COMPUTER SCIENCE C++(083)

ASSIGNMENT BOOKLET

CLASS XII

Academic Session: 2017-18

INDEX

Class XII

S.No.	Topic	Page #
1.	Month wise breakup of syllabus	3
2.	Concepts of Object Oriented Programming	4
3.	Function Overloading	5
4.	Classes and Objects	6
5.	Assignment No 1	10
6.	Constructors and Destructors	12
7.	Assignment No 2	14
8.	Inheritance	15
9.	Assignment No 3	18
10.	Arrays	20
11.	Assignment No 4	27
12.	Pointers	28
13.	Assignment No 5	36
14.	File Handling	39
15.	Assignment No 6	45
16.	Linked Lists, Stacks and Queues	48
17.	Assignment No 7	56
18.	Boolean Algebra	57
19.	Assignment No 8	63
20.	Communication and Networking Concepts	65
21.	Assignment No 9	74
22.	Structured Query Language	76
23.	Assignment No 10	89
24.	Program List	95
25.	Practical Guidelines	98
26.	Instructions for printing Practical file and Project Report	99

Computer Science Syllabus
Class XII

March

Object Oriented Programming
Function Overloading

April

Classes and Objects
Constructors and Destructors
Inheritance: Extending Classes

May

Data File Handling
Project Allocation

July

Arrays Revision
Pointers
Linked Lists, Stacks & Queues

August

Program List Completion
First Term Examination

September

Structured Query Language
Boolean Algebra

October

Concepts of Networking

November

Revision

December

Second Term Examination

Object Oriented Programming In C++

Handout I– OOPS Concepts

Programming Paradigm: it defines the methodology of designing and implementing programs using the key features and building blocks of a programming language.

Procedural Programming: In procedural programming paradigm, the emphasis is on doing things i.e., the procedure or the algorithm. The data takes the back seat in procedural programming paradigm. Also, this paradigm does not model real world well.

Object Oriented Programming: The object oriented programming paradigm models the real world well and overcomes the shortcomings of procedural paradigm. It views a problem in terms of objects and thus emphasizes on both procedures as well as data. It follows a bottom up approach in program design and emphasizes on safety and security of data.

Important Concepts in OOPS:

Data Abstraction:

It refers to the act of representing essential features without including the background details. Example: For driving , only accelerator, clutch and brake controls need to be learnt rather than working of engine and other details.

Data Encapsulation:

It means wrapping up data and associated functions into one single unit called class.

A class groups its members into three sections: public, private and protected, where private and protected members remain hidden from outside world and thereby helps in implementing data hiding.

Modularity:

The act of partitioning a complex program into simpler fragments called modules is called as modularity. It reduces the complexity to some degree.

It creates a number of well defined boundaries within the program.

Inheritance:

Inheritance is the process of forming a new class from an existing class or base class. The base class is also known as parent class or super class.

Derived class is also known as a child class or sub class. Inheritance helps in reusability of code , thus reducing the overall size of the program.

Polymorphism:

Poly means many and **morphs** mean form, so polymorphism means one name multiple forms. It is the ability for a message or data to be processed in more than one form.

C++ implements Polymorphism through Function Overloading, Operator overloading and Virtual functions.

RECTANGLE()	TRIANGLE()	CIRCLE()
Area(rectangle)	Area(triangle)	Area(circle)

Advantages of OOPS:

1. Reusability of code
2. Ease of fabrication and maintenance.
3. Ease of comprehension.
4. Easy redesign and extension.

Handout II –Implementing Polymorphism in C++ through Function Overloading

C++ allows you to specify more than one definition for a **function** name in the same scope, which is called **function overloading**.

An overloaded declaration is a declaration that had been declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation).

When you call an overloaded **function**, the compiler determines the most appropriate definition to use by comparing the argument types you used to call the function with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function is called **overload resolution**.

You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. **You can not overload function declarations that differ only by return type**.

Following is the example where same function **print()** is being used to print different data types:

```
#include <iostream.h>
void print(int i) {
      cout << "Printing int: " << i << endl;
void print(double f) {
      cout << "Printing float: " << f << endl;              }
void print(char* c) {
      cout << "Printing character: " << c << endl;       }
int main(void)
{
  print(5);              // Call print to print integer
  print(500.263);  // Call print to print float
  print("Hello C++");              // Call print to print character
   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Printing int: 5
Printing float: 500.263
Printing character: Hello C++
```

Handout III – Classes and Objects in C++

Classes:

The classes are the most important feature of C++ that leads to Object Oriented programming. Class is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating instance of that class.

The variables inside class definition are called as data members and the functions are called member functions.

For example: Class of birds, all birds can fly and they all have wings and beaks. So here flying is a behavior and wings and beaks are part of their characteristics. And there are many different birds in this class with different names but they all posses this behavior and characteristics.

Similarly, class is just a blue print, which declares and defines characteristics and behavior, namely data members and member functions respectively. And all objects of this class will share these characteristics and behavior.

A **Class** is a group of similar objects that share common characteristics and relationships. It acts as a blueprint for defining objects.

It is a collection of variables, often of different types and its associated functions.

Class just binds data and its associated functions under one unit there by enforcing encapsulation.

Classes define types of data structures and the functions that operate on those data structures.

Declaration/Definition:

A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations.

```
class class_name {
        access_specifier_1:
            member1;
        access_specifier_2:
            member2;
        ...
        } object_names;
```

Where class_name is a valid identifier for the class, object_names is an optional list of names for objects of this class. The body of the declaration can contain members that can either be data or function declarations, and optionally access specifiers.

Note: the default access specifier is private.

```
Example: class Box {    int a;
                        public:
                                double length; // Length of a box
                                double breadth; // Breadth of a box
                                double height; // Height of a box
                                 };
```

Access specifiers in Classes:

Access specifiers are used to identify access rights for the data and member functions of the class. There are three main types of access specifiers in C++ programming language: private, public, protected.

Private	Class members declared as private can be used only by member functions and friends (classes or functions) of the class.
Protected	Class members declared as protected can be used by member functions and friends (classes or functions) of the class. Additionally, they can be used by classes derived from the class.
Public	Class members declared as public can be used by any function.

Why Access Specifiers?

Access control helps prevent you from using objects in ways they were not intended to be used. Thus it helps in implementing data hiding and data abstraction.

Objects:

Class is mere a blueprint or a template. No storage is assigned when we define a class. Objects are instances of class, which holds the data variables declared in class and the member functions work on these class objects. Objects are basic run time entities in an object oriented system.

An **object** is an identifiable entity with some characteristics and behavior.
For example: Dogs have characteristics (name, color, breed) and behavior (barking, fetching, wagging tail). Cars also have characteristics (number of doors, number of seats, type of engine, fuel type, average performance) and behavior (moving forward, moving backward, turning, changing gear, applying brakes). Identifying the characteristics and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming. These real-world observations all translate into the world of object-oriented programming.

Software objects are conceptually similar to real-world objects: they too consist of state and related behavior. An object stores its state in fields (variables in some programming languages) and exposes its behavior through functions.

Creating object / defining the object of a class:

The general syntax of defining the object of a class is:-

Class_name object_name;

In C++, a class variable is known as an object. The declaration of an object is similar to that of a variable of any data type. The members of a class are accessed or referenced using object of a class.

```
Box  Box1;          //Declare   Box1  of  type  Box
Box  Box2;          //Declare   Box2  of  type  Box
```

Both of the objects Box1 and Box2 will have their own copy of data members.

Accessing / referencing members of a class:

All member of a class are private by default. Private member can be accessed only by the function of the class itself. Public member of a class can be accessed through any object of the class. They are accessed or called using object of that class with the help of dot operator (.).

The general syntax for accessing data member of a class is:-

Object_name.Data_member=value;

The general syntax for accessing member function of a class is:-

Object_name. Function_name (actual arguments);

The dot ('. ') used above is called the **dot operator or class member access operator**. The dot operator is used to connect the object and the member function. The private data of a class can be accessed only through the member function of that class.

Class methods definitions (Defining the member functions):

Member functions can be defined in two places:-

1. **Outside the class definition**

The member functions of a class can be defined outside the class definitions. It is only declared inside the class but defined outside the class. The general form of member function definition outside the class definition is:

Return_type Class_name:: function_name (argument list)

{ Function body }

Where symbol :: is a **scope resolution operator**.

The **scope resolution operator (::)** specifies the class to which the member being declared belongs, granting exactly the same scope properties as if this function definition was directly included within the class definition.

```
class sum
{ int A, B, Total;
public:
        void getdata ();
        void display ();        };
void sum:: getdata ()        // Function definition outside class definition Use of :: operator
{ cout<<" \n enter the value of A and B";
cin>>A>>B; }
void sum:: display ()        // Function definition outside class definition Use of :: operator
{ Total =A+B;
cout<<"\n the sum of A and B="<<Total; }
```

2. **Inside the class definition**

The member function of a class can be declared and defined inside the class definition.

```
class sum
{       int A, B, Total;
public:
        void getdata ()
                { cout< "\n enter the value of A and B"; cin>>A>>B;}
        void display ()
                { total = A+B;cout<<"\n the sum of A and B="<<total;} };
```

Passing an Object as an Argument to a member function:

```
/*C++ PROGRAM TO PASS OBJECT AS AN ARGUMEMT. Adds the two heights given in ft and inches. */
#include<iostream.h>
#include<conio.h>
class height
{  int  feet,  inches;
public:
        void getht(int f,int i)
        {       feet=f; inches=i;          }
        void putheight()
        {       cout<<"\nHeight is:"<<feet<<"feet\t"<<inches<<"inches"<< endl;      }
        void sum(height a, height b)
        {       height n;
                n.feet = a.feet + b.feet;
                n.inches = a.inches + b.inches;
                if(n.inches >=12)
                        { n.feet++;
                        n.inches = n.inches -12;}
                cout<< endl<< "Height is: "<< n.feet<< " feet and "<< n.inches<< "inches" <<endl;}
        };
void main()
  {     height h,a,d;
        clrscr();
        h.getht(6,5); a.getht(2,7);
        h.putheight(); a.putheight();
        d.sum(h,a); getch();      }
  /**********OUTPUT**********
  Height is: 6 feet 5 inches
  Height is: 2 feet 7 inches
  Height is: 9 feet and 0 inches
```

What is the difference between struct and classes in C++:

In C++, a *structure* is a class defined with the struct keyword. Its members and base classes are public by default. A class defined with the class keyword has private members and base classes by default. This is the only difference between struct and classes in C++.

Inline Functions:

Inline functions definition starts with keyword inline. The compiler replaces the function call statements with the function code itself (expansion) and then compiles the entire code. They run little faster than normal functions as function calling overheads are saved. A function can be declared inline by placing the keyword inline before it.

```
inline void square(int a)
{          cout<<a*a;              }
void main()
{
 square(4);                // These two statements are translated to:
 square(8);                //{cout<<4*4;} and {cout<<8*8;} respectively as
 }}              //the function is inline, its call is replaced by its body.
```

Assignment No. 1

Q 1. Define classes. Give one example.

Q 2. What do you understand by instance of a class. Give one example.

Q 3. What is the need of object oriented programming?

Q 4. Differentiate between Procedural programming and object oriented programming.

Q 5. Define data encapsulation. How is it implemented in C++?

Q 6. Define data hiding. How is it implemented in C++?

Q 7. Define data abstraction. How is it implemented in C++?

Q 8. Define polymorphism. How is it implemented in C++?

Q 9. Differentiate between public, private and protected data members.

Q 10. What do you understand by access specifiers?

Q 11. Differentiate between Inline Functions and Non Inline Functions. Give a suitable example using C++ code to illustrate the same.

Q 12. Define a class in C++ with the following description:

- A data member ***Flight number*** of type integer
- A data member ***Destination*** of type string
- A data member ***Distance*** of type float
- A data member ***Fuel*** of type float
- A member function **CALFUEL( )** to calculate the value of fuel as per the flowing criteria.

Distance	**Fuel**
<= 1000	500
More than 1000 and <= 2000	1100
More than 2000	2200

Public Members

- A function **FEEDINFO( )** to allow to enter values for flight number, destination, Distance & call function **CALFUEL( )** to calculate the quantity of fuel
- A function **SHOWINFO( )** to allow user to view the content of all the data members

Q 13. Given the following code fragment:

```
#include<iostream.h>
# include<conio.h>
class X
        { int x,y;
        void count(void);
        public :
                int a,b;
                void getval(int, int);
```

```
            void prval(void);
 }; //X's member function's definition

X O1;
void main( )
{  class Y
      {    int i,j;
           void abc( );
      public :
           float k;
           void get( );
            void prin( );
        };
Y O2;
X O3;
}  //end of main( )

void func1( )
{  X O4;
   :    }
void func2( ) { : }
```

Which all members (data & functions) of classes X and Y can be accessed by main(), func1(), func2( ) **?**

Q 14. A class '**time**' has the following members:

Data Members:

Hour of type int
Minute of type int

Member Function:

Readtime(int h, int m);
Showtime( );
Addtime(time T1, time T2);

Write a program using classes to input two different objects FT and ST, print their sum(assuming 24 hours clock time) e.g.,

INPUT :

FT= 6 hrs 35 minutes ST= 3 hrs 45 minutes

OUTPUT :

T = FT+ST
= 10 hrs 20 minitues.

Q 15. What are static class members? Explain with a suitable example.

Q 16. What are nested classes?

Handout IV: Constructors and Destructors

CONSTRUCTORS :

A member function with the same name as its class is called a Constructor and it is used to initialize an object of that class with legal initial values. It is fired automatically as soon as an object of the given class is created. It need not be called explicitly. **But it must be placed inside the public section of the class definition.**

Example : class student

```
{       int rollno;
        float marks;
  public:
        student( )   //Constructor
        { rollno=0; marks=0.0; }
                //other public members
};
```

TYPES OF CONSRUCTORS:

1. Default Constructor:

A constructor that accepts no parameter is called the Default Constructor. If you don't declare a constructor or a destructor, the compiler makes one for you. The default constructor and destructor take no arguments and do nothing. Default constructor provided by the compiler initializes the object data members to default value.

```
class Cube
{           int side;
        public:
           Cube()              //default constructor
            { side=10; }
};
int main()
{ Cube c;
    cout << c.side;
}
```

OUTPUT: 10

2. Parameterized Constructors:

A constructor that accepts parameters is known as Parameterized Constructor, also called as Regular Constructor. Using this Constructor you can provide different values to data members of the objects, by passing the appropriate values as argument.

```
class Cube
{
        int side;
public:
            Cube(int x)
            { side=x; }
};
int main()
{ Cube c1(10), c2(20), c3(30);
cout << c1.side; cout << c2.side; cout << c3.side; }
```

OUTPUT : 10 20 30

3. <u>Copy Constructors</u>:

The copy constructor is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. The copy constructor is used to:

Initialize one object from another of the same type. Copy an object to pass it as an argument to a function. Copy an object to return it from a function.

A copy constructor is a constructor of the form **classname(classname &).**The compiler will use the copy constructors whenever you initialize an instance using values of another instance of the same type.

Copying of objects is achieved by the use of a copy constructor and an assignment operator.

Note: ***The argument to a copy constructor is passed by reference, the reason being that when an argument is passed by value, a copy of it is constructed. But the copy constructor is creating a copy of the object for itself, thus, it calls itself. Again the called copy constructor requires another copy so again it is called. In fact it calls itself again and again until the compiler runs out of the memory .so, in the copy constructor, the argument must be passed by reference.***

<u>DESTRUCTORS:</u>

A destructor is also a member function whose name is the same as the class name but is preceded by tilde("~"). It is executed automatically by the compiler when an object is destroyed.

Destructors are usually used to deallocate memory and do other cleanup for a class object and its class members when the object is destroyed.

A destructor is called for a class object when that object passes out of scope or is explicitly deleted.

Example :

```
class TEST
{    int Regno,Max,Min,Score;
   public:
              TEST( )          // Default Constructor
                    {}
              TEST (int Pregno,int Pscore)          // Parameterized Constructor
             {         Regno = Pregno ;
                       Max=100;
                       Max=100;
                       Min=40;
                       Score=Pscore;                    }
               ~ TEST ( )              // Destructor
            {          cout<<"TEST Over"<<endl;        }
};
```

Note:

Constructors and destructors **do not have return type, not even void** therefore they cannot return values.

The compiler automatically calls constructors when creating class objects and calls destructors when class objects go out of scope.

Constructors can be overloaded.

Assignment No. 2
Constructors and Destructors

Q1. What are constructors and destructors? Give an example to illustrate the use of both.
Q2. Define Constructor overloading. Give an example.
Q3. Define Copy Constructor. Give an example.
Q4. Answer the questions (i) and (ii) after going through the following class :

```
class Test
    {       char Paper[20];
            int Marks;
    public :
            Test()                          //Function 1
                    { strcpy(Paper, "Computer");
                    Marks = 0;}
            Test(char P[])                  //Function 2
                     {strcpy(Paper, P);
                     Marks = 0;}
            Test(int M)                     //Function 3
                      {strcpy(Paper, "Computer");
                      Marks = M;}
            Test(char P[],int M)            //Function 4
                       {strcpy(Paper, P );
                       Marks =M;}
    };
```

i. Which feature of Object Oriented Programming is demonstrated using Function 1, Function 2, Function 3, Function 4 in the above class Test?
ii. Write statements in C++ that would execute Function 2 and Function 4 of class Test.

Handout V: Inheritance

Inheritance is the process by which new classes called *derived* classes are created from existing classes called *base* classes.
The derived classes have all the features of the base class and the programmer can choose to add new features specific to the newly created derived class.
The idea of inheritance implements the **IS A** relationship. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.

Features or Advantages of Inheritance:

- Reusability of Code
- Saves Time and Effort
- Faster development, easier maintenance and easy to extend
- Capable of expressing the inheritance relationship and its transitive nature which ensures closeness with real world problems .

Base & Derived Classes:
A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes. A class derivation list names one or more base classes and has the form:

class derived-class: visibility-mode base-class

Where visibility mode is one of **public, protected,** or **private**.
For example, if the *base* class is *MyClass* and the derived class is sample it is specified as:

class sample: public MyClass.

The above makes sample have access to both *public* and *protected* variables of base class *MyClass*.
Consider a base class **Shape** and its derived class **Rectangle** as follows:

```
#include <iostream.h>
class Shape            // Base class
{  public:
    void setWidth(int w)
    {              width = w;  }
    void setHeight(int h)
    {              height = h;  }
   protected:
    int width;  int height;
};
class Rectangle: public Shape // Derived class
{  public:
    int getArea()
    {              return (width * height);  }
};
int main(void)
{ Rectangle Rect;
  Rect.setWidth(5);
  Rect.setHeight(7);
  cout << "Total area: " << Rect.getArea() << endl; return 0;
  }
```

When the above code is compiled and executed, it produces the following result:

```
Total area: 35
```

Modes of Inheritance:

When deriving a class from a base class, the base class may be inherited through **public, protected** or **private** inheritance. The type of inheritance is specified by the visibility mode as explained above. We hardly use **protected** or **private** inheritance, but **public** inheritance is commonly used. While using different type of inheritance, following rules are applied:

Public Inheritance: When deriving a class from a **public** base class, **public** members of the base class become **public** members of the derived class and **protected** members of the base class become **protected** members of the derived class. A base class's **private** members are never accessible directly from a derived class, but can be accessed through calls to the **public** and **protected** members of the base class.

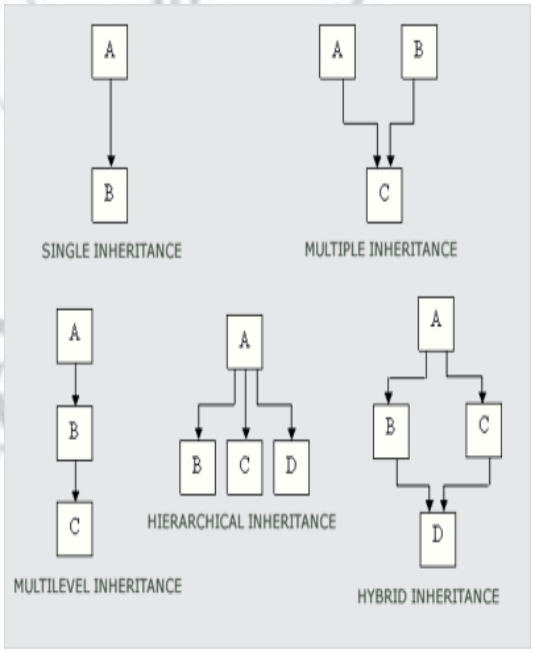
Protected Inheritance: When deriving from a **protected** base class **public** and **protected** members of the base class become **protected** members of the derived class.

Private Inheritance: When deriving from a **private** base class, **public** and **protected** members of the base class become **private** members of the derived class.

We can summarize the different access types according to who can access them in the following way:

Access	public	protected	private
Same class	Yes	Yes	Yes
Derived classes	Yes	Yes	No
Outside classes	Yes	No	No

A derived class inherits all base class methods with the following exceptions: Constructors, destructors and copy constructors of the base class, Overloaded operators of the base class, The friend functions of the base class.



Types of Inheritance:

1. **Single class Inheritance:** Single inheritance is the one where you have a single base class and a single derived class.

2. **Multilevel Inheritance:** When the Derived class itself is used to derive another class.

3. **Multiple Inheritance:** In Multiple inheritances, a derived class inherits from multiple base classes. It has properties of both the base classes.

