Officer	15000

Aggregate Functions : Aggregate functions are used to calculate the summary values of the columns. We have following commonly used aggregate functions in SQL.

AVG – To compute average value.
SUM- To find the total value.
MAX – To find maximum value.
MIN – To find minimum value.
COUNT – To count non-null values in a column.
COUNT(*) – To count total number of rows in a column.
Eg. SELECT SUM(salary) from empl;
It calculates the total salary for all the employees.

Eg. SELECT SUM(salary) from empl WHERE desig=”OFFICER”;
It calculates the total salary for all the OFFICERS.

Eg. SELECT AVG(salary) from empl WHERE desig="OFFICER";
It calculates the average salary for all the OFFICERS only.

Eg. SELECT MAX(salary) from empl;

It calculates the maximum salary.

Desig	SUM(Salary)
Officer	38000
Clerk	11000
Manager	18000

Eg. SELECT COUNT(*) from empl;
It counts the total number of rows in the table empl. So it gives 6 for the table empl as it has only 6 records.

Eg. SELECT COUNT(desig) from empl;
It counts the total number of non-null designation in the table empl. So it gives 6 for the table empl .

Eg. SELECT COUNT(DISTINCT desig) from empl;
It counts the total number of non-null distinct designation in the table empl. So it gives 3 for the table empl.

GROUP BY Clause : The GROUP BY clause is used in SELECT statements to divide the table into groups. Grouping can be done by a column name, or with aggregate functions in which case the aggregate produces a value for each group.

Eg. SELECT desig, SUM(salary) FROM empl GROUP BY desig;
It will display the sum of salary for each group on the basis of designation. Output is as follows :

Eg. To place conditions on the groups HAVING Clause is used.
SELECT desig, SUM(Salary) FROM empl GROUP BY desig HAVING SUM(Salary) >15000;
Following output gets displayed :

Desig	SUM(salary)
Officer	38000
Manager	18000

To create new tables with existing tables : You can create new tables from existing tables. Suppose we want to create a table new with the help of existing table empl containing name and salary of officers only, following command is used –

CREATE TABLE new AS (SELECT name, salary FROM empl WHERE desig="OFFICER");

UNION, INTERSECT, and ***MINUS*** operation : Some rules must be followed with these operations.
1. Both the SELECT statements must be UNION compatible; this means the select lists of SELECT statement must have the same number of columns having similar data types and order.
2. The first query in the UNION statement may contain the INTO clause.
3. The ORDER BY clause can occur only at the end of the UNION statement.

4. The GROUP BY and HAVING clause are allowed within individual queries.

Suppose we have 2 tables EMPL1 and EMPL2 as follows –

Table : EMPL1

Empno	**Name**	**Desig**	**Salary**
1	A	Officer	15000
2	B	Clerk	4000
3	C	Manager	18000
4	D	Officer	12000
5	E	Clerk	7000

Table : EMPL2

Empno	**Name**	**Desig**	**Salary**
3	C	Manager	18000
4	D	Officer	12000
6	F	Officer	11000
7	G	Clerk	7500

Eg. SELECT * FROM empl1 UNION SELECT * FROM empl2;

Common rows of EMPL1 and EMPL2 come only once in the resultant table It gives the following output –

Empno	**Name**	**Desig**	**Salary**
1	A	Officer	15000
2	B	Clerk	4000
3	C	Manager	18000
4	D	Officer	12000
5	E	Clerk	7000
6	F	Officer	11000
7	G	Clerk	7500

Eg. SELECT * FROM empl1 INTERSACT SELECT * FROM empl2;

Common rows of EMPL1 and EMPL2 come only in the resultant table. It gives the following output –

Empno	**Name**	**Desig**	**Salary**
3	C	Manager	18000
4	D	Officer	12000

Eg. SELECT * FROM empl1 MINUS SELECT * FROM empl2;

It displays only those rows which are in table empl1 but not in table empl2. It gives the following output –

Empno	**Name**	**Desig**	**Salary**
1	A	Officer	15000
2	B	Clerk	4000
5	E	Clerk	7000

INSERT command :This is used to add rows or result of a query in a table.

