Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in themodel answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills).
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgment on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

1. (A) Attempt any SIX of the following: **MARKS 12**

(a) State any four application of object oriented programming.
(Any four, each application-1Mark,)
[Note: Any relevant application can be considered]

Ans:

Applications Object oriented programming are as follows:

- Real time systems
- Simulation and modeling
- Object oriented databases
- Hypertext, hypermedia and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems

(b) Define constructor. State any two type of constructor.
(Definition 1Mark, Any two types each-1/2 Mark)

Ans: Definition- A constructor is a special member function whose task is to initialize the objects of its class.
Types of constructor:-

- Default constructor
- Parameterized constructor
- Copy constructor
- Constructor with default value
- Multiple constructor

(c) State any two access specifier with example.
(Any two from following list, Each access specifier with example-1Mark)
[Note: any relevant example can be considered]

Ans: List of access specifiers:-
1. Private **2. Protected** **3. Public**

Example:-
```
class student
{
 private:
        int roll_no;
protected:
        int marks;
public:
        void getdata();
};
```

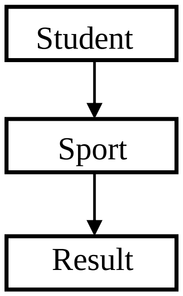
(d) Define constructor overloading.
(Definition-2Marks)

Ans:
Defining more than one constructor function in a class is called as constructor overloading.

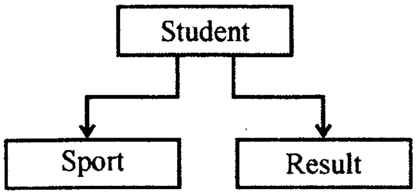
(e) Give the types of inheritance for following diagram:
(Each correct identified inheritance type-1Mark)

Ans:

(i)



(ii)



(i) Multilevel Inheritance
(ii) Hierarchical Inheritance

(f) What is pointer? Give any example.
(Pointer definition-1Mark, Example-1Mark)

Ans:
Definition: - A pointer is a variable that stores address of another variable of similar data type.
Example: -
```
int *ptr, a;
ptr=&a;
```

(g) Write a syntax to create a pointer for object.
(State syntax-2 Marks)

Ans: Syntax: - class_name *pointer_variable;

(h) State any two types of polymorphism.
(State any two each type-1Mark)

Ans:

1. Compile Time Polymorphism
2. Run Time Polymorphism

(B) Attempt any TWO of the following: **MARKS 08**

(a) Explain any two visibility modes with example.
(Any two modes-each mode description 1Mark, Example 1Mark)

Ans:

Visibility modes:-

- Private: when a base class is privately inherited by a derived class, 'public' and 'protected' members of base class become 'private' members of derived class and therefore the public and protected members of base class can be accessed by the member functions of the derived class. 'Private' members of base class are not inherited in derived class.

Example:-

```
class base
{
private:
int a;
protected:
int n;
public:
int c;
void accept()
{
cin>>a;
}
};
class derived:private base
{
private:
int d;
public:
void getdata()
{
accept();
cin>>b>>c>>d;
}
}d;
void main()
{
d.getdata();
}
```

- Public: when a base class is publicly inherited by a derived class, 'public' members of base class becomes 'public' members of derived class and protected members of base class becomes protected members of derived class. 'Private' members of base class are not inherited in derived class.

Example:-

```
class base
{
private:
int a;
protected:
int b;
public:
int c;
void accept()
{
cin>>a;
}
};
class derived:public base
{
private:
int d;
public:
void getdata()
{
cin>>b>>c>>d;
}
}d;
```

```
void main ()
{
d.accept();
d.getdata();
}
```

- Protected: when a base class is inherited in derived class in protected mode, 'protected' and 'public' members of base class becomes protected members of derived class. 'Private' members of base class are not inherited in derived class.

Example:-

```
 class base
{
private:
int a;
protected:
int n;
public:
int c;
void accept()
{
cin>>a;
}
};
class derived:protected base
{
private:
int d;
public:
void getdata()
{
accept();
cin>>b>>c>>d;
}
}d;
void main()
{
d.getdata();
}
```

(b) Explain any two types of constructor with syntax and example.
(Any two types each type description-1Mark, Syntax-1/2 Mark, Example-1/2Mark)

Ans:

Types of constructor:-

- Default constructor: - A constructor that does not accept parameters is called as default constructor.
  Syntax:- constrctor_name();
  Example:- class account

```
{
        int accno,bal;
public:
        account()
        {
        accno=1;
        bal=1000;
        }
};
```

- Parameterized constructor: - A constructor that accepts parameters is called as parameterized constructor.
  Syntax:- constrctor_name(datatype parameter1, datatype parameter1,…, datatype parameter n);
  Example:- class account

```
{
        int accno,bal;
public:
        account(int a,int b)
        {
        accno=a;
        bal=b;
        }
};
```

- Copy constructor:-A constructor that is used to declare and initialize an object from another object is called as copy constructor.
  Syntax:- constructor_name(class_name &object_name);
  Example:- class account

```
                {
                        int accno, bal;
                public:
                        account( )
                        {
                                accno =10001;
                                bal = 5000;
                        }
                        account(account &a)
                        {
                        accno=a.accno;
                        bal=a.bal;
                        }
                };
        void main( )
            {
                account b; // default constructor gets invoked;
                account c =b;  //copy constructor for c gets invoked
            }
```

- Constructor with default value:- A constructor that accepts parameters and in which some parameters can be declared with default value is called as constructor with default value.