4. **Hierarchical Inheritance:** In hierarchical Inheritance, it's like an inverted tree. So multiple classes can inherit from a single base class.

5. **Hybrid Inheritance:** It combines two or more forms of inheritance .In this type of inheritance, we

can have mixture of number of inheritances but this can generate an error of using same name function from no of classes, which will bother the compiler to how to use the functions.

a. Therefore, it will generate errors in the program. This has known as ambiguity or duplicity.

b. Ambiguity problem can be solved by using **virtual base classes.**

Multiple Inheritance Example:

A C++ class can inherit members from more than one class and here is the extended syntax:

class derived-class: visibility-mode baseA, visibility-mode baseB....

Where visibility-mode is one of **public, protected,** or **private** and would be given for every base class and they will be separated by comma as shown above. Let us try the following example:

```
#include <iostream.h>
class Shape      {          // Base class Shape
  public:
   void setWidth(int w)
   {        width = w;
    }
   void setHeight(int h)
   {        height = h;
    }
protected:
   int width;
   int height;
};
class PaintCost             // Base class PaintCost
{
  public:
   int getCost(int area)
   {        return area * 70;
     }
};
class Rectangle: public Shape, public PaintCost // Derived class
 { public:
     int getArea()
     {              return (width * height);  }
};
int main(void)
{         Rectangle Rect; int area;
          Rect.setWidth(5); Rect.setHeight(7);
          area = Rect.getArea();
          cout << "Total area: " << Rect.getArea() << endl; // Print the area of the object.
          cout << "Total paint cost: $" << Rect.getCost(area) << endl; // total cost of painting
          return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Total area: 35
Total paint cost: $2450
```

Assignment No. 3
Inheritance in C++

Q1. What is the need of inheritance in OOPS?

Q2. Differentiate between Abstract class and concrete class. Give one example of each.

Q3. What do you understand by visibility modes?

Q4. Define the following :

a. Single Level Inheritance
b. Multi level Inheritance
c. Multiple Level Inheritance.

Give one example of each.

Q5. Consider the following and answer the questions that follow :

```
class School
{   int A;
    protected :
    int B, int C;
    public :
        void INPUT ( int t);
        void OUTPUT ( );
};
class Dept : protected School
{   int X,Y;
    protected :
                void IN(int, int);
    public    :
                void OUT ( );
};
class Teacher : public Dept
{   int P;
    void DISPLAY( );
    public :
                void ENTER( );
};
```

i) Name the Base class and Derived class of class Dept.
ii) Name the data members that can be accessed from function OUT( ).
iii) Name the private member function(s) of class Teacher.
iv) Is the member function OUT( ) accessible by the objects of Dept ?
v) How many bytes will be required by an object of class School and an object of class Teacher respectively.

Q6. Consider the following and answer the questions given below.

```
class University {
int NOC;
 protected:
     char name[25];
public:
     University() { };
     char State[25];
     void Enterdata();
     void Displaydata();
};
```

```
class College : public University
{          int NOD;
         protected:
           void Affiliation();
         public:
           College() { }
           void Enrol();
           void Show();
  };
class Department : public College
    {  char Dname[25];
       int NOf;
      protected:
        void Affiliation();
      public:
        Department() {}
        void Input();
         void Display();      };
```

i) Which class's constructor will be called first while declaring object of class Department?

ii) Which kind of inheritance is being depicted in the above example?

iii) If class college was derived privately from university, then, name members that could be directly accesses through the object(s) of class Department.

iv) How many bytes does an object belonging to the class Department require?

v) Which is the base class and the derived class of class College.

Q7. Identify the **syntax error**(s), if any (give reason for error) :

```
Class ABC
{  int  x  =
     10;
     float y;
     ABC ( ) { y=5; }
     ~ABC ( ) { }
};
void main( )
{     ABC a1, a2;
}
```

Handout VI: Arrays

In Computer Science, a **data structure** is a particular way of storing and organizing data in a computer so that it can be used efficiently. Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, Stacks are used in function call during execution of a program, while B-trees are particularly well-suited for implementation of databases. The data structure can be classified into following two types:

Simple Data Structure: These data structures are normally built from primitive data types like integers, floats, characters. For example arrays and structure.

Compound Data Structure: simple data structures can be combined in various ways to form more complex structure called compound structures. Linked Lists, Stack, Queues and Trees are examples of compound data structure.

Data Structure Arrays

Data structure array is defined as linear sequence of finite number of objects of same type with following set of operations:

1. Creating : defining an array of required size
2. Insertion: addition of a new data element in the in the array
3. Deletion: removal of a data element from the array
4. Searching: searching for the specified data from the array
5. Traversing: processing all the data elements of the array
6. Sorting : arranging data elements of the array in increasing or decreasing order
7. Merging : combining elements of two similar types of arrays to form a new array of same type In C++ an array can be defined as

Datatype arrayname[size];

Where size defines the maximum number of elements can be hold in the array.

For example

```
float b[10];                    //b is an array which can store maximum 10 float values
int c[5];
```

Array initialization

```
void main()
{       int b[10]={3,5,7,8,9};
    cout<<b[4]<<endl;
    cout<<b[5]<<endl;
}
```

Output is: 9 0

In the above example the statement int b[10]={3,5,7,8,9} assigns first 5 elements with the given values and the rest elements are initialized with 0. Since in C++ index of an array starts from 0 to size-1 so the expression b[4] denotes the 5th element of the array which is 9 and b[5] denotes 6th element which is initialized with 0.

Searching

We can use two different search algorithms for searching a specific data from an array

1. Linear search algorithm
2. Binary search algorithm

Linear search algorithm

In Linear search, each element of the array is compared with the given item to be searched for. This method continues until the searched item is found or the last item is compared.

```
#include<iostream.h>
int linear_search(int a[], int size, int item)
{
   int i=0;
   while(i<size&& a[i]!=item)
      i++;
   if(i<size)
   return i;              //returns the index number of the item in the array else
   return -1;          //item not present in the array so it returns -1 since -1 is not a legal index no.
}
void main()
{
    int  b[8]={2,4,5,7,8,9,12,15},size=8;
    int item;
    cout<<"enter a number to be searched for";
    cin>>item;
    int p=linear_search(b, size, item);          //search item in the array b
     if(p==-1)
         cout<<item<<" is not present in the array"<<endl;
    else
         cout<<item <<" is present in the array at index no "<<p;
}
```

In linear search algorithm, if the searched item is the first element of the array then the loop terminates after the first comparison (best case), if the searched item is the last element of the array then the loop terminates after size time comparisons (worst case) and if the searched item is middle element of the array then the loop terminates after size/2 time comparisons (average case). For large size array linear search not an efficient algorithm but it can be used for an unsorted array.

Binary Search algorithm

Binary search algorithm is applicable for already sorted array only. In this algorithm, to search for the given item from the sorted array (in ascending order), the item is compared with the middle element of the array. If the middle element is equal to the item then index of the middle element is returned, otherwise, if item is less than the middle item then the item is present in first half segment of the array (i.e. between 0 to middle-1), so the next iteration will continue for first half only, if the item is larger than the middle element then the item is present in second half of the array (i.e. between middle+1 to size-1), so the next iteration will continue for second half segment of the array only. The same process continues until either the item is found (search successful) or the segment is reduced to the single element and still the item is not found (search unsuccessful).

```
#include<iostream.h>
int binary_search(int a[ ], int size, int item)
{
    int first=0,last=size-1,middle; while(first<=last)
    {
    middle=(first+last)/2;
    if(item==a[middle])
       return middle;        // item is found else if(item<a[middle])
```

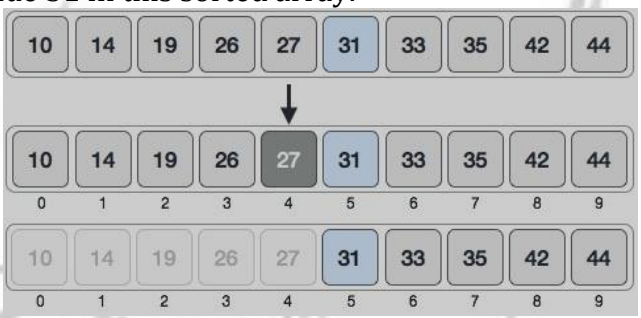
```
            last=middle-1; //item is present in left side of the middle element
          else
            first=middle+1; // item is present in right side of the middle element
          }
          return -1; //given item is not present in the array, here, -1 indicates unsuccessful search
       }
void main()
      {
        int b[8]={10,14,19,26,31,33,35,42,44},size=10;
        int item;
        cout<<"enter a number to be searched for";
         cin>>item;
        int  p=binary_search(b, size, item); //search item in the array b
        if(p==-1)
              cout<<item<<" is not present in the array"<<endl;
        else
              cout<<item <<" is present in the array at index no "<<p;
      }
```

Using the above algorithm, let us search for a value 31 in this sorted array.

First, we shall determine half of the array by using this formula –

mid = low + (high - low) / 2

Here it is, 0 + (9 - 0) / 2 = 4 (integer value of 4.5). So, 4 is the mid of the array.



Now we compare the value stored at location 4, with the value being searched, i.e. 31. We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the latter half of the array.

We change our low to mid + 1 and find the new mid value again.

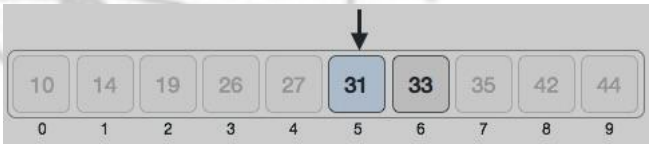
low = mid + 1

mid = low + (high - low) / 2

Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.

The value stored at location 7 is not a match, rather it is less than what we are looking for. So, the value must be in the lower part from this location.

Hence, we calculate the mid again. This time it is 5. We compare the value stored at location 5 with our target value. We find that it is a match.



We conclude that the target value 31 is stored at location 5.

Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

Sorting in Arrays

Sorting is nothing but storage of data in sorted order, it can be in ascending or descending order. The term Sorting comes into picture with the term Searching. Sorting arranges data in a sequence which makes searching easier.

We can use different sort algorithms for sorting a data in an array:

1. Bubble Sort
2. Selection Sort
3. Insertion Sort

Bubble Sort

This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

We take an unsorted array for our example.

Bubble sort starts with very first two elements, comparing them to check which one is greater.

Now, value 33 is greater than 14, so it is already in sorted locations.

Next, we compare 33 with 27.

We find that 27 is smaller than 33 and these two values must be swapped.

The new array should look like this –

Next we compare 33 and 35. We find that both are in already sorted positions.

Then we move to the next two values, 35 and 10.

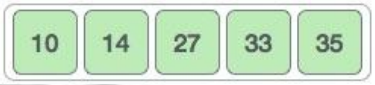
We know then that 10 is smaller 35. Hence they are not sorted.

We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this –

To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this-

Notice that after each iteration, at least one value moves at the end.

And when there's no swap required, bubble sorts learns that an array is completely sorted.



Function for Bubble sort

```
#include<iostream.h>
void BubbleSort(int a[],int size)
{   int temp;                                   // Bubble Sort Starts Here
    for(int i=0; i<=size; i++)
    {     for(int j=0; j<size; j++)
          {   if(a[j]>a[j+1])                   //Comparing adjacent elements
                    {         temp=a[j];                    //Swapping element in if statement
                              a[j]=a[j+1];
                              a[j+1]=temp;    }
          }
  }
}
```

Selection Sort

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

Consider the following depicted array as an example.

For the first position in the sorted list, the whole list is scanned sequentially. The first position where 14 is stored presently, we search the whole list and find that
10 is the lowest value. So we swap 14 with 10.

After one iteration 10, which happens to be the minimum value in the list, appears in the first position of the sorted list.

For the second position, where 33 is residing, we start scanning the rest of the list in a linear manner. We find that 14 is the second lowest value in the list and it should appear at the second place. We swap these values.

After two iterations, two least values are positioned at the beginning in a sorted manner.
The same process is applied to the rest of the items in the array. Following is a pictorial depiction of the entire sorting process –

Function for Selection Sort

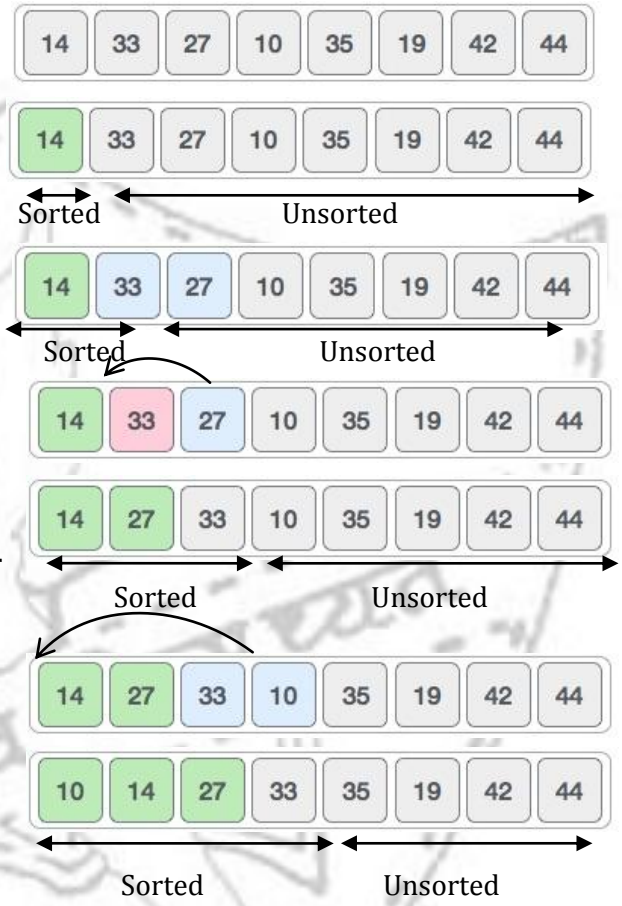
```
void selectionSort(int a[], int size)
{    int i, j, min, temp;
     for(i=0; i < size-1; i++ )
     {       min = i;                    //setting min as i
             for(j=i+1; j < size; j++)
             {       if(a[j] < a[min])
          //if element at j is less than element at min position
               {            min = j;      } //then set min as j
              }
  temp = a[i];
  a[i] = a[min];
  a[min] = temp;}         }
```

Insertion Sort

Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort.

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array).

We take an unsorted array for our example.



Initially the first element alone comprises the sorted array and hence is sorted by itself. For now, 14 is in sorted sub-list.

Insertion sort moves ahead and tries to find the correct place for the next element 33 which is to the right of 14. Hence no movement.

Next iteration 27 is the current element and its correct position is between 14 and 33.

It shifts 33 right and moves 27 between 14 and 33 hence, the sorted sub-list remains sorted after swapping.

Next, it finds the correct position for 10 which will be at the left of 14.

We shift 33, 27 and 14 right and insert 10 right in the beginning. So, by the end of third iteration, we have a sorted sub-list of 4 items.

Function for Insertion Sort

```
void insertionSort(int a[], int size)
{    int i, j ,temp;
     for (i = 1; i <=size; i++)
       {    j = i-1;
            temp=a[i];
            while (j >= 0 && temp< a[j])          //shift elements right till you find a smaller element
                 {       a[j+1] = a[j];
                         j--;
                  }
            a[j+1]=temp;
        }}
```

Merging two sorted arrays

This involves creating a third array from two sorted arrays which contains all their elements in sorted order. Both initial arrays are read simultaneously and compared and the smaller elements are placed

in the resultant array first. This process goes on till one of the arrays has been processed completely and then the remaining elements of the second array are copied.

Function to merge two sorted arrays

```
void Merge(int A[], int B[], int C[], int N, int M, int &K)
{
    int I=0, J=0;
    K=0;                          //Initialisation of counters for A, B, and C
    while (I<N && J<M)
    {
        if (A[I]<B[J])
            C[K++]=A[I++];
        else if (A[I]>B[J])
            C[K++]=B[J++];
        else
        {
            C[K++]=A[I++];
            J++;
        }
    }
    for (int T=I;T<N;T++)
        C[K++]=A[T];
    for (T=J;T<M;T++)
        C[K++]=B[T];
}
```

Assignment No : 4
Arrays

Q 1. Define the term data structure?

Q 2. List out four important operations associated with linear data structure.

Q 3. What is the pre condition for Binary Search to be performed on a single dimensional array?

Q 4. An array Val[1..15][1..10] is stored in memory with each element requiring 4 bytes of storage if the base address of array VAL is 1500, determine the location of VAL[12][9] when the array VAL is stored

i. Row Wise
ii. Column Wise

Q 5. An array S[40][30] is stored in the memory along the row with each of its element occupying 2 bytes, find out the memory location for the element S[15][5], if an element S[20][10] is stored to the memory location 5500.

Q 6. Write a function in C++ to combine the contents of two equi- sized arrays A and B by computing their corresponding elements with the formula 2*A[I]+3*B[I], where value I varies from 0 to N-1 and transfers the resultant content in the third same sized array.

Q 7. Write a user defined function in C++ to merge the contents of two sorted arrays A & B into third array C. Assume array A is sorted in ascending order, B is sorted in descending order, the resultant array is required to be in ascending order.

Q 8. Write a function which accepts an integer array and its size as arguments/ parameters and assign the elements into a two dimentional array of integers in the following format :

If the array is : 1, 2, 3, 4 , 5, 6

The resultant 2 D array is given below :

1 2 3 4 5 6
1 2 3 4 5 0
1 2 3 4 0 0
1 2 3 0 0 0
1 2 0 0 0 0
1 0 0 0 0 0

If the array is : 1, 2, 3

The resultant 2 D array is given below :

1 2 3
1 2 0
1 0 0

(**Hint** : *Use dynamic memory allocation to give memory to 2 D array*)

Handout VII: Pointers

A **pointer** is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it.

The general form of a pointer variable declaration is:
type *ptr-name;

Here, **type** is the pointer's base type; it must be a valid C++ type and **ptr-name** is the name of the pointer variable. The (*) asterisk you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer.

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

The **&** operator is used to assign the address of a variable to a pointer.
For eg: int a;
int *ptr=&a; //assigns the address of integer a to ptr which is an integer pointer.

Memory Allocation in a program: Memory is allocated in two ways for a C++ program:
Static Memory Allocation : The amount of memory to be allocated is known in advance and it allocated during compilation, it is referred to as Static Memory Allocation.
e.g. int a; // This will allocate 2 bytes for a during compilation.

Dynamic Memory Allocation : The amount of memory to be allocated is not known beforehand rather it is required to allocated as and when required during runtime, it is referred to as dynamic memory allocation.

C++ offers two operators for DMA – new and delete.
e.g int *x =new int; float *y= new float; // dynamic allocation
delete x; delete y; //dynamic deallocation

C++ Memory Map :

1. **Program Code** : It holds the compiled code of the program.
2. **Global Variables** : They remain in the memory as long as program continues.
3. **Stack** : It is used for holding return addresses at function calls, arguments passed to the functions, local variables for functions. It also stores the current state of the CPU.
4. **Heap** : It is a region of free memory from which chunks of memory are allocated via DMA functions.

Free Store : It is a pool of unallocated heap memory given to a program that is used by the program for dynamic memory allocation during execution.

Pointer Arithmetic: The arithmetic operations defined on pointers are addition, subtraction, increment and decrement.

When we increment or decrement a pointer variable it starts pointing to the next or previous location of the same datatype.

e.g. int *p; p++;

If current address of p is 100, then p++ statement will increase p to 102, not 101.

If *c is char pointer and *p is integer pointer then:

char pointer	c	c+1	c+2	c+3	c+4	c+5	c+6	c+7
Address	100	101	102	103	104	105	106	107
int pointer	p		p+1		p+2		p+3	

Adding 1 to a pointer actually adds the size of pointer's base type.

Creating Dynamic Array:

Syntax : pointer-variable = new data-type [size];

e.g. int * array = new int[10];

Note: array[0] will refer to the first element of array, array[1] will refer to the second element.

No initial values can be specified for arrays.

Array sizes must be supplied when new is used for array creation.

Two dimensional array :

```
int *arr, r, c;
r = 5; c = 5;
arr = new int [r * c];
Now to read the element of array, you can use the following loops:
for (int i = 0; i < r; i++)
{
        cout << "\n Enter element in row " << i + 1 << " : ";
        for (int  j=0; j < c; j++)
                cin >> arr [ i * c + j];
}
```

Pointers and Arrays:

C++ treats the name of an array as **constant pointer** which contains base address i.e address of first location of array. Therefore Pointer variables are efficiently used with arrays for declaration as well as accessing elements of arrays, because an array is a continuous block of memory locations and therefore pointer arithmetic can help to traverse in the array easily.

```
void main(){
        int *m;
        int marks[10] ={ 50,60,70,80,90,80,80,85,75,95};
        m = marks; // address of first location of array  or we can write it as m=&marks[0]

        for(int i=0;i<10;i++)
        cout<< *m++;
          // or
```

```
        m = marks; // address of first location of array  or we can write it as m=&marks[0]
        for(int i=0;i<10;i++)
        cout<< *(m+i);
}
```

Array of Pointers :

To declare an array holding 10 int pointers –

int * ip[10];

Memory would be allocated for 10 pointers that can point to integers.