Eg. INSERT INTO empl VALUES (9, “J”,”Manager”,17500);

The above command adds a complete record at the end of the table.

Eg. INSERT INTO empl (Empno, Desig) VALUES (10, ”Clerk”);

The above command adds only employee number and designation at the end of the table.
Same sequence must be followed in VALUES as it is in field list.

Eg. INSERT INTO empl SELECT * from empl2 WHERE salary>10000;
This adds only those records of table empl2 in empl where salary >10000.

DELETE command : It is just opposite to the INSERT command. It deletes records from a table.

Eg. DELETE FROM empl;
This command deletes all the records from the table empl. Only structure of the table is left now.

Eg. DELETE FROM empl WHERE salary < 10000;
This command deletes all those records where salary is less than 10000. Rest of the records are still intact. Now table looks like as follows –

Empno	**Name**	**Desig**	**Salary**
1	A	Officer	15000
3	C	Manager	18000
4	D	Officer	12000
6	F	Officer	11000

UPDATE command :To modify values in a table UPDATE command is used.

Eg. UPDATE empl SET salary =15000;
This command updates the values of column salary to 15000. So it will change the values for all the employees which is sometimes something like loosing the huge amount of data so it must be used very carefully.

Eg. UPDATE empl SET salary =15000 WHERE name="C";
This command updates the values of column salary to 15000 for employee named A.

Eg. UPDATE empl SET salary = salary + salary*.15 WHERE desig = "OFFICER";
This command updates the values of column salary by increasing it 15% for all the officers.

CREATE VIEW command : A table which physically does not exist is called view. A view is created with the help of an existing table. Why views are required? Sometimes we do not require the whole large table rather we want to take a view of it. So it prompts us to make views.

Eg. CREATE VIEW newempl AS SELECT * from empl WHERE salary>11000;
It creates a temporary table (view) named newempl which contains only those records of empl table where salary >11000.

Eg. CREATE VIEW newempl (emno, da) AS SELECT empno, salary*0.6 FROM empl;
It creates a view with two columns employee number and dearness allowance. da is something which is being calculated with the help of salary.

NOTE : SELECT statement used while creating a view can not include following-
(a) ORDER BY Clause

(b) INTO Clause

JOIN : It is used when data are required from more than one tables. The syntax is as follows –

SELECT column list of table1, column list of table2 FROM table1, table 2
WHERE table1.qualified column=table2. qualified column

There are generally three types of joins –
(a) Equi Join
(b) Natural Join
(c) Self Join

ALTER TABLE Command : It is used to change the structure of a table.
Eg. ALTER TABLE empl MODIFY (name CHAR(40));
This modifies the structure by changing the size of field column from 20 to 40.

Eg. ALTER TABLE empl ADD (tax DECIMAL);
This adds the a new column tax.
DROP TABLE and **DROP VIEW** command : Both the commands are responsible for deleting the a table and view respectively.

Eg. DROP TABLE empl;
DROP VIEW abc;

Chapter-13 (Boolean Algebra)

Boolean Constant : True or False (1 or 0) are called Boolean Constant.
Boolean Variable : A Boolean variable is a kind of variable which can take one value out of true or false.
Boolean Expression : It consists of combinations of Boolean constant and variables.

Basic Postulates of Boolean Algebra :
1.
(i) If X is a Boolean variable and if $X \neq 1$ then $X = 0$
(ii) If X is a Boolean variable and if $X \neq 0$ then $X = 1$
2.
OR relation
0.0=0
0.1=1
1.0=1
1.1=1
3.
AND relation
0.0=0
0.1=0
1.0=0
1.1=1
4.
0'=1
1'=0

Principal of duality : Dual of a Boolean expression can be derived as –
(i) Changing + to .
(ii) Changing . to +
(iii) Replacing 0 by 1 and 1 by 0
(+ is OR operation, and . is AND operation)

Basic Theorems :
Properties of 0 and 1: If X is Boolean variable.
(i) 0+X=X
(ii) 1+X=1
(iii) 0.X=0
(iv) 1.X=X
Truth Table proof of (i) 0+X=X

0	X	Result
0	0	0
0	1	1

As columns X and Result are identical.
So 0+X=X is proved.

Likewise rest properties can be proved.

Indempotence Law :Law states

X + X =X

X . X = X

Law of Involution :

(X')' = X

Complementarity Law :

X + X' = 1

X . X' = 0

Commutative Law :

X + Y = Y + X

X . Y = Y . X

Associative Law :

X + (Y + Z) = (X + Y) + Z

X . (Y . Z) = (X . Y) . Z

Distributive Law :

X(Y + Z) = XY + XZ

X + YZ = XY + XZ

Absorption Law :

X + XY = X

X(X + Y)= X

De Morgan's Theorem :

(i) (X + Y)' = X' . Y'

(ii) (X.Y)' = X' + Y'

Proof (i) (X + Y)' = X' . Y'

Suppose in (i) Both sides are equal, therefore it must fulfil complementarity Laws.

Suppose K= X + Y

Then K + K' = 1 ….. (I)

K . K' = 0……(II)

Proving (I)

X'Y' + (X + Y) = 1

=(X + Y) + X'Y' (Ref' Commutative Law)

=(X + Y + X')(X + Y + Y') (Ref' Distributive Law)

=(X + X' + Y)(Y + Y' + X)

=(1 + Y)(1 + X) (Ref' A + A' = 1)

=1.1

=1

Proving (II)

(X'Y').(X + Y) = 0

=X'Y'X + X'Y'Y (Ref' Distributive Law)
=XX'.Y' + YY'.X'
=0.Y' + 0.X' (Ref' A.A' = 0)
=0 + 0
=0
So Both (I) and (II) are proved , Hence (i) is proved.
Similarly (ii) can be proved

Give the dual of following Boolean expressions.

(a)	1+0
(b)	1.0
(c)	X + Y
(d)	X.1

Ans.

(a)	0.1
(b)	0+1
(c)	X.Y
(d)	X.0

(Note : All Boolean variable remain unchanged)

MINTERMS : A minterm is a product of all the literals (with or without the bar within a logic system)

Problem : Convert X + YZ to minterms.
Ans. X + YZ
X.1.1 + 1.Y.Z
X.(Y+Y')(Z+Z') + (X+X').Y.Z (A+A'=1)
XYZ + XY'Z + XYZ' + XY'Z' + (X + X').Y.Z (Distributive law)
XYZ + XY'Z + XYZ' + XY'Z' +XYZ + X'YZ (Distributive Law)
XYZ + XY'Z + XYZ' + XY'Z' + X'YZ
(The above expression is also called Sum of Products)

Problem : Find the minterm designation (Shorthand notation) for X'YZ.
Ans. Taking 1 for non-complemented variable and 0 for complemented variable.
The binary equivalent is 011
Decimal equivalent of 011 = $0*2^2 + 1*2^1 + 1*2^0$
=0 + 2 + 1
= 3
minterm designation is m_3

What is Canonical expression? -
A Boolean expression composed entirely either of minterms or maxterms is referred to as canonical expression.

Problem : Convert X + YZ to maxterms.
Ans. X + YZ
=(X + Y)(X + Z)
=(X + Y)(X + Z) (Distributive Law)
=(X + Y + 0)(X + 0 + Z)
=(X + Y + ZZ')(X + YY' + Z) (AA' = 0)

=(X + Y + ZZ’)(X + Z + YY’)
=(X + Y + Z)(X + Y + Z’) (X + Y + Z)(X + Y’ + Z) (Distributive Law)
=(X + Y + Z)(X + Y + Z’)(X + Y’ + Z) (Common term once)

Problem : Find the maxterm designation (Shorthand notation) for X’+Y+Z.