Now each of the pointers, the elements of pointer array, may be initialized. To assign the address of an integer variable phy to the fourth element of the pointer array, we have to write ip[3] = & phy;

Now with *ip[3], we can find the value of phy.

int *ip[5];

Ip--->	Index	0	1	2	3	4
	address	1000	1002	1004	1006	1008

int a = 12, b = 23, c = 34, d = 45, e = 56;

	Variable	a	b	c	d	e
	address	1050	1065	2001	2450	2725
	Value	12	23	34	45	56

ip[0]=&a; ip[1]=&b; ip[2] = &c; ip[3] = &d; ip[4] = &e;

	Index	ip[0]	ip[1]	ip[2]	ip[3]	ip[4]
	Array ip value	1050	1065	2001	2450	2725
	address	1000	1002	1004	1006	1008

ip is now a pointer pointing to its first element of array ip. Thus ip is equal to address of ip[0], i.e. 1000 *ip (the value of ip[0]) = 1050

* (* ip) = the value of *ip = 12
* (ip+3) = * * (1006) = * (2450) = 45

Pointers and Strings :

Pointers can be used very effectively to handle strings too.

E.g :

```
void main()
    {char str[] = "computer";
    char *cp;
    cp=str;
    cout<<str ;   //display string
    cout<<cp;    // display string
    for (cp =str; *cp != '\0'; cp++) // display character by character by character
    cout << "--"<<*cp;       // pointer arithmetic
```

```
str++;    // Invalid  because str is an array and array name is constant pointer
cp++;     // Valid because pointer is a variable
cout<<cp;}
```

Output :

Computer
Computer
--c--o--m--p--u--t--e--r
omputer

An array of char pointers is very useful for storing multiple strings in memory.
char *subject[] = { "Chemistry", "Physics", "Maths", "CS", "English" };

In the above given declaration subject[] is an array of char pointers whose element pointers contain base addresses of respective subject names. That is, the pointer element subject[0] stores the base address of string "Chemistry", the pointer element subject[1] stores the above address of string "Physics" and so on.

An array of pointers makes more efficient use of available memory by consuming lesser number of bytes to store the string.

An array of pointers makes the manipulation of the strings much easier. One can easily exchange the positions of strings in the array using pointers without actually touching their memory locations.

Pointers and CONST :

A constant pointer means that the pointer in consideration will always point to the same address. Its address cannot be changed.

A pointer to a constant refers to a pointer which is pointing to a symbolic constant. Look the following example:

```
int m = 20;                          // integer m declaration
int *p = &m;                         // pointer p to an integer m
++ (*p);                             // ok : increments int pointer p
int * const c = &n;                  // a const pointer c to an intger n
++ (* c);                            // ok : increments value pointed at by c i.e. its contents
++ c;                                // wrong : pointer c is const – address can't be modified
const int cn = 10;                   // a const integer variable cn
const int *pc = &cn;                 // a pointer to a const int
++ (* pc);                           // wrong : int * pc is const – contents can't be modified
++ pc;                               // ok : increments pointer pc
const int * const cc = *k;           // a const pointer to a const integer
++ (* cc);                           // wrong : int *cc is const
++ cc;                               // wrong : pointer cc is const
```

Pointers and Functions :

A function may be invoked in one of two ways :

1. call by value 2. call by reference

The second method call by reference can be used in two ways :

1. by passing the references 2. by passing the pointers

Reference is an alias name for a variable.

For ex :

```
int m =23;
int &n = m;
int *p;
p = &m;
```

Then the value of m i.e. 23 is printed in the following ways :

```
cout <<m;       // using variable name
cout << n;      // using reference name
cout << *p;     // using the pointer
```

Invoking Function by Passing the References :

When parameters are passed to the functions by reference, then the formal parameters become references (or aliases) to the actual parameters to the calling function.

That means the called function does not create its own copy of original values, rather, it refers to the original values by different names i.e. their references.

For example the program of swapping two variables with reference method :

```
#include<iostream.h>
void main()
{       void swap(int &, int &);
        int a = 5, b = 6;
        cout << "\n Value of a :" << a << " and b :" << b;
        swap(a, b);
        cout << "\n After swapping value of a :" << a << "and b :" << b;
}
void swap(int &m, int &n)
{       int temp; temp = m;
        m = n;
        n = temp;
}
```

output :

Value of a : 5 and b : 6

After swapping value of a : 6 and b : 5

The above program can be re-written using pointers also:

Invoking Function by Passing the Pointers:

When the pointers are passed to the function, the addresses of actual arguments in the calling function are copied into formal arguments of the called function.

That means using the formal arguments (the addresses of original values) in the called function, We can make changes in the actual arguments of the calling function. For eg:

```
#include<iostream.h>
void main()
{       void swap(int *m, int *n);
        int a = 5, b = 6;
        cout << "\n Value of a :" << a << " and b :" << b;
        swap(&a, &b);
        cout << "\n After swapping value of a :" << a << "and b :" << b;          }

void swap(int *m, int *n)
{       int temp;
        temp = *m;
        *m = *n;
        *n = temp;      }
```

Input :
Value of a : 5 and b : 6
After swapping value of a : 6 and b : 5

Function returning Pointers:

The way a function can returns an int or a float, it can also return a pointer. The general form of prototype of a function returning a pointer would be
Type * function-name (argument list);

```
#include    <iostream.h>
int *min(int &, int &); void main()
{       int a, b, *c;
        cout << "\nEnter a :";          cin >> a;
        cout << "\nEnter b :";          cint >> b;
        c = min(a, b);
        cout << "\n The minimum no is :" << *c;          }
int *min(int &x, int &y)
{       if (x < y )
                return (&x);
      else
              return (&y);      }
```

Objects as Function arguments:

Objects are passed to functions in the same way as any other type of variable is passed.

When it is said that objects are passed through the call-by-value, it means that the called function creates a copy of the passed object.

A called function receiving an object as a parameter creates the copy of the object without invoking the constructor. However, when the function terminates, it destroys this copy of the object by invoking its destructor function.

If you want the called function to work with the original object so that there is no need to create and destroy the copy of it, you may pass the reference of the object. Then the called function refers to the original object using its reference or alias.

Also the object pointers are declared by placing in front of a object pointer's name.
Class-name * object-pointer;

Eg. Student *stu;

The member of a class is accessed by the arrow operator (->) in object pointer method.
Eg :

```
#include<iostream.h>
class Point
{       int x, y;
public :
        Point()     {x = y = 0;}

        void getPoint(int x1, int y1)
        {x = x1; y = y1; }
        void putPoint()
        {cout << "\n Point : (" << x << ", " << y << ")";}
};
void main()
{
point p1, *p2;
cout << "\n Set point at 3, 5 with object";
p1.getPoint(3,5);
cout << "\n The point is :";
p1.putPoint();
p2 = &p1;
cout << "\n Print point using object pointer :";
p2->putPoint();
cout << "\n Set point at 6,7 with object pointer";
p2->getPoint(6,7);
cout<< "\n The point is :";
p2->putPoint();
cout << "\n Print point using object :";
p1.getPoint();}
```

If you make an object pointer point to the first object in an array of objects, incrementing the pointer would make it point to the next object in sequence.

```
student stud[5], *sp;
---
sp = stud;              // sp points to the first element (stud[0])of stud
sp++;                   // sp points to the second element (stud[1]) of stud
sp + = 2;               // sp points to the fourth element (stud[3]) of stud
 sp--;                  // sp points to the third element (stud[2]) of stud
```

You can even make a pointer point to a data member of an object. Two points should be considered :

1. A Pointer can point to only public members of a class.
2. The data type of the pointer must be the same as that of the data member it points to.

this Pointer :

In class, the member functions are created and placed in the memory space only once. That is only one copy of functions is used by all objects of the class. Therefore if only one instance of a member function exists, how does it come to know which object's data member is to be manipulated?

Every object in C++ has access to its own address through an important pointer called this pointer. The **this** pointer is an implicit parameter to all member functions. Therefore, inside a member function, **this** may be used to refer to the invoking object.

Friend functions do not have a this pointer, because friends are not members of a class. Only member functions have a **this** pointer.

For eg:

```
#include <iostream.h>
 class Box {
   public:
     Box(double l = 2.0, double b = 2.0, double h = 2.0)          // Constructor definition
{     cout <<"Constructor called." << endl;
       length = l;
       breadth = b;
       height = h;        }

     double Volume()
{      return length * breadth * height;       }

     int compare(Box box)
{    return this->Volume() > box.Volume();     }

   private:
     double length;              // Length of a box
     double breadth;             // Breadth of a box
     double height;              // Height of a box
};

int main(void)
{  Box Box1(3.3, 1.2, 1.5);              // Declare box1
     Box Box2(8.5, 6.0, 2.0);            // Declare box2

   if(Box1.compare(Box2)) {
     cout << "Box2 is smaller than Box1" <<endl;
   } else {
     cout << "Box2 is equal to or larger than Box1" <<endl;
   }
return 0;}
```

When the above code is compiled and executed, it produces the following result:

```
Constructor called.
Constructor called.
Box2 is equal to or larger than Box1
```

Assignment No : 5
Pointers

Q 1. What are Pointers? What is its use?
Q 2. What is the difference between :
- a) Arrays and Pointers
- b) Static and Dynamic memory allocation.
- c) **new** and **delete** operators.

Q 3. What is the use of following operators : *, &,-> ?
Q 4. What do you understand by this pointer?
Q 5. What do you understand by Free Pool of memory?
Q 6.What are Memory Leaks. What are the possible reasons for it.
Q 7. Rewrite the following program after removing error(s), if any. Underline each correction.

a)
```
#include<iostream.h>
struct triangle
{
        int a,b,c;
}
void input (triangle *T)
{
        T.  a  -
        =5; T. b
        /=2;  T.
        c      *=
        0.5;
}
void display (triangle T)
{
        cout<<T. a << " : ";
        cout<<T. b << " : ";
        cout<< T.c<< " \n ";
}
void main ( )
{
        triangle A = 10, 9, 9 ;
        input(A);
        display(A);
        input(A);
        display(A);
         input(A);
        display(A);
}
```

b)
```
Void main( )
{ const int I; I = 20;
        const int*ptr = &i ; (*ptr++);
        int j = 15; ptr = &j;
}
```

Q 8.Find the output of the following :

a)
```
# include <iostream.h>
# include<conio.h>
```

```
void main ( )
{
    clrscr( );
    void func( int, int *);
    int a[5] = {2, 4, 6, 8,
    10}; int i, b = 5;
    for (i = 0; i<5; i ++)
    {
    func(a[i], &b);
    cout <<a[i] << "\t"<< b<< "\n";
    }
}
void func( int p, int *q);
{ p = *q +=2; }
```

b)
```
# include<iostream.h>
void main ( )
{
        int array [ ] = {2, 3, 4, 5 };
        int *aptr=array;
        int value = *aptr ;
        cout<<value<< "\n";
        value = *aptr++ ; cout<<value<< "\n";
        value = *aptr ; cout<<value<< "\n";
        value = *++aptr ; cout<<value<< "\n";
}
```

c)
```
# include<iostream.h>
void main( )
{
    int arr [ ] = {4, 6, 10, 12}; int * ptr = arr;
    for (int i = 1; i<= 3; i++)
    {
        cout<<*ptr << "#"; ptr++;
    }
    for (i = 1; i<= 2; i++)
    {
        (*ptr)* = 3;
        cout<<*ptr; - -ptr ;
    }
    for (i = 0; i< 4; i++) cout<<arr[i] << "@";
    cout<<endl;
}
```

d)
```
void main()
{ int x [] = {10, 20, 30, 40, 50};
 int *p, **q, *t;
  p = x; t = x+1; q = &t;
  cout<<*p<< "," <<**q<< "," << *t++;
}
```

e)
```
# include<iostream.h>
```

```
# include<string.h>
class state
{
        char *s_name;
        int size;
public:
        state( ){size =0;s_name = new car [size+1]; }
        state( char *s)
        {       size = strlen(s);
                s_name = new car [size+1];
                strcpy(s_name,s);
        }
        void display( )
        {       cout<<s_name<<endl;
        }
        void replace(state & a, state &b)
        {       size = a.size +b.size;
                delete s_name;
                s_name = new car [size+1];
                strcpy(s_name, a.s_name);
                strcat(s_name, b.s_name);
        }};
void main()
{       char * temp = "Delhi";
        state state1(temp), state2("Mumbai"), state3("Nagpur"),s1,s2;
        s1.replace(state1, state2);
        s1.replace(s1, state3);
        s1.display( );
        s2.display( );
}};
```

Q 9.Point out the errors in the following C++ statements:

a) const int *pc ; *pc =10; (*pc)++;

b) float num = 3.10, const *pc = # float num1= 5.8; pc =&num1;

c) float x = 12.5; float xptr =&x;

d) float *rptr; int *iptr; iptr = fptr;

Handout VII: File Handling in C++

File

- A file is a stream of bytes stored on some secondary storage devices.
- **Text file:** A text file stores information in readable and printable form. Each line of text is terminated with an **EOL** (End of Line) character.
- **Binary file:** A binary file contains information in the non-readable form i.e. in the same format in which it is held in memory.

File Stream

Stream: A stream is a general term used to name flow of data. Different streams are used to represent different kinds of data flow.

There are three file I/O classes used for file read / write operations.

- **Ifstream:** can be used for read operations. (from file to program)
- **Ofstream:** can be used for write operations. (from program to file)
- **Fstream:** can be used for both read & write operations. (both ways)

fstream.h:

- This header file includes the definitions for the stream classes ifstream, ofstream and fstream. In C++ **file input output** facilities implemented through fstream.h header file.
- It contain predefines set of operation for handling file related input and output, fstream class ties a file to the program for input and output operation.

Opening of a file:

A file can be opened using:

- **Constructor method**. This will use default streams for file input or output. This method is preferred when file is opened in input or output mode only. Example : **ofstream file("student.dat"); or ifstream file("student.dat");**
- **Open() member function** of the stream class. It is preferred when file is opened in various modes i.e ios::in, ios::out, ios::app, ios::ate etc.
  e.g **fstream file;**
  file.open("book.dat", ios::in | ios::out | ios::binary);

File modes: The various modes in which a file can be opened are:

ios::out	It opens file in output mode (i.e write mode) and place the file pointer in beginning, if the file already exists it will overwrite the file.
ios::in	It opens file in input mode(read mode) and permit reading from the file.
ios::app	It opens the file in write mode, and places file pointer at the end of file i.e. to add new contents and retains previous contents. If file does not exist it will create a new file.
ios::ate	It opens the file in write or read mode, and places file pointer at the end of file i.e. input/ output operations can performed anywhere in the file.
ios::trunc	It truncates the existing file (empties the file).
ios::nocreate	If file does not exist this file mode ensures that no file is created and open() fails.
ios::noreplace	If file does not exist, a new file gets created but if the file already exists, the open() fails.
ios::binary	Opens a file in binary mode.

Some other important functions in file handling:

- **eof():** This function determines the end-of-file by returning true(non-zero) for end of file otherwise returning false(zero).
- **close():** This function terminates the connection between the file and stream associated with it. Syntax: Stream_object.close(); **e.g file.close();**

Text File functions:

Char I/O :

- **get() –** read a single character from text file and store in a buffer. e.g **file.get(ch);**
- **put()** - writing a single character in textfile e.g. **file.put(ch);**
- **getline() -** read a line of text from text file store in a buffer. e.g **file.getline(s,80);**
- We can also use **file>>ch** for reading and **file<<ch** for writing in text. The cascade operator does not accept white spaces, so this method can be used for reading only one word of text at a time.

Binary file functions:

- **read()-** read a block of binary data or reads a fixed number of bytes from the specified stream and store in a buffer.
  Syntax : Stream_object.read((char *)& Object, sizeof(Object));
  e.g file.read((char *)&s, sizeof(s));
- **write() –** write a block of binary data or writes fixed number of bytes from a specific memory location to the specified stream.
  Syntax : Stream_object.write((char *)& Object, sizeof(Object));
  e.g file.write((char *)&s, sizeof(s));

Note:

- Both functions take two arguments.
- The first is the address of variable, and the second is the length of that variable in bytes. The address of variable **must be type cast to type char*(pointer to character type)**
- The data written to a file using write( ) can only be read accurately using read( ).

File Pointer: The file pointer indicates the position in the file at which the next input/output is to occur.

Moving the file pointer in a file:

seekg()	It places the file pointer to the specified position in input mode of file. **e.g file.seekg(p,ios::beg); or file.seekg(-p,ios::end); or file.seekg(p,ios::cur)** i.e. to move to **p** byte position from beginning, end or current position.
seekp()	It places the file pointer to the specified position in output mode of file. **e.g file.seekp(p,ios::beg); or file.seekp(-p,ios::end); or file.seekp(p,ios::cur)** i.e. to move to **p** byte position from beginning, end or current position.
tellg()	This function returns the current working position of the file pointer in the input mode. **e.g int p=file.tellg();**
tellp()	This function returns the current working position of the file pointer in the output mode. **e.g. int p=file.tellp();**

Steps To work with a File:

- Declare an object of the desired file stream class(ifstream, ofstream, or fstream)
- Open the required file to be processed using constructor or open function.
- Process the file.
- Close the file stream using the object of file stream.

Sample programs for creating a Text File

To create a text file using strings I/O

```
#include<fstream.h>    //include header file for file operations
void main()
{char s[80], ch;
ofstream file("myfile.txt"); //open myfile.txt in default output mode
do{     cout<<"\n enter line of text";
        gets(s); //standard input
        file<<s;  // write in a file myfile.txt
        cout<<"\n more input y/n";
        cin>>ch;
}while(ch!='n'||ch!='N');
file.close();
} //end of main
```

To create a text file using characters I/O

```
#include<fstream.h>    //header file for file operations
void main()
{       char  ch;
ofstream file("myfile.txt"); //open myfile.txt in default output mode
do{     ch=getche();
if (ch==13)  //check if character is enter key
        cout<<'\n';
else
        file<<ch;   // write a character in text file 'myfile.txt '
} while(ch!=27); // check for escape key
file.close();     }  //end of main
```

Sample programs for reading and processing a Text File

Count the number of lowercase alphabets in a text file "BOOK.txt".

```
int countalpha()
{ifstream   Fin("BOOK.txt");   char
        ch;
        int       count=0;
        while(!Fin.eof())
                {Fin.get(ch);
                if (islower(ch))
                        count++;
                }
        Fin.close();
        return count;
}
```

Count the number of words having first character in uppercase.

```
int countword()
{ifstream Fin("BOOK.txt"); char ch[25];
int count=0; while(!Fin.eof())
                {Fin>>ch;
```

```
            if (isupper(ch[0])) count++;
            }
        Fin.close();
        return count;
}
```

Count the number of lines from a text files (a line can have maximum 70 characters or ends at '.')

```
int countword()
{ifstream    Fin("BOOK.txt");    char
        ch[70];
        int   count=0;
        if (!Fin)
        {cout<<"Error opening file!" ;
            exit(0);
        }
        while(1)
        {Fin.getline(ch,70,'.');
            if (Fin.eof())
                break;
            count++;
        }
        Fin.close();
        return count;
        }
```

Functions to view, search, add, modify and delete records from a binary file using classes and objects

```
#include<iostream.h>
#include<fstream.h>
#include<stdio.h>
class student
{       int rno;
        char name[30];
        int age;
public:
        void input()            //member function to input data members
        {       cout<<"\n enter roll no";
                cin>>rno;
                cout<<"\n enter name ";
                gets(name);
                cout<<"\n enter age";
                cin>>age;}
        void output()           //member function to display data members
        {       cout<< "\n roll no:"<<rno;
                cout<< "\n name :"<<name;
                cout<< "\n age:"<<age;}
        int getrno() { return rno;}             //accessor member function to return value of rno
```

```
};
void create_file()                          //create a binary file and write records in it
{       ofstream fout;
        char ch;
        fout.open("student", ios::out | ios:: binary);
        clrscr();
        student s;
        if(!fout)
        {       cout<<"File can't be opened";
                break;}
        do
        {       s.input();
                cout<<"\n more record y/n";
                cin>>ch;
                }while(ch!='n' || ch!='N');
        fout.close();
}
void read_file()                            //display all records from the binary file
{       ifstream fin;
        student s;
        fin.open("student.dat",ios::in | ios:: binary);
        fin.read((char *) &s,sizeof(student));
        while(file)
        {       s.output();
                cout<< "\n";
                fin.read((char *) & s,sizeof(student));          }
        fin.close();
}
void modify_record()                                //input a rno and modify record
{       student s;
        fstream file;
        file.open("student.dat",ios::in|ios::out|ios::ate|ios::binary);
        int r,pos=-1;
        cout<<"\n enter the rollo no of student whom data to be modified";
         cin>>r;
        file.read((char *)&s,sizeof(s));
        while(file)
        {       if (r==s.getrno())              //if a matching record is found
                {       cout<<"\n record is ";
                        s.output();
                        pos =file.tellg()-size(s);
                        break;}
                file.read((char *)&s,sizeof(s));          }
        if(pos>-1)
        {       cout<< "\n enter new   record";
                s.input();
                file.seekp(pos,ios::beg);               //reposition the file pointer on the same record
                file.write((char *)&s,sizeof(s));
```

```
                cout<< "\n record modified successfully";       }

        else
                cout<< "\n record does not exist";
        }
void delete_record()
{       fstream file("student.dat", ios::in|ios::binary);
        fstream newfile("newstu.dat",ios::out|ios::binary);
        student s;
        cout<<"\n enter the rollno no of student whom record to be deleted"; cin>>r;
        file.read((char *)&s,sizeof(s));
        while(file)
        {       if (r!=s.getrno())
                {       newfile.write((char *)&s,sizeof(s));     }
                //write all records on the newfile except the one to be deleted
                file.read((char *)&s,sizeof(s));
        }
file.close();
newfile.close();
remove("student.dat");                          //delete old student.dat
rename("newstu.dat","student.dat");             //rename newstu as student.dat
}
void search_record()
{       student s;
        fstream file;
        file.open("student.dat",ios::in|os::binary);
        int r,flag=-1;
        cout<<"\n enter the rollo no of student whom record to be searched";
        cin>>r;
        file.read((char *)&s,sizeof(s));
        while(file)
        {       if (r==s.getrno())
                {       cout<<"\n record is ";
                        s.output();
                        flag=1;
                        break;          }
        file.read((char *)&s,sizeof(s));
        }
        if(flag==1)
        cout<< "\n search successfull";
        else
        cout<< "\n search unsuccessfull";
        file.close();}
```

Assignment No. 6
File Handling

Q1. Which is the base class of all the classes?
Q2. Which are the two ways of opening a file?
Q3. Why is it not essential to include iostream.h if fstream.h is included in the program?
Q4. .Differentiate between:

a) ios:: ate and ios app
b) Binary Files and Text files
c) cin.getline( ) and cin.get( )
d) ios:: nocreate and ios:: noreplace
e) tellg( ) and tellp( )
f) seekg( ) and seekp( )

Q5. Assuming the class STOCK, write user defined functions in C++ to perform the following :

a) Write the objects of STOCK to a binary file
b) Read the objects of STOCK from binary file and display all the records with price less than 200 on the screen.

```
class STOCK
{       int item_no;
        char item[10];
        float price;
public :
        void entry( )
            {cin>>item_no;
            gets(item);
            cin>>price;}
        void show( )
            {cout<< item_no << item << price;}
};
```

Q6. Assuming the class employee given below, complete the definitions of the member functions.

```
class employee
 {      int emp_no;
        char name[10];
        float salary;
public :
        void entry( );                  // Input data from the user.
        void show( );                   // Display data on the screen.
        void write_file( )              // Write data to the file from an object.
        void read_file( )       };      // Read data from the file to an object.
```

Q7. Given the binary file sports.dat, containing records of the following structure type: struct Sports

```
{       int code;
        char Event[20];
        char participant[10][30];
         int no_of_participants;
};
```

a) Write a function in C++ that would read contents from the file sports.dat

and create a file named athletic.dat copying only those records from sports.dat where the event name is "Athletics".

b) Write a function in C++ to update the file with the new value of no_of_participants. The value of code and no_of_participants are read during the execution of the program.