Ans. Taking 0 for non-complemented variable and 1 for complemented variable.
The binary equivalent is 100
Decimal equivalent of 011 = $1*2^2 + 0*2^1 + 0*2^0$
=4 + 0 + 0
=4
maxterm designation is M_4

Minimization of Boolean expression :

Problem : Reduce X’Y’Z’ + X’YZ’ + XY’Z’ + XYZ’

Ans. X’Y’Z’ + X’YZ’ + XY’Z’ + XYZ’
= X’(Y’Z’ + YZ’) + X(Y’Z’ + YZ’)
=X’(Z’(Y’ + Y)) + X(Z’(Y’ + Y))
=X’(Z’.1) + X(Z’.1)
= X’Z’ + XZ’
=Z’(X’ + X)
=Z’(1) (X’ + X = 1)
=Z’

What is a Karnaugh map? **:** Karnaugh map or K-map is a graphical display of fundamental products in a truth table.

Rules of SOP reduction using K-map :

(1) Prepare the truth table for given function
(2) Draw an empty K-map for the given function (i.e. 2 variable K-map for 2 variable function; 3 variable K-map for 3 variable function; and so on)
(3) Map the given function by entering 1’s for the outputs as 1 in the corresponding squares.
(4) Enter 0’s in all left out empty squares.
(5) Encircle adjacent 1’s in form of octets, quads and pairs. Do not forget to roll the map and overlap.
(6) Remove redundant groups.
(7) Write the reduced expressions for all the groups and OR (+) them.

Problem : Reduce F(A,B,C,D) = (0, 2, 7, 8, 10, 15)

Ans. Mapping the given function in K-map.

AB \ CD	C’D’	C’D	CD	CD’
A’B’	1 0	1	3	1 2
A’B	4	5	1 7	6
AB	12	13	1 15	14
AB’	1 8	9	11	1 10

In above K-map we have

1. Pair (m_7+m_{15}) 2.Quad (m_0+m_2+m_8+m_{10}) (This is a quad after map rolling)

Reduced expression of Pair = BCD
Reduced form of Quad = B’D’
Thus final reduced expression is BCD + B’D’
Logic Gates : We have some more logic gates.
(1) NOR gate – Complemented OR gate is called NOR gate.
(2) NAND gate - Complemented AND gate is called NAND gate.
(3) XOR gate – XOR gate produces 1 for only those input combinations that have odd number of 1’s.

<u>3 input digit Truth Tables for gates</u>

<u>NOR gate :</u>

A	**B**	**C**	**OUTPUT**
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

<u>NAND gate :</u>

A	**B**	**C**	**OUTPUT**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

<u>XOR gate :</u>

A	**B**	**C**	**OUTPUT**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Problem :What is the difference between Half Adder and Full Adder?
Ans : Half Adder adds two binary digits whereas Full Adder adds three binary digits.

Truth table for Half Adder

A	**B**	**SUM(XOR)**	**CARRY(AND)**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth table for Full Adder

A	**B**	**C**	**SUM(XOR)**	**CARRY**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Problem : A Boolean function F defined on three input variables X, Y, and Z is 1 if and only if number of 1 (one) inputs is odd. Draw the truth table for the above function and express it in canonical sum of product form.
Ans.
Truth Table for product terms(3-input)

A	**B**	**C**	**OUTPUT(F)**	**MINTERMS**
0	0	0	0	X'Y'Z'
0	0	1	1	X'Y'Z
0	1	0	1	X'YZ'
0	1	1	0	X'YZ
1	0	0	1	XY'Z'
1	0	1	0	XY'Z
1	1	0	0	XYZ'
1	1	1	1	XYZ

Adding all the minterms (product terms) for which output is 1, we get
X'Y'Z + X'YZ' + XY'Z' + XYZ
This is desired canonical Sum of Product.

Problem : What is encoder?
Ans : An Encoder converts numbers from one number system to another. Like Decimal to Binary, Hexa-decimal to Binary etc
Truth Table for decimal to binary encoder

Decimal Number	F3	F2	F1	F0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Chapter-14 (Communication & Network Concepts)

Network is an interconnected collection of autonomous computers. When more than two networks are connected and the data communication done across these networks is called Internetworking.