Q8. Write a function in C++ to print the count of the world **the** as an independent word in the text file story.txt.

For Example , if the content of the file story.txt is:

There was a monkey in <u>the</u> zoo. <u>The</u> monkey was very naughty.

Then the output of the program should be 2.

Q9. Assume a text file "Test.txt" is already created. Using this file, write a function to create three files "LOWER.TXT" which contains all the lowercase vowels and "UPPER.TXT" which contains all the uppercase vowels and "DIGIT.TXT" which contains all digits.

Q10. Consider the code given below :

```
void main( )
{   char ch = 'A';
    fstream fout ("data.dat", ios::out);
    fout<<ch;
    int p = fout.tellg( );
    cout<<p;
}
```

What is the output if the file content before the execution of the program is the string "ABC".

Q11. Observe the program segment given below carefully and fill the blanks marked as statement 1 and statement 2 using seekg( ) and seekp( )functions for performing the required task.

```
# include <fstream.h>
class Item
{       int Ino;
        char item[20];
public :
// Function to search and display the content from a particular record
        number void search (int);
// Function to modify the content of a particular record number
        void Modify (int);
};
void item :: Search( int RecNo)
{
        fstream file;
        file.open ("stock.dat", ios::binary|ios::in);
        _____ //Statement 1
        file.read((char*)this,   sizeof(item));
        cout<<Iino<< " "<<Item;
        file.close ( );
}
void item :: Modify( int RecNo)
{
        fstream file;
```

```
                file.open                    ("stock.dat",
                ios::binary|ios::in|ios::out); cout>>Ino;
                cin.getline(item,20);
                _____ //Statement 2
                file.write((char*)this,  sizeof(item));
                cout<<ino<< " "<<Item;
                file.close ( );
        }
};
```

Q12. Following is the structure of each record in the data file named "Product.dat". struct Product

```
{       char P_code[10];
        char P_Description [10];
        int stock;
};
```

Write a function in C++ to update the file with a new value of Stock. The stock and the P_code, whose stock is to be updated are read during the execution of the program.

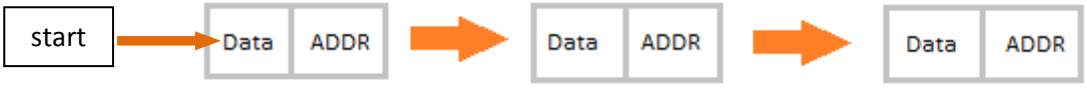
Q13. Following is the structure of each record in the data file named "Flights.dat". struct Flight

```
{       int Fl_No;
        char Starting [20];
        char Ending [20];
};
```

Write a function in C++ to delete a record from the file. The Fl_No, whose record is to be deleted is read during the execution of the program.

Handout VIII: Linked Lists, Stacks and Queues

Linked List is a linear data structure and it is very common data structure which consists of group of nodes in a sequence which is divided in two parts. Each node consists of its own data and the address of the next node and forms a chain.



Advantages of Linked Lists over arrays:

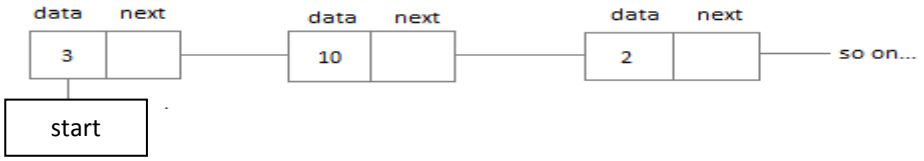
- They are a dynamic in nature which allocates the memory when required unlike arrays where array size must be known at the time of declaration and cannot be changed dynamically.
- Insertion and deletion operations can be easily implemented unlike arrays where it is limited by the size of the array and also requires shifting of elements.
- Linked List reduces the access time.

Disadvantages of Linked Lists:

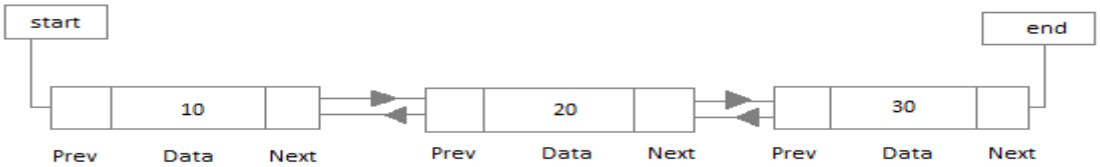
- The memory is wasted as pointers require extra memory for storage.
- Element cannot be accessed randomly; each node has to be accessed sequentially.
- Reverse Traversing is difficult in a singly linked list.

Types of Linked Lists:

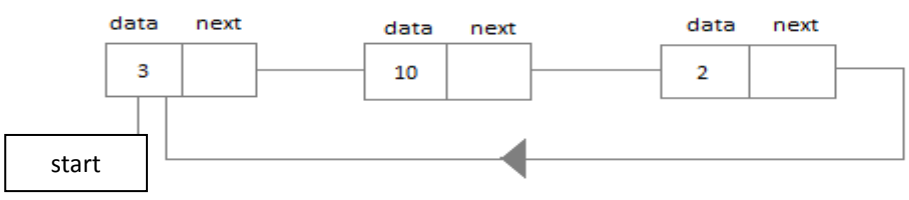
- **Singly Linked List :** Singly linked lists contain nodes which have a data part as well as an address part i.e. next, which points to the next node in sequence of nodes. The operations we can perform on singly linked lists are insertion, deletion and traversal.



- **Doubly Linked List:** In a doubly linked list, each node contains two links the first link points to the previous node and the next link points to the next node in the sequence.

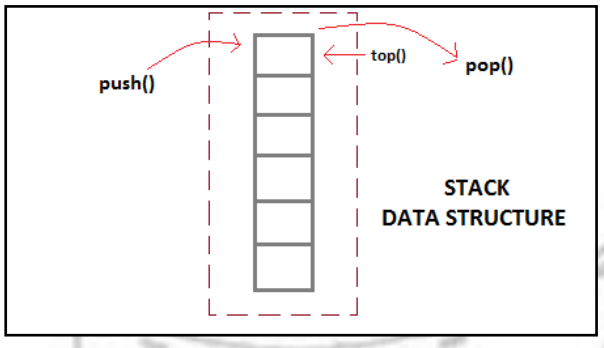


- **Circular Linked List :** In the circular linked list the last node of the list contains the address of the first node and forms a circular chain.



Stacks:

Stack is an abstract data type with a predefined capacity. It is a linear data structure that allows adding and removing elements in a particular order. Every time an element is added, it goes on the top of the stack, the only element that can be removed is the element that was at the top of the stack, just like a pile of objects.



Basic features of Stack

1. Stack is an ordered list of similar data type.
2. Stack is a **LIFO** structure. (Last in First out).
3. **push()** function is used to insert new elements into the Stack and **pop()** is used to delete an element from the stack. Both insertion and deletion are allowed at only one end of Stack called **Top**.
4. Stack is said to be in **Overflow** state when it is completely full and is said to be in **Underflow** state if it is completely empty.

Applications of Stack

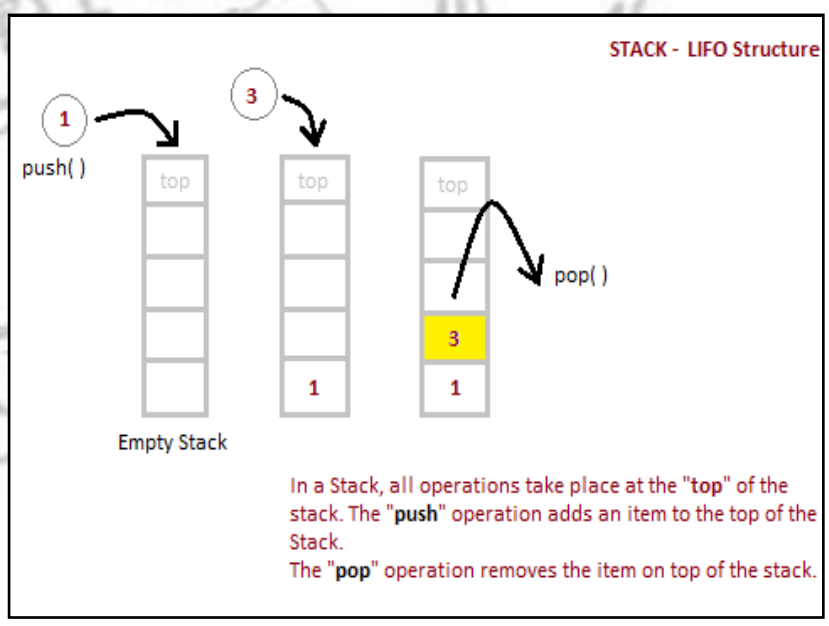
The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack.

There are other uses also like : **Parsing**, **Expression Conversion**(Infix to Postfix, Postfix to Prefix etc), function calls and many more.

Implementation of Stack

Stack can be easily implemented using an Array or a Linked List. Arrays are quick, but are limited in size and Linked List requires overhead to allocate, link, unlink, and deallocate, but is not limited in size. **Program for implementation of a Stack using an array:**

```
#include<iostream.h>
#include<conio.h>
#define size 4
class stack
{
   int data[size];
   int top;
public:
   stack()
   {      top=-1;   }
   void push();
   void pop();
   void display();
};
void stack::push()
{   if(top==size-1)
   {  cout<<"\nStack is full";
      return;   }
   else
   {  top++;
      cout<<"Enter Data : ";
      cin>>data[top];   }
}
```



```
void stack::pop()
{    if(top==-1)
        cout<<"\n Stack is empty";
    else
    {   cout<<data[top]<<"deleted "<<endl;
        top--;    }
}
void stack::display()
{   int t=top;
    while(t>=0)
    {  cout<<data[t]<<endl;
        t--;    }
}
void main()
{   stack st;
    int ch;
    do
    {  cout<<"\n1. Push\n2. Pop\n3. Display \n4.Quit\nEnter Choice(1-4) ";
        cin>>ch;
        switch(ch)
        {   case 1: st.push();break;
            case 2: st.pop();break;
            case 3: st.display();
        }
    }while(ch!=4);
}
```

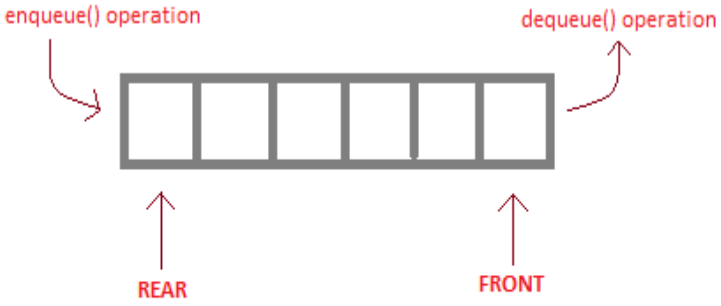
Program for Stack Implementation using Linked List:

```
#include<iostream.h>
#include<conio.h>
struct node
{           int data;
           node *next;
};
class stack
{           node *top;
public :
    stack()
          {          top=NULL;      }
     void push();
     void pop();
     void display();
    ~stack();
};
void stack::push()
{     node *temp;
     temp=new node;
     cout<<"Enter data :";
     cin>>temp->data;
     temp->next=top;
     top=temp;
}
```

```
void stack::pop()
{    if(top!=NULL)
    {    node *temp=top;
         top=top->next;
         cout<<temp->data<<"deleted";
         delete temp;     }
    else
         cout<<"Stack empty";
}
void stack::display()
{    node *temp=top;
    while(temp!=NULL)
    { cout<<temp->data<<" ";
     temp=temp->next;     }
}
stack::~stack()
{    while(top!=NULL)
    {    node *temp=top;
         top=top->next;
         delete temp;     }
}
void main()
{    stack st;
    char ch;
    do
    {    cout<<"Stack Options\nP for push \nO for Pop \nD for Display \nQ for quit";
         cin>>ch;
         switch(ch)
         {    case 'P': st.push();break;
              case 'O': st.pop();break;
              case 'D': st.display();break;
         }
    }while(ch!='Q');
}
```

Queues:

Queue is also an abstract data type or a linear data structure, in which the first element is inserted from one end called **REAR**(also called tail), and the deletion of existing element takes place from the other end called as **FRONT**(also called head). This makes queue as FIFO data structure, which means that element inserted first will also be removed first.



enqueue( ) is the operation for adding an element into Queue.

dequeue( ) is the operation for removing an element from Queue .

The process to add an element into queue is called **Enqueue** and the process of removal of an element from queue is called **Dequeue**.

Basic features of Queue

1. Like Stack, Queue is also an ordered list of elements of similar data types.

2. Queue is a **FIFO**(First in First Out) structure.

3. Once a new element is inserted into the Queue, all the elements inserted before the new element in the queue must be removed, to remove the new element.

4. **peek( )** function is oftenly used to return the value of first element without dequeuing it.

Applications of Queue

Queue, as the name suggests is used whenever we need to have any group of objects in an order in which the first one coming in, also gets out first while the others wait for there turn, like in the following scenarios :

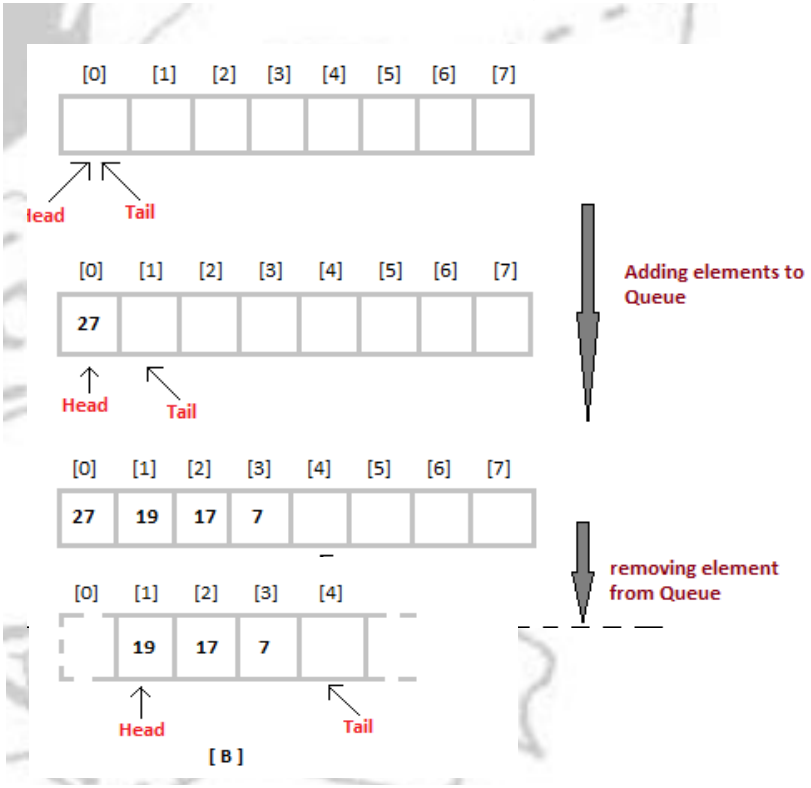
1. Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
2. In real life, Call Center phone systems will use Queues, to hold people calling them in an order, until a service representative is free.
3. Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive, First come first served.

Implementation of Queue

Queue can be implemented using an Array or Linked List. The easiest way of implementing a queue is by using an Array. Initially the **head**(FRONT) and the **tail**(REAR) of the queue points at the first index of the array (starting the index of array from 0). As we add elements to the queue, the tail keeps on moving ahead, always pointing to the position where the next element will be inserted, while the head remains at the first index.

When we remove element from Queue, we remove the element from **head** position and then move **head** to the next position.

After removal of first element, the size on Queue is reduced by one space each time.



Program for implementation of a Queue using an array:

```
#define SIZE 100
class queue{
        int a[100];
        int rear;    //same as tail
        int front;   //same as head
  public:
queue()  {rear = front = -1;     }
 void enqueue(int x);
 int dequeue();
 void display();
}
void queue :: enqueue(int x)
{            if( rear = SIZE-1)
            {            cout << "Queue is full";        }
             else
             {             a[++rear] = x;       }
}
int queue :: dequeue()
{ return a[++front];    }
void queue :: display()
{            int i;
             for( i = front; i <= rear; i++)
```

```
            {           cout << a[i];       }
}
void main()
{   queue obj; char ch;
    do
    {    cout<< "i. insert\nd. Delete\ns. Display\n q. quit ";
         cin>>ch;
         switch(ch)       {
             case 'i' : obj.enqueue();break;
             case 'd' : obj.dequeue();break;
             case 's' : obj.display();       }
    }while(ch!='q');
}
```

Program for Queue Implementation using Linked List:

```
#include<iostream.h>
#include<conio.h>
struct node
{   int data;
    node *next;
};
class queue
{    node *rear,*front;
public:
    queue()     { rear=NULL;front=NULL;}
    void qinsert();
    void qdelete();
    void qdisplay();
    ~queue();
};
void queue::qinsert()
{   node *temp=new node;
    cout<<"Data :";     cin>>temp->data;
    temp->next=NULL;
    if(rear==NULL)
    {    rear=temp;
         front=temp;     }
    else
    {   rear->next=temp;
         rear=temp;     }
}
void queue::qdelete()
{  if (front!=NULL)
    {   node *temp=front;
         cout<<front->data<<"deleted \n";
         front=front->next;
         delete temp;
         if(front==NULL)
              rear=NULL;
    }
    else
```

```
        cout<<"Queue Empty..";
}
void queue::qdisplay()
{   node *temp=front;
    while(temp!=NULL)
    {   cout<<temp->data<<endl;
        temp=temp->next;
    }
}
queue::~queue()
{    while(front!=NULL)
    {   node *temp=front;
        front=front->next;
        delete temp;
    }
}
void main()
{   queue obj; char ch;
    do     {
        cout<< "i. insert\nd. Delete\ns. Display\n q. quit ";
        cin>>ch;
        switch(ch)          {
            case 'i' : obj.qinsert();break;
            case 'd' : obj.qdelete();break;
            case 's' : obj.qdisplay();          }
    }while(ch!='q');
}
```

Circular Queue:

In a linear queue, when we delete any element, the front shifts right by one place but the position vacated by the dequeued element is not used later. So, when we perform more add and delete operations, **memory at the beginning of the array lies unused.** But in a circular queue, as we add more elements and reach the end of the array we circle back and check the front end. If there are places available because of previously deleted elements we start using them for newer elements being added in the queue.

```
#include<iostream.h>
#include<conio.h>
#define size 4
class cqueue
{ int data[size];
 int front,rear;
public:
 cqueue()  { front=-1;rear=-1; }
 void insert();
 void remove();
 void display();
};
void cqueue::display()
{int i=0;
cout<<"the list now is:\n";
if(front<=rear)
{     for(i=0;i<front;i++)cout<<"-";
```

```
        cout<<">>>";
        for(i=front;i<rear;i++)cout<<data[i]<<"-";
        cout<<data[rear]<<"<<<"<<endl;
        }
    else
     {    for(i=0; i<rear;i++) cout<<data[i]<<"-";
        cout<<data[rear]<<"<<<";
        for(;i<front;i++)cout<<"-";
        cout<<">>>";
        for(i=front;i<size;i++)cout<<data[i]<<"-";
        cout<<"wrap around"<<endl;
        }
     }
    void cqueue::insert()
    { if(rear==size-1&&front==0 || front==rear+1)
     { cout<<"\nCircular queue is full";
     return; }
     else if(rear==-1)
     { rear++;
     front++; }
     else if(rear==size-1)
     rear=0;
     else
     rear++;
     cout<<"Enter Data : ";
     cin>>data[rear];
    display();
    }
    void cqueue::remove()
    { if(front==-1)
     { cout<<"\n Circular Queue is empty";return; }
     cout<<data[front]<<" deleted"<<endl;
     if(front==rear)
     { front=-1;rear=-1; }
     else if(front==size-1)
     front=0;
     else
     front++;
    display();
    }
    void main()
    { cqueue cq;
     int ch;
     do
     { cout<<"\n1. Insert\n2. Remove\n3. Quit\nEnter Choice(1-3) ";    cin>>ch;
     switch(ch)
     { case 1: cq.insert();break;
       case 2: cq.remove();break; }
     }while(ch!=3);
    }
```

Assignment No: 7
Linked list, Stacks and Queues

Q1. Define Linked List? What is its advantage over arrays.

Q2. Differentiate between circular queues and d-queues.

Q3. Each node of a **STACK** contains the following information, in addition to the required pointer field :

a. acc_no
b. acc_name

Give the structure of node for the linked stack in question. TOP is a pointer which points to the topmost node of the **STACK**. Write the following functions.

i) **PUSH( )** – *To push a node to the **STACK** which is allocated dynamically.*
ii) **POP ( )** – *To remove a node from the **STACK** and release the memory.*

Q4. Change the following infix expression to its postfix equivalent: A +B * (- C) / A ^D
Show stack after each pass.

Q5. Solve the following **postfix** expression using the stack method, for the given values.
A B + C * D E / + where A=3, B=5, C=2, D=8, E=4. Show stack after each pass.

Q6. Solve the following postfix expression using a stack. TRUE FALSE and FALSE TRUE not or and
Show the contents of stack after execution of each operation.

Q7. Given the following class :

```
Char *msg[ ] = { "overflow", " underflow" };
class stack
{       int top;    //the stack pointer
        int stk [5]; // the elements
        void err_rep( int e_enum) { cout<< msg[e_enum]; }
public :
        stack ( ) { top=-1;} //function to initialize the stack pointer
        void push(int); // put new value in stk
        void pop ( ); // get the top value and display it
};
```

Define the member functions push( ) and pop( ) outside the class, the functions should invoke err_rep() in case of overflow/ underflow.

Q8. Given the following class :

```
char *msg[ ] = { "overflow", " underflow" };
class queue
{       int front, rear; //the queue pointers
        int que [15];  // the elements
        void err_rep( int e_enum)
        { cout<< msg[e_enum]; }
 public :
        queue ( ) { front=-1;rear=-1} //function to initialize the queue pointers
        void inst(int);   // add a new value in queue
        void remove ( );        // remove an element from the queue & display it
};
```

Define the member functions inst( ) and remove( ) outside the class, the functions should invoke err_rep( ) in case of overflow/ underflow.

Q9. Write functions to perform insert and delete operations on a circular queue of integers. (Array Implementation)

Handout IX: Boolean Algebra

Binary Decision: A decision which results into either YES(True) or NO(FALSE).
Boolean algebra is an algebra that deals with Boolean values((TRUE and FALSE) .

Truth table:
A truth table is composed of one column for each input variable (for example, A and B), and one final column for all of the possible results of the logical operation that the table is meant to represent (for example, A XOR B). Each row of the truth table therefore contains one possible configuration of the input variables (for instance, A = true B = false), and the result of the operation for those values.

Logical Operators: logic statements or truth functions are combined with the help of logical operators like AND, OR and NOT.
NOT Operator—Operates on single variable. It gives the complement value of the input variable.

X	$\overline{X}$
0	1
1	0

OR Operator -It is a binary operator and denotes logical Addition operation and is represented by "+" symbol

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 1

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

AND Operator – AND Operator performs logical multiplications and symbol is (.) dot.

0.0=0
0.1=0
1.0=0
1.1=1

Truth table:

X	Y	X.Y
0	0	0
0	1	0
1	0	0
1	1	1

Basic postulates of Boolean Algebra:
Boolean algebra consists of fundamental laws that are based on theorems of Boolean algebra. These fundamental laws are known as basic postulates of Boolean algebra. These postulates state basic relations in boolean algebra, that follow:
I. If x!= 0 then x=1 and If x!=1 then x=0

II. OR relations(logical addition)

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 1

III AND relations (logical multiplication)

0.0 = 0
0.1 = 0
1.0 = 0
1.1 = 1

IV Complement Rules $\bar{0} = 1, \bar{1} = 0$

Principal of Duality
This principal states that we can derive a Boolean relation from another Boolean relation by performing simple steps. The steps are:-
1. Change each AND(.) with an OR(+) sign
2. Change each OR(+) with an AND(.) sign

3. Replace each 0 with 1 and each 1 with 0
e.g
0+0=0 then dual is 1.1=1
1+0=1 then dual is 0.1=0

1. **Properties of 0 and 1:**0 + X = X, 1 + X = 1, 0.X = 0, 1.X = X
2. **Idempotence Law:**(a) X +X = X (b) X.X = X
3. **Involution Law:** $\overline{\overline{A}} = A$
4. **Complementarity Law:** $(a)X + \overline{X} = 1 (b)X.\overline{X} = 0$
5. **Commutative Law:(a)** X+Y = Y+X **(b)** X.Y = Y.X
6. **Associative Law: (a)** X + (Y+Z) = (X+Y) + Z **(b)** X.(Y.Z) = (X.Y).Z
7. **Distributive Law: (a)** X(Y+Z) = XY + XZ
(b) X+YZ = (X+Y) (X+Z)
Algebraic proof: Taking and simplifying RHS
= (X+Y)(X+Z)
=XX+XZ+XY+YZ X.X=X By Idempotence Law
=X+XZ+XY+YZ
=X(1+Z)+XY+XZ 1+Z=1 By Property Of 1
=X+XY+YZ
=X(1+Y)+YZ 1=Y=1 By Property Of 1
=X+YZ=LHS
8. **Absorption Law: (a)** X + XY = X
(b) X (X+Y)= X

Algebraic proof: Taking and simplifying LHS	
=X.X+XY	By Property Of 1
=X+XY	By Idempotence LAW
=X(1+Y)	
=X.1	By Property Of 1
=X=RHS	

9. **Third distributive Law:** X+X'Y=X+Y

Algebraic proof: Taking and simplifying LHS	
=X.1+X'Y	By Property Of 1
=X(1+Y)+X'Y	By Property Of 1
=X+XY+X'Y	
=X+(X+X')Y	X+X'=1
=X+Y=RHS	

10. **Demorgan's Theorems** (1) $\overline{X + Y} = \overline{X}.\overline{Y}$
(2) $\overline{X.Y} = \overline{X} + \overline{Y}$

Derivation of Boolean expression:-

Minterm :

A minterm is a Product of all the literals within the logic System.

Step involved in minterm expansion of Expression

1. First convert the given expression in sum of product form.
2. In each term is any variable is missing(e.g. in the following example Y is missing in first term and X is missing in second term), multiply that term with (missing term+complement(missing term))factor e.g. if Y is missing multiply with (Y+Y')
3. Expand the expression.
4. Remove all duplicate terms and we will have minterm form of an expression.

Example: Convert X+Y into minterms

=X+Y=X.1+Y.1

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

=XY+XY'+XY+X'Y

=XY+XY'+X'Y

Maxterm:

A maxterm is a sum of all the literals (with or without the bar) within the logic system. Boolean Expression composed entirely either of Minterms or Maxterms is referred to as **Canonical Expression**.

Canonical Form:

Canonical expression can be represented is derived from

(i) Sum-of-Products(SOP) form

(ii) Product-of-sums(POS) form

Sum of Product (SOP)

1. Various possible input values
2. The desired output values for each of the input combinations

X	Y	R
0	0	X'Y'
0	1	X'Y
1	0	XY'
1	1	XY

Product of Sum (POS)

When a Boolean expression is represented purely as product of Maxterms, it is said to be in Canonical Product-of-Sum form of expression.

X	Y	Z	R
0	0	0	X+Y+Z
0	0	1	X+Y+Z'
0	1	0	X+Y'+Z
0	1	1	X+Y'+Z'
1	0	0	X'+Y+Z
1	0	1	X'+Y+Z'
1	1	0	X'+Y'+Z
1	1	1	X'+Y'+Z'

Minimization of Boolean expressions:-

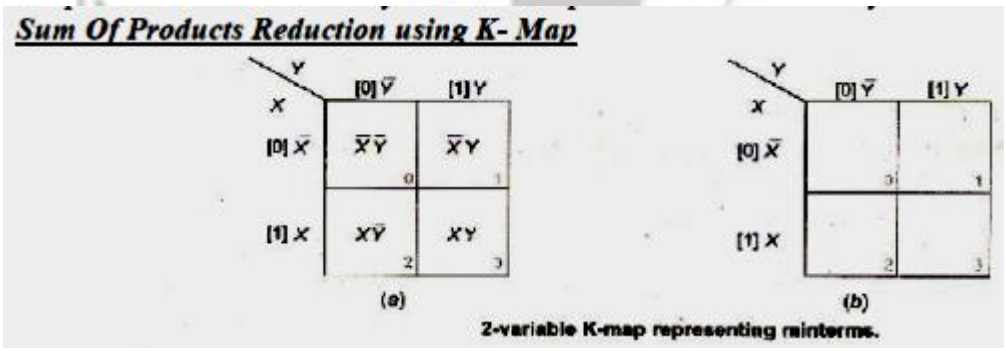
After obtaining SOP and POS expressions, the next step is to simplify the Boolean expression.

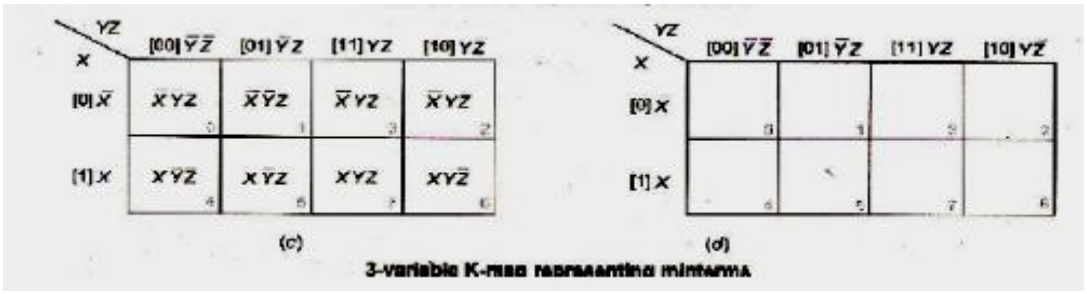
There are two methods of simplification of Boolean expressions.
1. Algebraic Method
2. Karnaugh Map

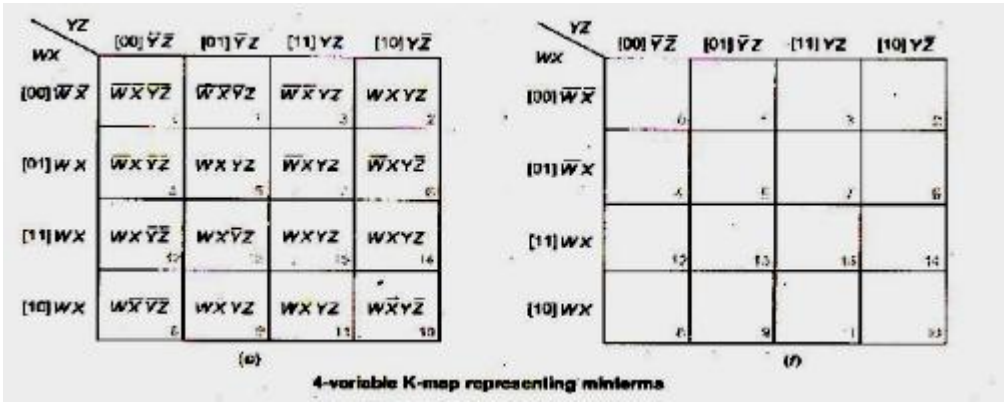
1. **Algebraic method:** This method makes use of Boolean postulates, rules and theorems to simplify the expression.

2. **Karnaugh Map**: Karnaugh map or K Map is a graphical display of the fundamental products in a truth table.
For example:

- Put a 1 in the box for any minterm that appears in the SOP expansion.
- Group together the largest adjacent blocks in octets, quads, pairs or singles.
    - Pair: Two adjacent 1's makes a pair and eliminates one variable
    - Quad: Four adjacent 1's makes a quad and eliminates two variables.
    - Octet: Eight adjacent 1's makes an Octet and eliminates three variables.
- Blocks can "wrap around" the edges.
- group together adjacent cells of 1s, to form largest possible rectangles of sizes that are powers of 2. Notice that you can overlap the blocks if necessary.







Formation of expression: Form one minterm for each group that has been created by keeping only the literals which are constant in all cells of the group and omitting the literals which change their values.

Add the minterms to form an SOP expression.

Example1: Reduce the following Boolean expression using K-Map:
F(P,Q,R,S)=Σ(0,3,5,6,7,11,12,15)

Soln:
This is 1 quad, 2pairs & 2 singles
Quad(m3+m7+m15+m11) reduces to RS
Pair(m5+m7) reduces to P'QS
Pair (m7+m6) reduces to P'QR
Single m0=P'Q'R'S' and m12=PQR'S'
hence the final expressions is F=RS + P'QS + P'QR + PQR'S' + P'Q'R'S'

RS / PQ	R'S' 00	R'S 01	RS 11	RS' 10
P'Q' 00	1 0	1	1 3	2
P'Q 01	4	1 5	1 7	1 6
PQ 11	1 12	13	1 15	14
PQ' 10	8	9	1 11	10

Example2: Reduce the following Boolean expression using K-Map:
F(P,Q,R,S)=Π(0,1,3,5,6,7,10,14,15)

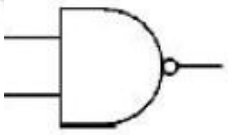
Soln:
Reduced expressions are as follows:
For pair 1, (P+Q+R)
For pair 2, (P'+R'+S)
For Quad 1, (P+S')
For Quad 2, (Q'+R')
Hence final POS expression will be

(P+Q+R) (P'+R'+S) (P+S') (Q'+R')

RS / PQ	R+S 00	R+S' 01	R'+S' 11	R'+S 10
P+Q 00	0 0	0 1	0 3	2
P+Q' 01	4	0 5	0 7	0 6
P'+Q' 11	12	13	0 15	0 14
P'+Q 10	8	9	11	0 10

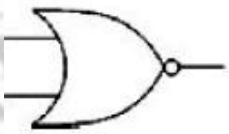
More about Logic Gates:

NAND gate (NAND = Not AND)
This is an AND gate with the output inverted, as shown by the 'o' on the output. The output is true if input A AND input B are NOT both true: Q = NOT (A AND B) A NAND gate can have two or more inputs, its output is true if NOT all inputs are true.



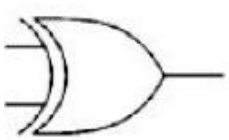
Input A	Input B	Output Q
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate (NOR = Not OR)
This is an OR gate with the output inverted, as shown by the 'o' on the output. The output Q is true if NOT inputs A OR B are true: Q = NOT (A OR B) A NOR gate can have two or more inputs, its output is true if no inputs are true.



Input A	Input B	Output Q
0	0	1
0	1	0
1	0	0
1	1	0

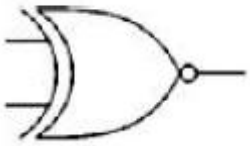
EX-OR (EXclusive-OR) gate
The output Q is true if either input A is true OR input B is true, but not when both of them are true: Q = (A AND NOT B) OR (B AND NOT A) This is like an OR gate but excluding both inputs being true. The output is true if inputs A and B are DIFFERENT. EX-OR gates can only have 2 inputs.



Input A	Input B	Output Q
0	0	0
0	1	1
1	0	1
1	1	0

Traditional symbol Truth Table

EX-NOR (EXclusive-NOR) gate
This is an EX-OR gate with the output inverted, as shown by the 'o' on the output. The output Q is true if inputs A and B are the SAME (both true or both false):



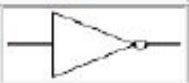
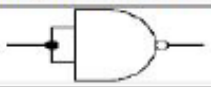
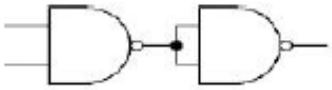
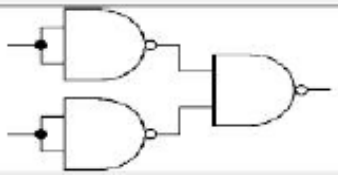
Traditional symbol

Input A	Input B	Output Q
0	0	1
0	1	0
1	0	0
1	1	1

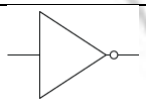
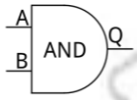
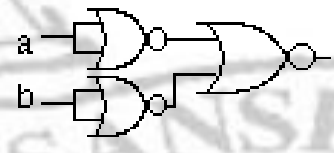
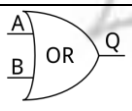
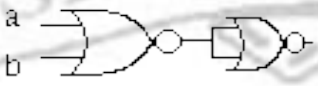
Truth Table

Q = (A AND B) OR (NOT A AND NOT B) EX-NOR gates can only have 2 inputs.

NAND gate equivalents
The NAND gate can be used to implement any other gate. The table below shows the NAND gate equivalents of NOT, AND, OR and NOR gates:

Gate	Equivalent in NAND gates
NOT 	
AND 	
OR 	

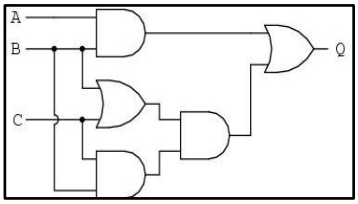
NOR gate equivalents
The NOR gate can also be used to implement other gatse. The table below shows the NOR gate equivalents of NOT, AND, OR gates:

GATE		Equivalent in NOR gates
NOT		
AND		
OR		

Assignment No : 8
Boolean Algebra

1. (a) State Distributive law and verify the same using truth table.

(b)Write the equivalent Canonical Sum of Product expression for the following Product of Sum Expression: **F(X,Y,Z)=∏(1, 3, 6, 7)**



(c)Write the equivalent Boolean Expression for the following Logic Circuit.

(d) Express **(x+y)(y+z')(x'+z)** in canonical **POS** form.

(e)Reduce the following Boolean expression using K-Map

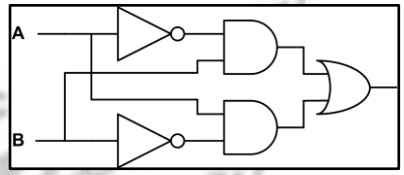
F(U,V,W,Z) = Σ(0, 1, 2, 3, 4, 10, 11)

(f) Prove algebraically:

i. (X+Y)(X+Z)=X+YZ

ii. X+Y'=X.Y+X.Y'+X'.Y'

2. (a) State and verify Associative Law.



(b) Write the equivalent expression for the following logical circuit:

(c) Express **P + Q'R** in canonical **SOP** form.

(d) Implement the following expressions with the help of NOR gates only:

i. (a'+b'+c')(a+b+c')

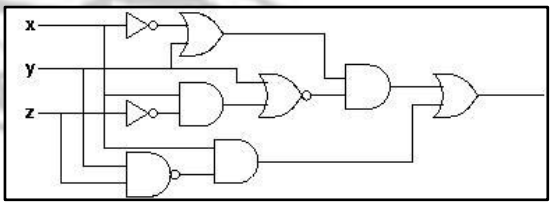
ii. (x+z)(y'+z)(x'+y+z)

(e) Reduce the following Boolean expression using K-map: **F(P,Q,R,S) = ∏ (0,3, 5, 6, 7, 11, 12, 15)**

(f) Simplify algebraically:
i. XY+YZ+YZ'
ii. X'Y'Z'+X'Y'Z+X'YZ+X'YZ'+XY'Z'+XY'Z

3. (a) State and verify DeMorgan's Law algebraically.

(b) Write the equivalent Boolean expression for the following logic circuit:



(c) Represent the Boolean expression **x.y' + y.z'** with the help of NAND gates only.

(d) Obtain simplified form of the following Boolean expression using Karnaugh Map

F(x,y,z,w) = Σ(1, 3, 4, 5, 7, 9, 11, 12, 13, 15)

(e) Why are NAND and NOR gates called universal gates?

4. (a) What is the significance of principle of duality?

(b) What are tautology and fallacy? Prove that (1+y) is a tautology and 0.y is a fallacy.