Primary goals of Networks are :

- **Resource Sharing :** Networks helps to share software and peripherals in a collection of computers.
- **Reliability :** A file can have different copies on different computers. If one of the computers fails to work, then user can work on the files which are copied on other computers.
- It makes Less expensive by sharing the hardware & software.

Nodes(Workstation) , Server and Network Interface Unit (NUI)

A computer that is attached to a network is called nodes.

Server : - A computer which provides sharing of software and hardware resources on the network is known as Server.

A Network Interface Unit is a device that is attached to each of workstation and the server on the network for communication.

TYPES OF NETWORKS

1. LOCAL AREA NETWORKS(LAN)

These are privately owned network within a single building or campus of a few kilometers in size.

2. METROPOLITAN AREA NETWORKS (MAN)

These are the networks that link computers within a city.

3. WIDE AREA NETWORKS(WAN)

These are the networks spread over large geographical area, often a country or continent.

Comparison of LAN, MAN & WAN

LAN	**MAN**	**WAN**
Diameter of not more than a few km like offices, schools	Spanning a small city or a town.	Span entire countries.
Owned by a single organization	Owned by one or more organization	Owned by multiple organization.
Error rates are much lower	Error rates are lower compare to WAN	Error rates are comparatively very high.
Operates at the speed between 1 and 10 mbps	Operates at the speed of at least several mbps	Operates at the speed of less than 1 mbps

Other Network concepts

BACK BONE NETWORK: A backbone network is a network that is used as a backbone to connect LANs together to form a WAN.

Eg : FDDI(Fiber Distributed Data Interface) is a back bone Network.

CLIENT / SERVER MODEL: The Client/Server network is a network in which a workstation can request information from a computer that can share resources known as server. The workstation that request information is called client or front-end. The server is also called as Back-end which contains data.

HOST : A computer which provides general user services over the network is called host computer or host machine or end system.

FULL CONNECTIVITY: When in a network each host is connected to other directly , then the network is said to be Fully Connected.

Network switching Techniques

Switching techniques are used to provide communication between two computers. There are :

1. <u>Circuit switching</u>

In this technique, initially complete physical connection between two computers is established, then the data will be transmitted. (Eg. Telephone system)

2. <u>Message switching</u>

In this technique, source computer sends data to the receiving end via switching office. The switching office to another switching office, if the link is free.

3. <u>Packet switching</u>

In this technique, the data will be converted into fixed size of packet and transmitted .

NETWORK TOPOLOGY

The pattern of interconnection of nodes in a network is called the topology. Different types of topologies are:-

1. BUS OR LINEAR TOPOLOGY

It consists of a single length of the transmission medium onto which the various nodes are attached. Ethernet is a very widely used bus topology LAN.

Advantages :

- Short cable length : It allows a very short length to be used for networking
- A new node can be added to the network without disturbing the other nodes.

Disadvantages:

- Low reliability : a break or short circuit in the cable can halt the whole network.
- Data collision : When a message from two nodes comes on the line, a collision occurs.

2. STAR TOPOLOGY

In this topology each workstation is directly connected to a central node. Any communication between the workstations must pass through the central node.

Advantages :

- If a node stops working, the system continues uninterrupted.
- Networking management is easy.

Disadvantages:

- If the hub (central node) crashes, whole of network fails.
- More cabling is needed as compared to bus/ ring topology.

3. CIRCULAR OR RING TOPOLOGY or LOOP NETWORK

In this topology each node is connected to two and only two neighboring nodes. The data is transmitted, circulate along the ring. After a data packet travels a full circle, it is removed at the source node.

Advantages :

- One node can transmit data at a time, so collision of data does not occur.
- Short cable length : It requires less amount of cabling comparatively

Disadvantages:

- If one of the Nodes fails to work, the entire network has failed.
- Network reconfiguration is difficult : It is not possible to shut down even one or two nodes for reconfiguration.