(c) Write the POS form of Boolean Function F(U,V,W) which is represented in the truth table as :

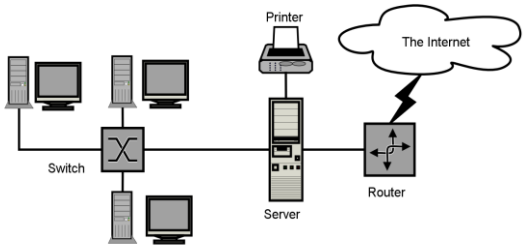
U	V	W	F
0	0	0	0
0	0	1	0
0	I	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

(d) Obtain reduced form of the following Boolean expression using K Map: **F(x,y,z,w) =∏ (2, 3, 4, 5,6, 7,8, 10,11)**

Handout X: Networking and Communication Concepts

Network

- A collection of interconnected computers is called a computer network.
- Two computers are said to be interconnected if they are capable of sharing and exchanging information.



Need

- **Resource Sharing** means to make all programs, data and peripherals available to anyone on the network irrespective of the physical location of the resources and the user.
- **Reliability** means to keep the copy of a file on two or more different machines, so if one of them is unavailable (due to some hardware crash or any other) them its other copy can be used.
- **Cost Factor** means it greatly reduces the cost since the resources can be shared
- **Communication Medium** means one can send messages and whatever the changes at one end are done can be immediately noticed at another.

Evolution of Networking

1. **ARPANET**: In 1969, The US govt. formed an agency named ARPANET (Advanced Research Projects Agency NETwork) to connect computers at various universities and defense agencies. The main objective of ARPANET was to develop a network that could continue to function efficiently even in the event of a nuclear attack.

2. **Internet (INTERconnection NETwork)**: The Internet is a worldwide network of computer networks. It is not owned by anybody.

3. **Interspace**: InterSpace is a client/server software program that allows multiple users to communicate online with real – time audio, video and text chat in dynamic 3D environments.

NETWORK TERMINOLOGY

- **Nodes/ Workstation:** the computers that are attached to the network and are seeking to share the resources of the network.
- **Server**: A computer that facilitates the sharing of data, software and hardware resources on the network.
- **Network Interface Unit**: An interpreter that helps to establish communication between the server and workstations.
- **MAC(Media Access Control) Address**: The physical address assigned by the NIC manufacturer.

SWITCHING TECHNIQUES

Switching techniques are used for transmitting data/ provide communication across networks.
Different types are:

1. **Circuit Switching**: In the Circuit Switching technique, first, the complete end-to-end physical connection between the source and the destination computers is established and then the message is transmitted through the path. The main advantage of this technique is guaranteed delivery of the message. Mostly used for voice communication.
2. **Message Switching**: In the Message switching technique, no physical path is established between sender and receiver in advance. This technique follows the store and forward mechanism.
3. **Packet Switching**: In this switching technique fixed size of packet can be transmitted across the network.

Comparison between the Various Switching Techniques: Criteria	Circuit Switching	Message Switching	Packet Switching
Path established in advance	Yes	No	No
Store and forward technique	No	Yes	Yes
Message follows multiple routes	No	Yes	Yes

DATA COMMUNICATION TERMINOLOGIES

Data channel:- The medium used to carry information / data is carried from one point to another in the network.

Baud & bits per second (bps):- It is used to measure the information carrying capacity of a communication channel.

Measurement Units:- bit

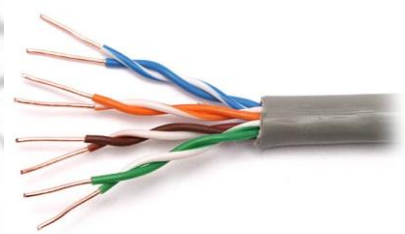
1 Byte= 8 bits
1 KBPS (Kilo Byte Per Second)= 1024 Bytes
1 Kbps (kilobits Per Second) = 1024 bits
1 Mbps (Mega bits Per Second)=1024 Kbps

Bandwidth:- The difference between the highest and lowest frequencies of a transmission channel. Is directly proportional to amount of data transmitted or received per unit time. Unit of measurement: kHz, GHz or THz.

Transmission media:

Connecting cables in a network. Can be categorized as **guided**(includes cables) and **unguided**(Include waves through air, water or vacuum).

1. Twisted pair cable: - It consists of two identical 1 mm thick copper wires insulated and twisted together. The twisted pair cables are twisted in order to reduce crosstalk and electromagnetic induction.



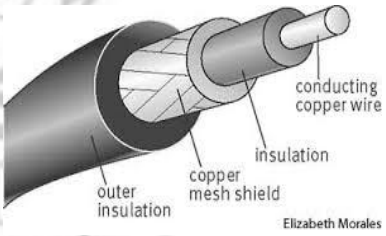
Advantages:

(i) It is easy to install, connect and maintain.
(ii) It is very inexpensive

Disadvantages:

(i) High attenuation. It is incapable to carry a signal over long distances without the use of repeaters.
(ii) Due to low bandwidth, these are unsuitable for broadband applications.

2. Co-axial Cables: It consists of a solid wire core surrounded by one or more foil or braided wire shields, each separated from the other by some kind of plastic insulator. It is mostly used in the cable wires.

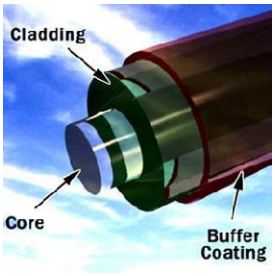


Advantages:

(i) Data transmission rate is better than twisted pair cables.
(ii) Offers higher bandwidth.
(iii) It provides a cheap means of transporting multi-channel television signals around metropolitan areas.

Disadvantages:

(i) Expensive than twisted pair cables.
(ii) Difficult to manage and reconfigure.



3. Optical fiber: - An optical fiber consists of thin glass fibers that can carry information in the form of visible light.

Advantages:

(i) Transmit data over long distance with high security.

(ii) Data transmission speed is high
(iii) Provide better noise immunity
(iv) Bandwidth is up to 10 Gbps.

Disadvantages:
(i) Expensive and fragile as compared to other guided media.
(ii) Need special care while installation.

4. Infrared: - The infrared light transmits data through the air and can propagate throughout a room, but will not penetrate walls. It is a secure medium of signal transmission.
The infrared transmission has become common in TV remotes, automotive garage doors, wireless speakers etc.

5. Radio Wave: - Radio Wave an electromagnetic wave with a frequency < 3GHz. The transmission making use of radio frequencies is termed as radio-wave transmission.

Advantages:
(i) Radio wave transmission offers mobility.
(ii) It is cheaper than laying cables and fibers.
(iii) It offers ease of communication over difficult terrain.

Disadvantages:
(i) Radio wave communication is insecure communication.
(ii) Radio wave propagation is susceptible to weather effects like rains, thunder storms etc.

6. Microwave Wave: - The Microwave transmission is a line of sight transmission. Microwave signals travel at a higher frequency than radio waves(> 3GHz) and are popularly used for transmitting data over long distances.

Advantages:
(i) It is cheaper than laying cable or fiber.
(ii) It has the ability to communicate over oceans/ difficult terrain.

Disadvantages:
(i) Microwave communication is an insecure communication.
(ii) Signals from antenna may split up and transmitted in different way to different antenna which leads to reduce to signal strength.
(iii) Microwave propagation is susceptible to weather effects like rains, thunderstorms etc.
(iv) Bandwidth allocation is extremely limited in case of microwaves.

7. Satellite link: - The satellite transmission is also a kind of line of sight transmission that is used to transmit signals throughout the world.

Advantages:
(i) Area covered is quite large.
(ii) No line of sight restrictions such as natural mountains, tall building, towers etc.
(iii) Earth station which receives the signals can be fixed position or relatively mobile.

Disadvantages:-
(i) Very expensive as compared to other transmission mediums.
(ii) Installation is extremely complex.
(iii) Signals sent to the stations can be tampered by external interference.

Network devices:

- **Modem**: A MODEM (MOdulator DEModulator) is an electronic device that enables a computer to transmit data over telephone lines. There are two types of modems, namely, internal modem and external modem.
- **RJ45 connector: -** The RJ-45(Registered Jack) connectors are the plug-in devices used in the networking and telecommunications applications. They are used primarily for connecting LANs, particularly Ethernet.

- **Ethernet Card**: - It is a hardware device that helps in connection of nodes within a network.
- **Hub:** A hub is a hardware device used to connect several computers together. Hubs can be either active or passive. Hubs usually can support 8, 12 or 24 RJ45 ports.
- **Switch:** A switch (switching hub) is a network device which is used to interconnect computers or devices on a network. It filters and forwards data packets across a network. The main difference between hub and switch is that hub replicates what it receives on one port onto all the other ports while switch keeps a record of the MAC addresses of the devices attached to it.
- **Gateway**: A gateway is a device that connects dissimilar networks.
- **Repeater:** A repeater is a network device that amplifies and restores signals for long distance transmission.
- **Bridge:** A network device that establishes an intelligent connection between two local networks with the same standard.
- **Router:** used to separate different segments in a network to improve performance and reliability. Can handle different protocols.

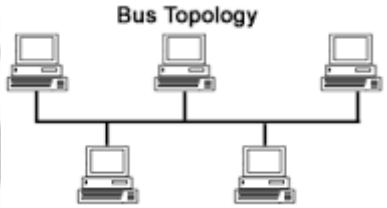
Types of Networks:
LAN (Local Area Network): A Local Area Network (LAN) is a network that is confined to a relatively small area. It is generally limited to a geographic area such as writing lab, school or building. It is generally privately owned networks over a distance not more than 5 Km.

MAN (Metropolitan Area Network): MAN is the networks cover a group of nearby corporate offices or a city and might be either private or public.

WAN (Wide Area Network): These are the networks spread over large distances, say across countries or even continents through cabling or satellite uplinks are called WAN.
PAN (Personal Area Network): A Personal Area Network is computer network organized around an individual person. It generally covers a range of less than 10 meters. Personal Area Networks can be constructed with cables or wirelessly.

Network topologies and types
Topology :
Topology refers to the way in which the workstations attached to the network are interconnected.

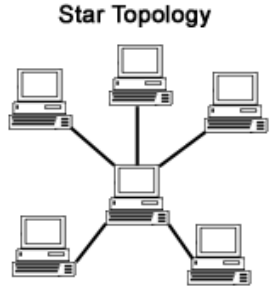
I. **The BUS Topology: -** The bus topology uses a common single cable to connect all the workstations. Each computer performs its task of sending messages without the help of the central server. However, only one workstation can transmit a message at a particular time in the bus topology.



Advantages:
(i) Easy to connect and install.
(ii) Involves a low cost of installation time.
(iii) Can be easily extended.

Disadvantages:-
(i) The entire network shuts down if there is a failure in the central cable.
(ii) Only a single message can travel at a particular time.
(iii) Difficult to troubleshoot an error.

II. **The STAR Topology: -** A STAR topology is based on a central node which acts as a hub. A STAR topology is common in homes networks where all the computers connect to the single central computer using it as a hub.



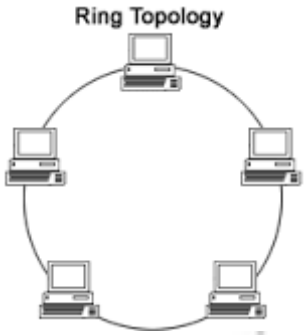
Advantages:
(i) Easy to troubleshoot
(ii) A single node failure does not affect the entire network.
(iii) Fault detection and removal of faulty parts is easier.

(iv) In case a workstation fails, the network is not affected.

Disadvantages:-

(i) Difficult to expand.
(ii) Longer cable is required.
(iii) The cost of the hub and the longer cables makes it expensive over others.
(iv) In case hub fails, the entire network fails.

III. **The RING Topology:-** Each node is connected to two and only two neighbouring nodes. Data is accepted from one of the neighbouring nodes and transmitted onwards to another.



Advantages:

(i) Short cable length
(ii) Suitable for optical fibre
(iii) Fault detection and removal of faulty parts is easier.
(iv) In case a workstation fails, the network is not affected.

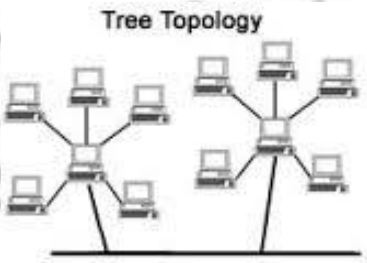
Disadvantages:-

(i) A single node failure causes entire network failure.
(ii) Difficult to diagnose faults.

IV. **The TREE Topology: -** The tree topology combines the characteristics of the linear bus and the star topologies. It consists of groups of star – configured workstations connected to a bus backbone cable.

Advantages:

(i) Eliminates network congestion.
(ii) The network can be easily extended.
(iii) Faulty nodes can easily be isolated from the rest of the network.



Disadvantages:

(i) Uses large cable length.
(ii) Requires a large amount of hardware components and hence is expensive.
(iii) Installation and reconfiguration is very difficult.

Network protocol:

A protocol means the rules that are applicable for a network.
It defines the standardized format for data packets, techniques for detecting and correcting errors and so on.
A protocol is a formal description of message formats and the rules that two or more machines must follow to exchange those messages.

Types of protocols are:

1. **Hypertext Transfer Protocol** (**HTTP**) is a communications protocol for the transfer of information on the intranet and the World Wide Web. HTTP is a request/response standard between a client and a server. A client is the end-user; the server is the web site.
2. **FTP (File Transfer Protocol)** is the simplest and most secure way to exchange files over the Internet. The objectives of FTP are:

   - To promote sharing of files (computer programs and/or data).
   - To encourage indirect or implicit use of remote computers.
   - To shield a user from variations in file storage systems among different hosts.
   - To transfer data reliably, and efficiently.

3. **TCP/IP (Transmission Control Protocol / Internet Protocol)TCP** - is responsible for verifying the correct delivery of data from client to server. Data can be lost in the intermediate

network. TCP adds support to detect errors or lost data and to trigger retransmission until the data is correctly and completely received.

IP - is responsible for moving packet of data from node to node. IP forwards each packet based on four byte destination address (the IP number).

4. **SLIP(Serial Line Internet Protocol)** delivers IP packets over dial-up/ serial lines.
5. **PPP(Point to Point Protocol)** transmits IP packets over serial lines.

Telnet-

It is an older internet utility that lets us log on to remote computer system.

Wireless/Mobile Computing:

Wireless communication is simply data communication without the use of landlines. Mobile computing means that the computing device is not continuously connected to the base or central network.

1. **GSM(Global System for Mobile communication):** it is leading digital cellular system. In covered areas, cell phone users can buy one phone that will work anywhere the standard is supported. It uses narrowband TDMA, which allows eight simultaneous calls on the same radio frequency.

2. **CDMA(Code Division Multiple Access)**: it is a digital cellular technology that uses spread-spectrum techniques. CDMA does not assign a specific frequency to each user. Instead every channel uses the full available spectrum.

3. **TDMA(Time Division Multiple Access):** divides a radio frequency into time slots and allocates slots to multiple calls. This way, single frequency can support multiple, simultaneous data channels.

4. **SIM(Subscriber Identity Module)**

5. **WLL(Wireless in Local Loop) :** WLL is a system that connects subscribers to the **Public Switched Telephone Network(PSTN)** using radio signals as a substitute for other connecting media.

6. **GPRS(General Packet Radio Service):** technology for radio transmission of small packets of data esp. between mobile devices and Internet.

7. **1G, 2G, 3G and 4G networks:** The 'G' in wireless networks refers to the generation of the underlying wireless technology.
    - 1G networks were the first analog cellular systems which were used purely for voice calls and no data services were offered.
    - 2G cellular systems were the first digital systems which offered improved sound quality, better security and higher capacity.
    - 3G systems offer data rates of 384 kbits/s and more.
    - 4G offers no improvement in making calls but very fast web-experience compared to 3G.
8. **Email(Electronic Mail):** Email is sending and receiving messages by computer.

9. **Chat:** Online textual talk in real time is called Chatting.

10. **Video Conferencing**: a two way videophone conversation among multiple participants is called video conferencing.

11. **SMS(Short Message Service):** SMS is the transmission of short text messages to and from a mobile phone, fax machine and or IP address.

12. **EDGE(Enhanced Data rates for Global Evolution)** is a radio based high speed mobile data standard.

13. **Wi-Fi:** refers to **Wireless Fidelity,** which lets you connect to the internet without a direct line from your PC to the ISP.

Web Services:

- **WWW**: The World Wide Web or W3 or simply the Web is a collection of linked documents or pages, stored on millions of computers and distributed across the Internet.
- **Telnet**: an Internet utility that lets you log onto remote computer systems.
- **HTML (Hyper Text Markup Language)**:- HTML is a computer language that describes the structure and behavior of a web page. This language is used to create web pages.
- **XML (eXtensible Markup Language)**:- Extensible Markup Language (XML) is a meta language that helps to describe the markup language.
- **HTTP (Hyper Text Transfer Protocol)**:- A protocol to transfer hypertext requests and information between servers and browsers.
- **Domain Names:** A domain name is a unique name that identifies a particular website and represents the name of the server where the web pages reside.
- **URL**:- The Uniform Resource Locator is a means to locate resources such as web pages on the Internet.
- **Website**: A collection of related web pages stored on a web server is known as a website.
- **Web browser**: A software application that enables to browse, search and collect information from the Web is known as Web browser.
- **Web Servers**: The web pages on the Internet are stored on the computers that are connected to the Internet. These computers are known as web servers.
- **Web Hosting**: - Web Hosting or website hosting is the service to host, store and maintain the websites on the World Wide Web.
- **Web Scripting**: - The process of creating and embedding scripts in a web page is known as Web Scripting. Types of Scripts:-
  (i) **Client Side Scripts**: - Client side scripts supports interaction within a webpage. E.g. VB Script, Java Script, PHP (**H**ypertext **P**reprocessor).
  (ii) **Server Side Scripts**: - Server side scripting supports execution at server – end. E.g. ASP, JSP, PHP

Network Security Concepts:

- **Viruses:** Viruses are programs which replicate and attach to other programs in order to corrupt the executable codes. Virus enters the computer system through an external source and become destructive.
- **Worms**: Worms are also self- replicating programs that do not create multiple copies of itself on one computer but propagate through the computer network. Worms log on to computer systems using the username and passwords and exploit the system.
- **Trojan horse: -** Pretending to be a useful program, it can be used to intrude the computer system in order to exploit the resources. Such a program can also enter into the computer through an email or free programs downloaded through the Internet.
- **Spams**: Unwanted e-mail (usually of a commercial nature sent out in bulk)
- **Cookies:** Cookies are the text messages sent by a web server to the web browser primarily for identifying the user.
- **Firewall**: A firewall is used to control the traffic between computer networks. It intercepts the packets between the computer networks and allows only authorized packets to pass.
- **Cyber Law**: Cyber law refers to all the legal and regulatory aspects of Internet and the World Wide Web.

- **Cyber Crimes**: Cyber crime involves the usage of the computer system and the computer network for criminal activity.
- **Hacking**: Hacking is an unauthorized access to computer in order to exploit the resources or to gain knowledge about the system.
- **Crackers**: Malicious programmers who break into secure systems.
- **Intellectual Property**: A product of intellect that has commercial value, including copyrighted property such as artistic works and ideational property.

OPEN SOURCE TERMINOLOGIES

1. **Free Software:** The S/W's is freely accessible and can be freely used changed improved copied and distributed by all and payments are needed to make for free S/W.
2. **Open Source Software:** S/w whose source code is available to the customer and it can be modified and redistributed without any limitation .OSS may come free of cost but nominal charges has to pay nominal charges (Support of S/W and development of S/W).
3. **FLOSS (Free Libre and Open Source Software) :** S/w which is free as well as open source S/W. (Free S/W + Open Source S/W).
4. **GNU (G**NU's **N**ot **U**nix**) :** GNU project emphasize on the freedom and its objective is to create a system compatible to UNIX but not identical with it.
5. **FSF (Free Software Foundation) :** FSF is a non –profit organization created for the purpose of the free s/w movement. Organization funded many s/w developers to write free software.
6. **OSI (Open Source Initiative)** : Open source software organization dedicated to cause of promoting open source software it specified the criteria of OSS and its source code is not freely available.
7. **W3C(World Wide Web Consortium)** : W3C is responsible for producing the software standards for World Wide Web.
8. **Proprietary Software:** Proprietary Software is the s/w that is neither open nor freely available, normally the source code of the Proprietary Software is not available but further distribution and modification is possible by special permission by the supplier.
9. **Freeware:** Freeware are the software freely available, which permit redistribution but not modification (and their source code is not available). Freeware is distributed in *Binary Form* (ready to run) without any licensing fees.
10. **Shareware:** Software for which license fee is payable after some time limit, its source code is not available and modification to the software are not allowed.
11. **Localization:** localization refers to the adaptation of language, content and design to reflect local cultural sensitivities .e.g. Software Localization: where messages that a program presents to the user need to be translated into various languages.
12. **Internationalization:** Opposite of localization.

OPEN SOURCE / FREE SOFTWARE

1. **Linux:** Linux is a famous computer operating system. Linux server set of program LAMP(Linux, Apache, MySQL, PHP)
2. **Mozilla :** Mozilla is a free internet software that includes
    - a web browser
    - an email client
    - an HTML editor
    - IRC client
3. **Apache server:** Apache web server is an open source web server available for many platforms such as BSD, Linux, and Microsoft Windows etc.
4. **MYSQL :** MYSQL is one of the most popular open source database system.
5. **PostgreSQL :** Postgres SQL is a free software object relational database server .

6. **Pango :** Pango project is to provide an open source framework for the layout and rendering of internationalized text into GTK + GNOME environment..
7. **OpenOffice :** OpenOffice is an office applications suite. It is intended to compatible and directly complete with Microsoft office.
8. **Tomcat :** Tomcat functions as a servlet container. Tomcat implements the servlet and the
9. JavaServer Pages .
10. **PHP(Hypertext Preprocessor) :** PHP is a widely used open source programming language for server side application and developing web content.
11. **Python: Python** is an interactive programming language.
12. Amoeba OS capable of making system calls.