4. TREE or HIBRID TOPOLOGY

It is a modified form of bus topology. The shape of the network is that of an inverted tree with the central root branching and sub branching to the intermediates of the network.

Advantages :

- It is easy to extend the network.
- It is easy to isolate defective nodes without disturbing the other nodes.

Disadvantages:

- If the root node fails to work , the entire network fails.
- Low reliability : a break or short circuit in the cable can halt the branches of network.

5. GRAPH TOPOLOGY

In this technology, nodes are connected together in a random fashion. A link can connect two or more nodes.

Different Access Methods

Access methods are used to control the use of the cabling system by devices to share the cable for communication. These are :-

1. **CSMA/CD (Carrier Sense Multiple Access/Collision Detection)** : - In CSMA/CD, each and every node listens before transmitting on the network and will only send the message, if the network is free. It will retransmit the data, if there is a collision.
2. **Token Passing** : In this access is controlled by passing the token from one node to another.

DATA COMMUNICATION DEVICES

These are devices is used to provide an interconnection between the computer and the communication channel like cables.

The various Communication Devices are:

1.	MODEM
2.	REPEATER
3.	BRIDGE
4.	ROUTER
5.	GATEWAY

MODEM (Modulation Demodulation)

It is a device used as an interface between a computer and a communication channel or media. It converts data from digital to analog signals and vice versa.

REPEATER: It is a device that amplifies a signal being transmitted on the network. It is used in long network lines, which exceed the maximum rated distance for a single run.

BRIDGE: It is a device used to link two similar networks. A bridge is used to connect LANs of same type.

ROUTER: It is a device used to link two networks which are handling different protocols. A router can connect LAN to mainframe network.

Eg : Netware V3.11, Cisco 1600

Difference between Router and Bridge is that the router uses logical addresses(protocols) for connection but Bridge uses physical addresses.

GATEWAY:

It is a device used to link two dissimilar networks. A gateway works at higher level than routers because it can connect two or more completely different networks for communication. Eg: Web gateways

APPLICATIONS OF NETWORK or INTERNET and ITS CONCEPTS OR SEVICES OF INTERNET

1. **E-mail**

It is a popular way of communication on the internet. By which user can send and receive mails to any part of the world. It is one of the cheap and faster way of communication compare to conventional Mail.

2. **FTP** : It is a tool / protocol used for transfer of files between two or more computers. The FTP is used to download or upload files in the internet.

3. **REMOTE LOGIN OR TELNET :**

Remote login means to login remote computer without any network. Telnet is an application used to login to one computer on the internet from another. Telnet connection uses the Internet and not the telephone network.

4 **WWW (WORLD WIDE WEB)** : - It is a set of protocol used to access any documents on the internet through Uniform Resource Address (URL) . It is also a graphical , user friendly interface to the internet called Browser. It allows the users to browse information across the internet.

5 **NETWORK FILE SYSTEM (NFS)**

It is a utility which tells how the files are organized and retrieved on it. It depends upon the operating system being used by the network. Eg : Network NFS v1.2 is a popular NFS software that integrates UNIX system with the resources and files system of the network.

6. **Lynx :-** It is a full screen browser for internet.

7. **Mosaic** : It is used for accessing internet. It has a simple windows interface for network navigation.

8. **UseNet :** In internet, Usenet is the way to meet people and share information. Usenet newsgroup is a special group set up by people who want to share current topics and other information.

9. **Gopher** : - It is a menu oriented web browser. It helps to search and retrieve information on the internet.

10. **ARCHIE** : - It is a tool for accessing files on internet. It is an effective tool on internet for locating files that are stored on FTP servers.

11. **WAIS (WIDE AREA INFORMATION SERVICES)**

WAIS is a networked information retrieval system. This system can search all documents containing a particular keyword using internet database.

12. **Search Engine** : This is software which can search for information on the internet on various websites and web pages across the world.

Eg : YAHOO, HOTBOT etc.