Tips to solve Questions based on Networking

1. **Where Server should be placed**: Server should be placed in the building where the number of computers is **maximum**.
2. **Suggest a suitable cable layout of connection**: A suitable cable layout can be suggested in the following two ways:-
    i. **On the Basis of Server:** First the location of the Server is found out. Server is placed in that building where the number of computers are maximum (According to 80 – 20 rule). After finding the server position, each building distance is compared with the Server building directly or indirectly (taking other building in between). The shortest distance is counted whether it is through directly or indirectly.
    ii. **On the Basis of Distance from each building:** The distance between the each building is compared to all other buildings either directly or indirectly. The shortest distance is counted whether it is directly or through some other building.
3. **Where the following devices be placed**:
    i. **HUB / SWITCH:- In all the wings**
    ii. **REPEATER**: It is used if the distances higher than 70 m. It regenerates data and voice signals.
    iii. **ROUTER**: When one LAN will be connected to the other LAN.

Assignment No. 9
Communication And Network Concepts

Q1. Name two transmission media for networking.

Q2. Give the full form of the following :

i. GSM
ii. XML
iii. FTP
iv. PHP

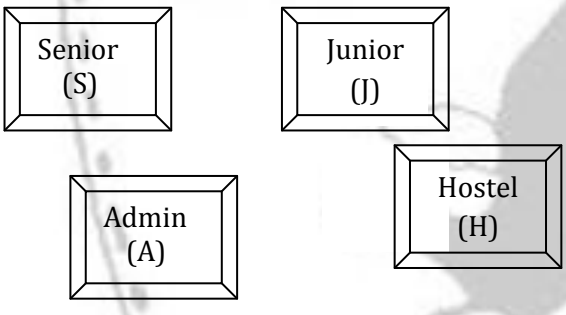
Q3. Differentiate between Hackers and Crackers.

Q4. What is the difference between LAN and WAN?

Q5. What is the purpose of having web browser? Name any one commonly used browser.

Q6. Give two major reasons to have network security.

Q7. Indian Public School in Darjeeling is setting up the network between its different wings. It has four wings named as Senior (S), Junior(J), Admin(A) and Hostel(H):



Wing A to Wing S	100 m
Wing A to Wing J	200 m
Wing A to Wing H	400 m
Wing S to Wing J	300 m
Wing S to Wing H	100 m
Wing J to Wing H	450 m
Number of Computers :	
Wing A	10
Wing S	200
Wing J	100
Wing H	50

Distance between various Wings :

a. Sugga. Suggest a suitable Topology for networking the computer of all the wings.

b. Suggest the most suitable wing to house the server of this school. Justify your answer.

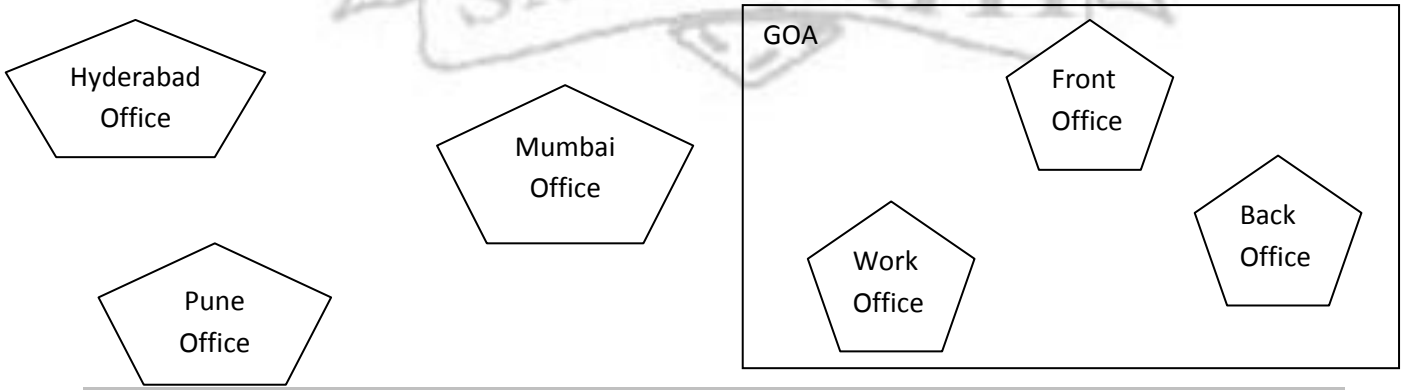
c. Suggest the placement of the following devices with justification :

i. Repeater
ii. Hub/ Switch

d. Mention an economic technology to provide internet accessibility to all the wings.

Q 8."United Group" is planning to start their offices in four cities of India.. The company has planned to set up their head office in Goa in three locations and have named their Goa offices as "Front Office", "Back Office" and "Work office". The company's regional offices are located at "Pune", "Mumbai" and "Hydrabad".

A rough layout of the same is as follows:



Distance between various Wings :		
Place From	**Place to**	**Distance**
Front Office	Back Office	15 K.M.
Front Office	WorkOffice	35 K.M.
Front Office	Pune Office	350K.M.
Front Office	Bombay Office	300 K.M.
Front Office	Hydrabad Office	1000 K.M.
Number of Computers :		
Front Office	150	
Back Office	70	
Work Office	20	
Pune Office	65	
Mumbai Office	90	
Hyderabad Office	100	

a. Suggest a network type(out of LAN, MAN, WAN) for connecting each of the following set of their offices :
   - Front Office and Back Office
   - Front Office and Pune Office

b. Which device will you suggest to be procured by the company for connecting all the computers within each of their offices out of the following devices?
   - Modem
   - Telephone
   - Switch/ Hub

c. Which of the following communication media, will you suggest to be procured by the company for connecting their local offices in Goa for very effective and fast communication ?
   - Ethernet Cable
   - Optical Fiber
   - Telephone Cable

d. Suggest a cable/ Wiring layout for connecting the company's local offices located in Goa. Also, suggest an effective method/ technology for connecting the company's regional office at "Pune", "Bombay" and "Hydrabad".

Handout XI: Database Concepts and Structured Query Language

Introduction to Databases

- A collection of data is referred to as database and a database (management) system is basically a computer based record keeping system.
- Database permits the retrieval of data and continuous modification of data needed for control of operations.

The advantages provided by a database system are:

1. **Reduced data redundancy :**

   Duplication of data is data redundancy. It leads to the problems like wastage of space and data inconsistency.

2. **Shared data:**

   The database allows sharing of data by several users. This means each user may have access to the same database/table/record at a same time.

3. **Secured data:**

   Data is vital to any organization and some of it may be confidential. Confidential data must not be accessed by unauthorized persons. Authentication schemes can be laid down, giving different levels of users, different permissions to access data.

4. **Integrated data:**

   This means that data is accurate and consistent. Checks can be built in to ensure correct values are entered. For example, while placing an order, the quantity must be a number above zero. Also, if an order is placed with a supplier, supplier must exist.

Relational Model :

Relational Model was proposed in 1970 by E.F. Codd of the IBM. It is a dominant model for commercial data processing applications. Nearly, all databases are based on this model. Let us explore this model in detail.

Relation: A relation may be thought of as a set of rows with several columns. A relation has the following properties:

- **Row** is a real world entity or relationship.
- All values in particular column are of same kind.
- Order of columns is immaterial. Order of rows is immaterial.
- Each row is distinct.
- For a row, each column must have an atomic value indivisible).
- For a row, a column cannot have more than one value.
- **Field**- Set of characters that represents specific data element.
- **Domain:** A domain is a pool of values from which the actual values present in a given column are taken.
- **Tuple:** This is the horizontal part of the relation. One row represents one record of the relation. The rows of a relation are also called tuples.
- **Attributes** – The columns of a table are also called attributes. The column is the vertical part of the relation.
- **Degree:** The number of attributes(columns) in a relation determine the degree of a relation.
- **Cardinality** – It is the number of rows (or tuples) in a table.

Keys: Keys come here for expressing difference among rows in terms of their attributes.

- **Primary Key** – It is a column (or columns) in a table that uniquely identifies each row. A

primary key value is unique and cannot be null. There is only one primary key for a table.

- **Candidate key** – It is a column (or columns) that uniquely identify rows in a table. Any of the identified candidate keys can be used as the table's primary key.
- **Alternate key** – Any of the candidate keys that are not part of the primary key is called an alternate key.
- **Foreign key** – It is a column (or a set of columns) that refers to the primary key in another table i.e. it is used as a link to a matching column in another table.

To understand the concept of keys let's take an example, suppose in a school class there are four students who are eligible for being monitor. All these students are called *candidate key*. The student who selected for monitor will be treated as *primary key*.

Now suppose a student who become a monitor is not available in class, so the class teacher choose another student from rest three student who are eligible, for taking the responsibly of monitor. This student is called *alternate key*.

Relational Algebra:

The relational algebra is a collection of operations on relations. The various operations of relational algebra are as following:

(A) The Select Operation:

- The select operation selects tuples from a relation that satisfy a given condition.
- The selection is denoted by lowercase Greek letter σ (sigma).
- Suppose we have table named Items as shown in Fig. (a) and to select those tuples from Items relation where the price is more than 19.00, we shall write

$$\sigma \text{ price} > 9.00 \text{ (Items)}$$

- That means from table Item, select the tuples satisfying the condition price > 9.00. The relation that results from above query is shown in Fig. (b).

Fig (a)

Item#	Item-Name	Price
11	Milk	15.00
12	Cake	5.00
13	Bread	9.00
14	Ice Cream	14.00
15	Cold Drink	8.00

Fig(b)

Item#	Item-Name	Price
11	Milk	15.00
14	Ice Cream	14.00

- In a selection condition, all the relational operators(=, ≠, <, ≤, >, ≥) may be used.
- More than one condition may be combined using connectives and (denoted by∧) and or (denoted by ∨).
- To find those tuples pertaining to prices between 5.00 and 9.00 from relation Items, we shall write:

$$\sigma \text{ price} > 4.00 \wedge \text{price} < 9.00 \text{(Items)}$$

(B) The Project Operation:

- The Project Operation yields a "vertical" subset of a given relation in contrast to the "horizontal" subset returned by select operation.
- The projection lets you select specified attributes in a specified order and duplicating tuples

are automatically removed.

- Projection is denoted by Greek letter pi(π).
- Suppose we have table Suppliers and to project Supplier names and their cities from the relation Supplier, we shall write:
  Π Supp-Name, City (Suppliers)

(C) The Cartesian Product Operation:

- The Cartesian product is a binary operation and is denoted by a cross (x).
- The Cartesian product of two relations a and B is written as A x B.
- The Cartesian product of two relation yields a relation with all possible combinations of the tuples of the two relations operated upon.
- All tuples of first relation are concatenated with all the tuples of second relation to form the tuples of the new relation.

Suppose we have two relations Student and Instructor as following:

Student

Stud#	Stud-Name	Hosteler
S001	Meenakshi	Y
S002	Radhika	N
S003	Abhinav	N

Instructor

Inst#	Inst-Name	Subject
101	K. Lal	English
102	R.L. Arora	Maths

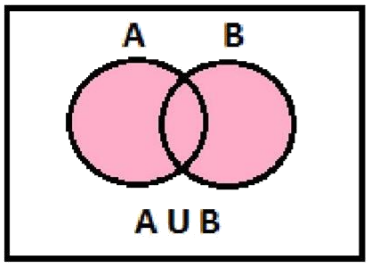
- The cartesian product of these two relations, Students x Instructor, will yield a relation that will have a degree of 6 (3 + 3: sum of degrees of student and Instructor) and a cardinality 6 (3 x 2: product of cardinalities of two relations).
- The resulting relation (Students x Instructor) is as following:

Stud#	Stud-Name	Hosteler	Inst#	Inst-Name	Subject
S001	Meenakshi	Y	101	K. Lal	English
S001	Meenakshi	Y	102	R.L. Arora	Maths
S002	Radhika	N	101	K. Lal	English
S002	Radhika	N	102	R.L. Arora	Maths
S003	Abhinav	N	101	K. Lal	English
S003	Abhinav	N	102	R.L. Arora	Maths

The resulting relation contains all possible combinations of tuples of the two relations.

(D) The Union Operation:

- The union operation requires two relations and produces a third relation that contains tuples from both the operand relations.
- The union operation is denoted by U. Thus, to denote the union of two relation X and Y, we will write as X U Y.
- For a union operation A U B to be valid, the following two conditions must be satisfied by the two operands A and B:
  1. The relations A and B must be of the same degree. That is, they must have the same number of attributes.
  2. The domains of the *i*th attributes of A and the *i*th



attributes of b must be the same.

3. Suppose we have two Drama and Song as following:

Drama

Rollno	Name	Age
13	Kush	15
17	Swati	14

Song

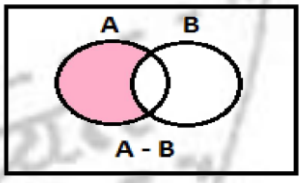
Rollno	Name	Age
2	Manya	15
10	Rishabh	15
13	Kush	15

Result of Song U Drama will be as following:

Rollno	Name	Age
2	Manya	15
10	Rishabh	15
13	Kush	15
17	Swati	14

Note that one duplicating tuple (13, Kush, 15) has been automatically removed

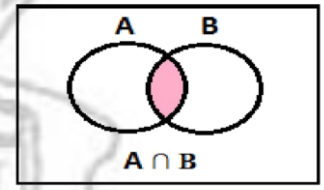
(E) The Set Difference Operation:



- The set difference operation denoted by – (minus) allows us to find tuples that are in one relation but not in another.
- The expression A – B results in a relation containing those tuples in A but not in B.
- Suppose we have two tables Drama and Song as given above.
- Result of Song – Drama will be as following:

Rollno	Name	Age
2	Manya	15
10	Rishabh	15

(F) The Set Intersection Operation:



- The set intersection operation finds tuples that are common to the two operands relations.
- The set intersection operation is denoted by ∩. That means A ∩ B will yield a relation having tuples common to A and B.
- Suppose we have two tables Drama and Song as given above.
- Result of Song ∩ Drama will be as following:

Rollno	**Name**	**Age**
13	Kush	15

Referential Integrity :

Referential Integrity is a system of rules that a DBMS uses to ensure that relationships between records in related table are valid, and that users don't accidentally delete or change related data.

Conditions to set Referential Integrity:

- The matching field from the primary table is a primary key or has a unique index
- The related fields have the same data type
- Both tables belong to the same database

When referential integrity is enforced, given rules should be followed:

- One can't enter a value in the foreign key field of the related table that doesn't exist in the primary key of the primary table.
- One can't delete a record from a primary table, if matching records exist in related table.
- One can't change a primary key value in the primary table, if that record has related records.

For example, suppose Table B has a *foreign key* that points to a field in Table A. Referential integrity would prevent you from adding a record to Table B that cannot be linked to Table A. In addition, the referential integrity rules might also specify that whenever you delete a record from Table A, any records in Table B that are linked to the deleted record will also be deleted. This is called *cascading delete.*

Finally, the referential integrity rules could specify that whenever you modify the value of a linked field in Table A, all records in Table B that are linked to it will also be modified accordingly. This is called *cascading update.*

Structured Query Language (SQL)

- In order to access data within the database, all programmers and users must use, Structured Query Language (SQL).
- SQL is the set of commands that is recognized by all RDBMS.
- The Structured Query Language (SQL) is a language that enables you to create and operate on relational database, which are sets of related information stored in tables.
- The SQL (Structured Query Language) has proved to be a standard language as it allows users to learn one set of commands and use it to create, retrieve, alter, and transfer information regardless of whether they are working on a PC, a workstation, a mini, or a mainframe.

Classification of SQL Statements:

SQL provides many different types of commands used for different purposes. These commands can be divided into following categories:

i. Data Definition Language (DDL): A database scheme is specified by a set of definitions which are expressed by a special language called a data definition language(DDL). CREATE, ALTER, DROP, GRANT, REVOKE etc are some of the DDL commands

ii. Data Manipulation Language (DML): It includes the following

- Retrieval of information stored in database
- Insertion of new information into database
- Deletion of information from database

INSERT INTO, UPDATE, DELETE, SELECT etc are some of the DML commands.

iii. Transaction Control Language (TCL):

iv. Session Control commands

v. System Control commands.

SQL General Data Types

A data type defines what kind of value a column can contain. Each column in a database table is required to have a name and a data type.

The following table lists the general data types in SQL:

Data type	Description
CHAR(n)	Character string. Fixed-length n
VARCHAR(n)	Character string. Variable length. Maximum length n
BOOLEAN	Stores TRUE or FALSE values
INTEGER(p)	Integer numerical (no decimal). Precision p
DECIMAL(p,s)	Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal
NUMERIC(p,s)	Exact numerical, precision p, scale s. (Same as DECIMAL)
FLOAT(p)	Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values

Creating A Table

Creating a basic table involves naming the table and defining its columns and each column's data type. The SQL **CREATE TABLE** statement is used to create a new table.

Syntax:

The basic syntax of the CREATE TABLE statement is as follows –

```
CREATE TABLE table_name(
  column1 datatype,
  column2 datatype,
    .....
  columnN datatype,
  );
```

The following code block is an example, which creates a Persons table –

```
CREATE TABLE Persons (
   PersonID int,
   LastName varchar(255),
   FirstName varchar(255),
   Address varchar(255),
   City varchar(255)
);
```

SQL Constraints:
SQL constraints are used to specify rules for data in a table. Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.
Syntax:

```
CREATE TABLE table_name (
   column1 datatype constraint,
   column2 datatype constraint,
   column3 datatype constraint,
);
```

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted. Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.
The following constraints are commonly used in SQL:

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified
- **INDEX** - Use to create and retrieve data from the database very quickly

Example:

```
CREATE TABLE emp(
   Eno        integer        NOT NULL PRIMARY KEY ,
   Ename      char(20)       NOT NULL,
   Job        char(20)       DEFAULT = 'CLERK',
   Salary     decimal        CHECK (Salary > 2000)
                             );
```

There are two types of constraints:

- **Column constraints:** Apply only to individual columns.
- **Table constraints:** Apply to group of one or more columns.

Example:

```
CREATE TABLE items
(  icode          char(5)      NOT NULL ,
   descp          char(20)     NOT NULL,
   ROL            integer,
   QOH            integer,
   CHECK          (ROL < QOH),                    This is a table constraint.
  PRIMARY KEY  (icode, descp) );
```

Modifying a table:
The SQL **ALTER TABLE** command is used to add, delete or modify columns in an existing table. You should also use the ALTER TABLE command to add and drop various constraints on an existing table.
Syntax:

```
ALTER TABLE table_name ADD column_name datatype;
```

The basic syntax of an ALTER TABLE command to **DROP COLUMN** in an existing table is as follows:

```
ALTER TABLE table_name DROP COLUMN column_name;
```

The basic syntax of an ALTER TABLE command to change the **DATA TYPE** of a column in a table is as follows:

```
ALTER TABLE table_name MODIFY COLUMN column_name datatype;
```

The basic syntax of an ALTER TABLE command to add a **NOT NULL** constraint to a column in a table is as follows:

```
ALTER TABLE table_name MODIFY column_name datatype NOT NULL;
```

Inserting records in a table:

The SQL INSERT INTO Statement is used to add new rows of data to a table in the database.

Syntax:

There are two basic syntaxes of the INSERT INTO statement which are shown below.

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);
```

Here, column1, column2, column3,...columnN are the names of the columns in the table into which you want to insert the data.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

The SQL INSERT INTO syntax will be as follows –

```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
```

Example:

The following statements would create two records in the CUSTOMERS table.

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS
VALUES (2, 'Chetan', 25, 'Delhi', 1500.00 );
```

Deleting a table in SQL

The DROP TABLE statement is used to drop an existing table in a database.

Syntax:

DROP TABLE *table_name*;

Be careful in using this command all information contained in the table is lost with it.

Selecting data from a table

The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax: SELECT column1, column2, columnN FROM table_name;

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

```
SELECT * FROM table_name;
```

WHERE clause:

The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.

The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc., which we would examine later in the subsequent pages.

Syntax:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition];
```

You can specify a condition using the comparison or logical operators like >, <, =, **LIKE, NOT**, etc.

Example: Consider the CUSTOMERS table having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1	Romesh	32	Ahmedabad	2000.00
2	Milan	25	Delhi	1500.00
4	Chetan	25	Mumbai	6500.00
5	Harsh	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000 –

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;
```

This would produce the following result –

ID	NAME	SALARY
4	Chetan	6500.00
5	Harsh	8500.00
6	Komal	4500.00

The AND Operator

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause such that all of them should be satisfied for a record to be chosen.

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 AND age < 25;
```

This would produce the following result –

ID	NAME	SALARY
6	Komal	4500.00

The OR Operator

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause such that even if one of them is true the record is selected for output.