13. **ISP(INTERNET SERVICE PROVIDER) :** - It is a company which offers various options and packages to the general public for internet access. Eg: VSNL, MTNL

14 **. URL(Uniform Resource Locator) :** - The technique used to address documents on the web is called the URL eg : http://www.yahoo.com

15. **WEB PAGE , WEB SITE & Home Page**: - A Web page is an HTML document that is stored on a web server . A collection of web pages belonging to a particular person or organization is called WEB PAGE . eg : http://www.yahoo.com . The very first page of a website is known as Home Page.

16. **ELECTRONIC COMMERCE (E-COMMERCE):** E-COMMERCE refers to the buying and selling of products and services over the internet. It is merely an online business activity including shopping, banking, investment etc.

17. **VIDEO CONFERENCING :-** means two or more people hear and see each other, share whiteboard and share other applications.

<u>DATA SECURITY or (Need for File Protection)</u>
DATA SECURITY refers to ensuring that the data don't get used by unauthorized people and remains in original form. To ensure data security or protection various measures are taken like :

1. Identification
Every authorized user of a given computer is given a personal identity code called User-id . The user-id will be verified while log-in time i.e. starting time of the computer for user identification.

2. Authentication(password- protection)
Authentication is the access control process wherein a valid user's checking is established by asking and comparing the password.

3. Authorization of Files
It is an act of giving authority to a user to become an authorized user of the system. The authorization can give to a program or files to do certain process on that like reading/ writing/ execution.

<u>FILE SECURITY</u>
It refers to ensuring the data / contents in the file don't get used by unauthorized people and remain in original form.

<u>FILE PROTECTION</u>
File protection refers to the measures taken against unauthorized access to the file.

<u>DIFFERENT WAYS TO DO FILE ACCESS RESTRICTION</u>
1. USER AUTHORISATION
2. FILE ACCESS PERMISSION (FAP)

After authorization and authentication, file access are controlled by the file access permissions . FAP that grants permission to user do certain operation on a file like read / write / execute. If a user has only read operation on a file, then it is not possible to delete or modify the file.

<u>SOFTWARE PIRACY</u>
It refers to illegal usage of software. If software is copied millions of times and sold to various unauthorized users, this would be known as software piracy.

<u>COMPUTER VIRUS</u> : - It is malicious software which damages the contents on disk and the data will be damaged.

FIRE WALL : It is a system which reinforce information security between two networks.

COMPUTER COMMUNICATION & DATA SECURITY
<u>COMPUTER COMMUNICATION</u>
COMMUNICATION is the transfer of data from one computer to the other.
<u>DIFFERENT MODES OF DATA TRANSMISSION / COMMUNICATION</u>
<u>1.DIGITAL & ANALOG TRANSMISSION/COMMUNICATION</u>
Digital Communication uses digital signals for communication. A digital signal is a sequence of voltage pulses which represents binary form(Bits).

Analog Communication uses analog signals for transmission. Analog Signal consists of continuous variable electrical waves. It uses general purpose communication channels like telephone lines.

<u>2 Parallel & Serial Communication</u>

In Parallel communication the data is transmitted using multiple wires with each wire carrying each bit.
In Serial communication the bits are sent one after another in a series along the same wire.

1. Synchronous and Asynchronous communication
In synchronous transmission characters are transmitted as group of bits, preceded and followed by control characters
In asynchronous transmission each character is transmitted separately preceded by a start bit and followed by a stop bit.

- Bandwidth : It is the capacity of a medium to transmit a signal.
- Base band Modulation which carries digital signal for transmission at single frequency.
- Broadband Modulation which carries Radio frequency signal for transmission at different frequencies.