```
SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000 OR age < 25;
```

This would produce the following result –

ID	NAME	SALARY
4	Chetan	6500.00
5	Harsh	8500.00
6	Komal	4500.00

Some other SQL Logical Operators:

BETWEEN
The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.

Example: SELECT * FROM CUSTOMERS WHERE AGE BETWEEN 25 AND 27;

IN
The IN operator is used to compare a value to a list of literal values that have been specified.

Example: SELECT * FROM CUSTOMERS WHERE AGE IN (25, 27);

LIKE
The LIKE operator is used to compare a value to similar values using wildcard operators. The wildcards used for pattern matching are '%'(Matches one or more characters) and '_'(Matches one character).

Example: SELECT * FROM CUSTOMERS WHERE NAME LIKE 'Ko%';
Some more examples of pattern matching:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_%'	Finds values that start with "a" and are at least 3 characters long
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

NOT The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**

IS NULL/ IS NOT NULL
The NULL operator is used to compare a value with a NULL value.

Example: SELECT * FROM CUSTOMERS WHERE AGE IS NOT NULL;

UNIQUE
The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

DISTINCT keyword:
The SQL **DISTINCT** keyword is used in conjunction with the SELECT statement to eliminate all the duplicate records and fetching only unique records. There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only those unique records instead of fetching duplicate records.

Syntax

```
SELECT DISTINCT column1, column2,.....columnN
FROM table_name
WHERE [condition]
```

ORDER BY clause:
The SQL ORDER BY clause is used to sort the data in ascending or descending order, based on one or more columns. Some databases sort the query results in an ascending order by default.
Syntax:

```
SELECT column-list FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort that column should be in the column-list.

GROUP BY clause:
The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. This GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
Syntax:

```
SELECT column1, column2 FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

This clause is normally used in combination with one or more aggregate function to find out the count, sum or average of a particular field in a group.
Example:
Consider the CUSTOMERS table having the following records –

ID	NAME	AGE	ADDRESS	AMOUNT
1	Romesh	32	Ahmedabad	2000.00
2	Harsh	27	Delhi	1500.00
4	Chetan	25	Mumbai	6500.00
2	Harsh	27	Delhi	8500.00
1	Romesh	32	Ahmedabad	4500.00

The command to find out the total amount of each customer would be:

```
SELECT NAME, SUM(AMOUNT)
FROM CUSTOMERS
GROUP BY NAME;
```

And the output will be:

NAME	SUM(AMOUNT)
Chetan	6500.00
Harsh	10000.00
Romesh	6500.00

The other aggregate functions that can be used are count(), avg(), max(), min().

HAVING clause:

The HAVING Clause enables you to specify conditions that filter which group results appear in the results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

Syntax:

```
SELECT column1, column2 FROM table1
WHERE [ conditions ] GROUP BY column1
HAVING [ conditions ];
```

Deleting records from a table in SQL

The SQL DELETE Query is used to delete the existing records from a table. You can use the WHERE clause with a DELETE query to delete the selected rows, otherwise all the records would be deleted.

Syntax:

```
DELETE FROM table_name
WHERE [condition];
```

Modifying records

The SQL **UPDATE** Query is used to modify the existing records in a table. You can use the WHERE clause with the UPDATE query to update the selected rows, otherwise all the rows would be affected.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

You can combine N number of conditions using the AND or the OR operators.

SQL Joins

The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Consider the following two tables –

CUSTOMERS Table

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

ORDERS Table

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08	3	3000
100	2009-10-08	3	1500
101	2009-11-20	2	1560
103	2008-05-20	4	2060

If we join these two tables in our SELECT statement as shown below:
SELECT ID, NAME, AGE, AMOUNT
FROM CUSTOMERS, ORDERS
WHERE CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

It will produce the following result:

ID	NAME	AGE	AMOUNT
3	Kaushik	23	3000
3	Kaushik	23	1500
2	Khilan	25	1560
4	Chaitali	25	2060

CREATE VIEW Command
A view is a virtual table with no data of its own. It is like a window through which you can view the data of another table called the base table. It can be created with the following command:
CREATE VIEW custbelow
AS SELECT *
FROM customers
WHERE age <25;
The above view will have the details of all customers from the Customers table whose age is below 25.

Assignment No. 10.1
Database Concepts and Structured Query Language

Q1. What is a database system? What is its need?

Q2. What is a relation?

Q3. What is the difference between Union and Cartesian Product of two relations?

Q4. What are constraints? How many types of constraints are there in SQL?

Q5. What is a Primary Key?

Q6. What is an Alternate Key?

Q7. What is a candidate Key?

Q8. What do you understand by degree of relation?

Q9. What do you understand by cardinality of relation?

Q10. What is the difference between Group BY and Order BY clauses?

Q11. What is the difference between Where and Having clauses?

Q12. Explain the syntax and use of the following statements in SQL:

i. Alter Table
ii. Drop Table
iii. Update
iv. Delete

Q13. Define the following :

i. Attribute
ii. Tuple
iii. Foreign Key
iv. Data redundancy
v. DDL
vi. DML

Structured Query Language
Assignment No : 10.2

Write SQL commands for the following :

Q. 1 Create a table with the following specifications:

TABLE : BANK

Field name	**Type**	**Description**
Acc_no	Int	PRIMARY KEY
CName	char(20)	NOT NULL
Address	Char(30)	NOT NULL
Ph_No	Int	Can have NULL values
Amount	FLOAT	Minimum 5000
DOO	Date	Date of opening the account ; NOT NULL
Type	Char(1)	Type of account (S->savings, C->current, R->recurring)

Q.2 Insert the following records in the above table (*write commands for any three*):

Acc_no	**CName**	**Address**	**Ph_No**	**Amount**	**DOO**	**Type**
1	Karan	22/7, green Park, Delhi	321345	45000	12/01/1975	S
2	Puneet	3/1, Friends colony, Delhi	324567	89000	01/01/1985	R
3	Anirban	12/7, GK-1, New Delhi	432664	78000	11/10/1988	C
4	Yatin	11, A Block, Rohini, Delhi	256997	100000	06/08/1999	S
5	Sonia	32/4, green Park, Delhi	123456	8921	03/08/1998	S
6	Sophia	13/11, Friends colony, Delhi	NULL	45697	05/08/1996	R
7	Nikhil	112/7, GK-1, New Delhi	234890	14752	12/01/1975	C
8	Tarun	21, A Block, Rohini, Delhi	NULL	5500	01/01/1986	R
9	Jisha	113, A Block, Rohini, Delhi	126790	78451	11/10/1978	R
10	Tanmay	1/7, GK-1, New Delhi	345678	70000	06/08/1989	S

Considering the above table write SQL commands for the following :

Q 3. Display the record of all the customers.

Q 4. Display the record of all the customers sorted in descending order of their amounts.

Q 5. To count total number of customers for each type. Q 6. To display the total money deposited in the bank.

Q 7. To list out the names of all the customers, whose name starts with **S**.

Q 8. To display the Acc_no and names of all those customers, who do not have phone. Q 9. To display total amount in each type of account.

Q 10. To list the names and addresses of all the customers staying in Rohini.

Structured Query Language
Assignment No. 10.3

Write SQL commands for the following :
Q. 1 Create a table with the following specifications :

TABLE : SUPPLIER

Field name	**Type**	**Description**
S_no	Char(2)	PRIMARY KEY
PName	char(20)	NOT NULL
SName	Char(20)	NOT NULL
Qty	Int	Should be in the range of 100 to 500
Price	FLOAT	Should not be more than 50
City	Char(15)	Should be any of the four metropolitan cities.

Q.2 Insert the following 10 records in the above table (Write commands for any three):

S_No	**PName**	**SName**	**Qty**	**Price**	**City**
S1	Bread	Britania	150	8.00	Delhi
S2	Cake	Britania	250	20.00	Mumbai
S3	Coffee	Bru	170	45.00	Delhi
S4	Chocolate	Amul	380	10.00	Kolkata
S5	Souce	Kissan	470	36.00	Chennai
S6	Maggi	Nestle	340	10.00	Mumbai
S7	Biscuit	Britania	560	21.00	Delhi
S8	Jam	Kissan	220	40.00	Kolkata
S9	Tea	TATA	345	45.00	Chennai

Considering the above table answer the questions that follow :

Q 3. Display the record of all the suppliers.

Q 4. Display data of all the products whose quantity is between 170 and 370.

Q 5. Give Sname for the products whose name starts with 'C'.

Q 6. Sname, Pname,Price for all the products whose quantity is less than 200.

Q 7. To list out the names of all the products, whose name ends with **S**.
Q 8. To display S_No, Pname, Sname, Qty in descending order of qty.

Q 9. To count the number of distinct cities.

Q 10. To list the Snames, Qty, Price and value (Qty*Price) of each product.

Structured Query Language
Assignment No : 10.4

Write SQL commands for the following :

Q. 1 Create a table with the following specifications :

TABLE : CLUB

Field name	**Type**	**Description**
Coach_id	int	PRIMARY KEY
CName	char(20)	NOT NULL
Age	int	Should be between 28 to 40
Sports	Char(20)	Should be either of Karate, Squash, Basket Ball, Swimming
Pay	FLOAT	Should be between 1000 to 2500
DOA	Date	Date of Appointment
Sex	Char(1)	M- male, F- Female

Q.2 Insert the following 10 records in the above table (Write commands for any three) :

Coach_id	**CName**	**Age**	**Sports**	**Pay**	**DOA**	**Sex**
1	Karan	35	Karate	1000	12/01/1995	M
2	Puneet	34	Karate	1200	01/01/1995	M
3	Anirban	34	Squash	2000	11/10/1998	M
4	Yatin	33	Basket Ball	1500	06/08/1999	M
5	Sonia	36	Swimming	1100	03/08/1998	F
6	Sophia	36	Swimming	2200	05/08/1996	F
7	Nikhil	39	Squash	2400	12/01/1995	M
8	Tarun	37	Karate	2500	01/01/1996	M
9	Jisha	41	Swimming	2300	11/10/1998	F
10	Tanmay	37	Basket Ball	2200	06/08/1999	M

Considering the above table answer the questions that follow :

Q 3. To display all the information about the swimming coaches in the club.

Q 4. Display the names of all the coaches with their date of appointment in descending order.

Q 5. To display a report, showing coach name, pay, age and bonus(15% of pay) for all the coaches. Q 6. To display the total number of coaches in the club.

Q 7. To count the total number of male and female coaches. Q 8. To display total salary given to all the coaches.

Q 9. To display the salaries of all the coaches appointed after {01/01/1998}. Q 10. To increment the Pay of all the coaches by 500, earning less than 1500.

Structured Query Language
Assignment No : 10.5

Write SQL commands for the following :

Q. 1 Create a table with the following specifications :

TABLE : VOTER

Field name	Type	Description
V_no	int	PRIMARY KEY
VName	char(20)	NOT NULL
Address	Char(30)	NOT NULL
Ph_No	Int	Can have NULL values
Age	int	Should be >=18

Q.2 Insert the following 10 records in the above table (Write commands for any three) :

V_no	VName	Address	Ph_No	Age
1	Kiran	22/7, C R Park, Delhi	321345	32
2	Puneeta	113/1, Friends colony, Delhi	324567	18
3	Anubhav	12/7, GK-1, New Delhi	432664	47
4	Yatin	11, A Block, Rohini, Delhi	256997	22
5	Suniana	32/4, green Park, Delhi	123456	32
6	Sophia	13/11, Friends colony, Delhi	NULL	89
7	Nakul	112/7, GK-1, New Delhi	234890	41
8	Tarang	21, A Block, Rohini, Delhi	NULL	85
9	Nisha	113, A Block, Rohini, Delhi	126790	78
10	Tanmay	1/7, GK-1, New Delhi	345678	65

Considering the above table answer the questions that follow :

Q 3. Display the record of all the voters.

Q 4. Display the V_No, Vname, age of all the voters sorted in descending order of their names.

Q 5. To count total number of voters, staying in Friends colony.

Q 6. To change the phone number of the person whose voter number are 5 to 326768.

Q 7. To list out the names of all the voters who are more than 75 years of age.

Q 8. To display name and address of all the voters who are of more than 60 years of age.

Q 9. To count, total number of voters.

Q 10. To list the names and addresses of all the voters whose name start with **T** and end with **g.**

Structured Query Language
Assignment No : 10.6

Write SQL commands for the following :

Q. 1 Create a table with the following specifications :

TABLE : STUDENT

Field name	**Type**	**Description**
No	int	PRIMARY KEY
Name	char(20)	NOT NULL
Stipend	Float	Value should range bet. 400 & 600
Stream	Char(15)	Medical, Commerce, Humanities, Non Medical
AvgMarks	Float	Value should range bet. 0 & 100
Grade	Char(1)	Marks Grade >=75 A 50- 74 B <=49 C

Q.2 Insert the following 10 records in the above table (write commands for any three) :

No	**Name**	**Stipend**	**Stream**	**AvgMark**	**Grade**
1	Diwakar	400.00	Medical	78.5	A
2	Deepa	550.00	Commerce	89.5	A
3	Divya	600.00	Non Medical	68.6	B
4	Arun	475.00	Commerce	49.0	C
5	Sabina	425.00	Medical	40.25	C
6	John	600.00	Humanities	94.5	B
7	Robert	500.00	Commerce	88.5	A
8	Rubina	550.00	Non Medical	92.5	A
9	Vikas	575.00	Medical	67.5	B
10	Mohan	600.00	Humanities	73.0	A

Considering the above table answer the questions that follow :

Q 3. Display the records of all non medical students.

Q 4. List all the students sorted by their Average marks.

Q 5. To display a report, listing name, stipend, stream and amount of stipend received in a year assuming stipend is paid every month.

Q 6. To count the number of students with grade 'A'.

Q 7. To display average of average marks scored by the students of each stream.

Q 8. To count the number of medical students who are getting stipend greater than 500.

Q 9. To display total amount of stipend given to all the students per annum.

Q 10. To count the number of students with grade **A**.

Program List

General Instructions:

1. Practical files should contain program listings and outputs.
2. Each program should be headed with **Program number**.
3. Header comment of each program should contain : Name of the Program : _____

	Programmers Name:_____	
	Date: _____	
Prog#	**PROGRAMS**	**Date of Submission**
1.	Create a header file date.h to define a structure date with the following data members: dd int mm int yy int Now write the following functions: i. To accept 3 integers as parameters and set them to respective members of a date variable. ii. To verify the validity of a date and return 1 if valid and 0 if not.	March
2.	Write a menu driven program to execute the following functions : • Reads a decimal number and returns its octal equivalent. • Reads an octal number and returns its decimal equivalent.	March
3.	Define a structure **student** with the following members : Rollno **int** Name **char** [20] Marks **float**[5] Total **float** Per **float** DOB **date** (use date.h) Now, Write the following functions : 1. *To **input** rollno, name, marks in 5 subjects, date (use input and validate( ) from **date.h**) and calculate total & percentage of a student* 2. *To **display** the data of the students.* Write a menu driven function to **execute** the above functions for 5 students.	March
4.	WAP using classes to Add, Subtract and Multiply two 2 – D Arrays (Matrices).	April
5.	Define a class matrix with a single data member: a 2-D integer array and public member functions to do the following tasks: 1. *Input the data in the matrix.* 2. *Display the matrix.* 3. *Sum of Rows.* 4. *Sum of columns* 5. *Sum of all the elements* 6. *Sum of Left diagonal*	April

	7. Sum of Right diagonal *8. Print upper half of the diagonal* *9. Print lower half of diagonal* 10. Display transpose of the matrix	
6.	Define a class **Account** with the following **Data members** : Account Number **int** (Automatically Generated) Customer Name **char** [20] Type of Account **char (S ->** Saving Acct, **C** -> Current Account) Balance **float (**Minimum balance -> 1000) **Member Functions** : *1) Constructor to initialise the data members.* *2) To input the data members.* *3) To withdraw money after checking the balance.* *4) To display the data members.* Write a menu driven program to execute the above class for 5 account holders.	April
7.	Define a class **Product** with the following **Data members** : Product Number **int** (Automatically Generated) Product Name **char** [20] Price **float** **Member Functions** : *1) To input the data members.* *2) To display the data members.* **Global Functions:** *1) A function to sort the records (using Insertion Sort)of all products in the ascending order of their price.* *2) To insert a new record in the sorted array without altering the order of the records.* *3) To search for a Product(Using Binary Search)in the array of products.* Write a menu driven program to execute the above class and functions for 5 Products	July
8.	WAP to swap two strings from an array of strings using pointers.	July
9.	WAP to count the number of words in a text file.	May
10.	WAP to copy the contents of one text file to another.	May
11.	Define a class **Tour** with the following **Data members** : TCode int NoofAdults int NoofKids int Kilometers int	May

	TotalFare float **Member Functions** : *1) To input the data members.* *2) To display the data members.* *3) To write the data to a Binary File* *4) To read the data from the Binary file* *5) To count the number of records where NoofKids is 0.* Write a menu driven program to execute the above class.	
12.	WAP using classes/ structures to : 1. Create a Linked List. 2. Display the List 3. Insert a Node in it. 4. Delete a Node from it.	Aug
13.	WAP using classes/ structures to : (USING ARRAYS) 1. Create a Stack 2. Push an element into it. 3. Pop an element out of it.	Aug
14.	WAP using classes/ structures to : (USING LINKED LIST) 1. Create a Stack 2. Push an element into it. 3. Pop an element out of it.	Aug
15.	WAP using classes/ structures to : (USING ARRAYS) 1. Create a Queue 2. Insert an element into it. 3. Delete an element from it.	Aug
16.	WAP using classes/ structures to : (USING LINKED LIST) 1. Create a Queue 2. Insert an element into it. 3. Delete an element from it.	Aug

Class X|| (Practical) - C++

Duration: 3 hours

Total Marks : 30

1. Programming in C++ 10

One programming problem in C++ to be developed and tested in Computer during the examination. Marks are allotted on the basis of following:

		6 Marks
Documentation/Indentation	:	2 Marks
Output presentation	:	2 Marks

Notes: The types of problem to be given will be of application type from the following topics

- Arrays (One dimensional and two dimensional)
- Class(es) and objects
- Stack using arrays and or linked implementation
- Queue using arrays (circular) and or linked implementation
- Binary File operations (Creation, Displaying, Searching and modification)
- Text File operations (Creation, Displaying and modification)

2. SQL Commands 05

Five Query questions based on a particular Table / Relation to be tested practically on Computer during the examination. The command along with the result must be written in the answer sheet.

3. Project Work 05

The project has to be developed in C++ language with Object Oriented Technology and also should have use of Data files. (The project is required to be developed in a group of 2-4 students)

- Presentation on the computer
- Project report (Listing, Sample, Outputs, Documentations)
- Viva

  * 1 mark is for innovation while writing programme.

4. Practical File 06

Must have minimum 20 programs from the following topics

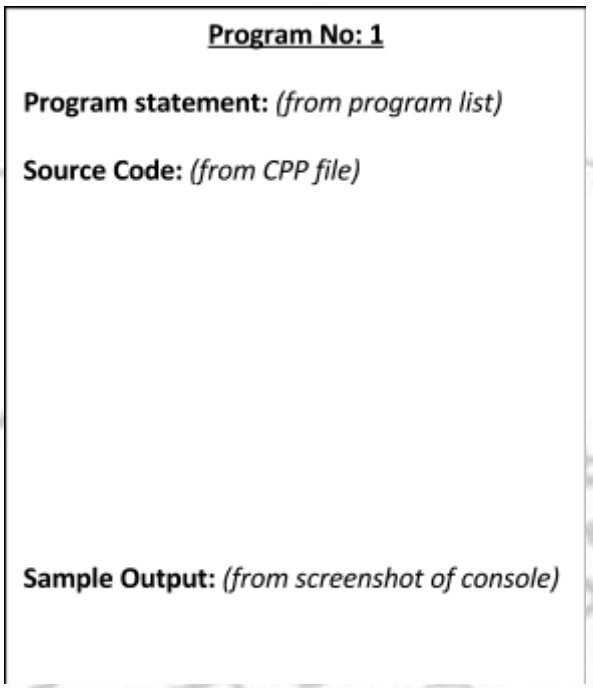
- Arrays (One dimensional and two dimensional, sorting, searching, merging, deletion' & insertion of elements)
- Class(es) and objects
- Stacks using arrays and linked implementation
- Queue using arrays & linked implementation (circular aslo).
- File (Binary and Text) operations (Creation, Updation, Query)
- Any computational Based problems
- 15 SQL commands along with the output based on any table/relation:

5. Viva Voce 04

Viva will be asked from syllabus covered in class X|| and the project developed by student.

Important instructions for Practical Examination(Preboard and Board)

1. The practical file and project report has to be submitted before the commencement of Second Term exam.
2. Printouts of all 17 programs from the program list and handwritten solutions for SQL assignments 10.2 to 10.6 from the Smartskills need to be filed in a school file, in the proper order for submission.
3. The program list copy should be filed on top.
4. The format for printing the programs is as follows:



Keep in mind the following points while printing:

1. Print all programs on a new page. Do not begin a new program from the middle of a page.
2. Use the same font specifications throughout the practical file.

Project File Printing:

- The project has to be printed before the Second Term exam.
- The project has to be printed only after approval by the teacher.
- The project file has to be printed individually for each group member.
- The project report has to bound spirally.
- The project file should have the following pages:
    1. Cover page(with Project Name, Subject, Your name, Class, Board Roll NO(leave blank space for this))
    2. Index
    3. Certificate of Authenticity
    4. Acknowledgement
    5. Project Synopsis
    6. Project Requirements

7. Flowchart/ Project Plan
8. Source Code
9. Sample Output
10. Validations / Limitations
11. Bibliography

- Each new section should be separated from the previous one with a proper separator sheet.
- Use the same font specifications throughout the project report.

The certificate of Authenticity should be typed as follows:

References for Study Material and Notes

www.mycbseguide.com
www.tutorialspoint.com
www.cplusplus.com/