<u>DIFFERENT COMMUNICATION CHANNELS / MEDIA</u>
Communication channels means the connecting cables that link various computers.
1. <u>TWISTED PAIR Cables</u>: This cable consists of two insulated copper wires twisted in a spiral pattern. These are used for medium range telephone communication.
<u>Advantages :</u> It is an inexpensive transmission media.
It can carry signals over long distances without the use of repeaters.
<u>Disadvantages:</u> Problems can occur due to differences in the electrical characteristics between pairs.
2. <u>COAXIAL CABLES</u>: It consists of a central copper wire surrounded by one or more protective covering. It is expensive than twisted pair cable but perform better. It is widely used for television signal.
<u>Advantages :</u> It provides a cheap means of transporting multi-channel television signals around metropolitan areas.
<u>Disadvantages :</u> It is expensive than twisted pair cable.
3. <u>Optical Fibers Cables:</u> It consists of a thin strand of light conducting glass or plastic fibers surrounded by protective coverings.
<u>Advantages :</u> It has very high speed of data transmission.
It offers low bit error rates and free from electrical noise and interferences.
It provides increased data security.
<u>Disadvantages</u> :
It is an expensive communication medium.
It has a limited number of direct connections on a given length of a cable.
<u>4. Microwave:</u> Microwave signals are used to transmit data and voice over long distances through air without using cables.
<u>Advantages :</u> It can be used for communication when it is not possible to communicate through grounds.
<u>Disadvantages:</u> It is an expensive communication medium.
<u>5. Radio-wave:</u> It uses specific radio frequencies for direct data communication with out using cables.
<u>Advantages :</u> It is extremely useful for customer services like home deliveries.
<u>disadvantages :</u> It has lack of data security.
It is not good for transfer of large files.
<u>6. Satellite:</u> It is also a cable less communication channel for extremely large distance via satellite.

Advantages : It can be used for communication when it is not possible to communicate through grounds.
Disadvantages: It is an expensive communication medium.

DIFFERENT COMMUNICATION PROTOCOLS
A protocol is a set of rules for communication. It governs data format, timing, sequencing, access control and error control required to initiate and maintain communication.
Different types of protocols are :

1. HTTP(Hyper Text Transfer Protocol)
It is a protocol used for transfer of hyper text between two or more computers.

2. FTP (File Transfer Protocol)
It is a protocol used for transfer of files between two or more computers.

3. TCP/IP(Transmission Control Protocol / Internet Protocol)
It is a native protocol for internet.It is a protocol for interconnecting networks or devices on same network. TCP is responsible for making sure that the data get through to the receiving end. If the receiving end fails to get data , then TCP will retransmits the data that did not get through.
Internet Protocol(IP) : - It provides service to TCP by sending the data to the destination computer.

4. PPP (Point to Point Protocol)
PPP allows a computer to use the TCP/IP protocol and to be connected directly to the Net.

5. SMTP (Simple Mail Transfer Protocol)
It is a protocol that enables electronic mail to move smoothly through the internet.

6. POP (Post Office Protocol)
It is used to retrieve E-mail from a mail server.

Communication Software/Packages
Communication packages are the software which helps the users to communicate across office or across the internet.
Most popular communication Packages are :
- CC: Mail
- Procomm Plus

ASCII Codes:

Number	Character	Number	Character	Number	Character
0 to 31	Control characters	64	@	96	`
32	space	65	A	97	a
33	!	66	B	98	b
34	"	67	C	99	c
35	#	68	D	100	d
36	$	69	E	101	e
37	%	70	F	102	f

38	&	71	G	103	g
39	'	72	H	104	h
40	(	73	I	105	i
41	)	74	J	106	j
42	*	75	K	107	k
43	+	76	L	108	l
44	,	77	M	109	m
45	-	78	N	110	n
46	.	79	O	111	o
47	/	80	P	112	p
48	0	81	Q	113	q
49	1	82	R	114	r
50	2	83	S	115	s
51	3	84	T	116	t
52	4	85	U	117	u
53	5	86	V	118	v
54	6	87	W	119	w
55	7	88	X	120	x
56	8	89	Y	121	y
57	9	90	Z	122	z
58	:	91	[	123	{
59	;	92	\	124	|
60	<	93	]	125	}
61	=	94	^	126	~
62	>	95	_	127	Delete
63	?				