  Syntax:- constrctor_name(datatype parameter1, datatype parameter1=value);

  Example:- class account

```
                {
                        int accno,bal;
                public:
                        account(int a,int b=1000)
                        {
                        accno=a;
                        bal=b;
                        }
                };
```

(c) Explain the concept of destructor in a class with example.
(Description of destructor-2Marks, Example-2Marks)

Ans:

Description:-

- A destructor is used to destroy the objects that are created by a constructor.
- It is member function whose name is same as the class name but proceeded by a tilde (~).
- A destructor never takes parameters and it does not return any value.
- It will be invoked by the compiler upon exit from the program (or block or function) to clean up storage that is no longer accessible.

- Syntax:- ~destructor_name()
```
                  {
                  }
```

Example:-
```
class student
{
public:
        student()
        {
                cout<<"object is initialized";
        }

        ~student()
        {
                cout<<"object destroyed";
        }
};
```

2. Attempt any FOUR of the following: **MARKS 16**

(a) Differentiate between OOP and POP.

(Any four points for each point -1Mark)

[Note: any other relevant point can be considered]

Ans:

Object oriented programming	Procedure oriented programming
1. Focus is on data.	1. Focus is on procedure.
2. Programs are divided into multiple objects.	2. Large programs are divided into multiple functions.
3. Data is hidden and cannot be accessed by external functions.	3. Data move openly around the system from function to function.
4. Objects communicate with each other through function.	4. Functions transform data from one form to another by calling each other.
5. Employs bottom-up approach in program design	5. Employs top-down approach in program design.
6. Object oriented approach is used in C++ language.	6. Procedure oriented approach is used in C language.

(b) Write a program to create a class "student" having data member as name. roll no. and percentage to read and display details for 10 students.

(Declaration of class with proper members-2Marks, creating array of object and calling functions- 2Marks)

Ans:

```
#include<iosream.h>
class student
{
char name[10];
int rollno;
float percentage;
public:
void getdata()
{
cin>>name>>rollno>>percentage;
}
void putdata()
{
cout<<name<<rollno<<percentage;
}
};
void main()
{
student s[10];
int i;
for(i=0;i<10;i++)
s[i].getdata();
for(i=0;i<10;i++)
s[i].putdata();
}
```

(c) Explain:

(i) Static member function (ii) Friend function

(Description of static member function-2Marks, description of friend function-2Marks)

Ans:

(i) Static member function: - a static member function can have access to only other static variable or functions declared in the same class. It can be called using the class name instead of object. It can be declared inside the class with static keyword placed before return type.

Syntax for declaration:-

```
static return_type function_name ()
{
}
```

Syntax for calling static function:- class_name::function_name( );

(i) Friend function: - The private members of a class cannot be accessed from outside the class but in some situations two classes may need access of each other's private data. So a common function can be declared which can be made friend of more than one class to access the private data of more than one class. The common function is made friendly with all those classes whose private data need to be shared in that function. This common function is called as friend function. Friend function is not in the scope of the class in which it is declared. It is called without any object. The class members are accessed with the object name and dot membership operator inside the friend function. It accepts objects as arguments.

Syntax:- friend return_type function_type(parameter1,parameter2,…,parameter n);

Syntax for calling friend function: - function_name(parameter1,parameter2,…,parameter n);

(d) Write a program to search the given element in the entered array using pointer.
(Correct logic for searching an element using pointer -4Marks)
[Note:- Array initialized at the time of declaration or accepting array elements without pointer can be considered]

Ans:

```
#include<iostream.h>
#include<conio.h>
void main()
{
    int a[5], i,*ptr, no, flag=0;
    clrscr();
ptr=&a[0];
    cout<<"\n enter array elements: \n";
        for(i=0; i<5; i++)
                {
                cin>>*ptr;
                ptr++;
                }
cout<<"enter element to be searched:\n";
        cin>>no;
        ptr=&a[0];
        for(i=0; i<5; i++)
        {
                if(*ptr==no)
                {
                    flag=1;
                    break;
                }
                    ptr++;
                }
        if(flag == 0)
        cout<<" number is not present in the array.\n";
        else
        cout<"number is present in the array";
getch();
```

}

(e) Differentiate between compile time & run time polymorphism.

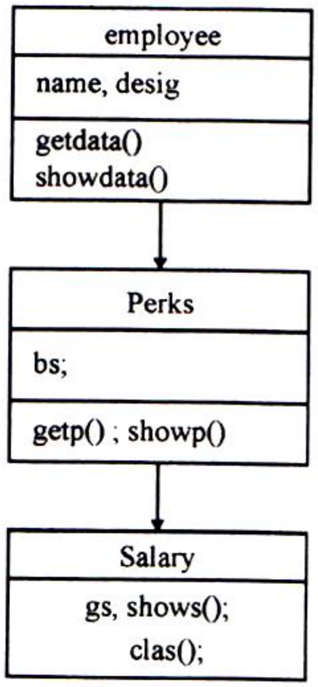
(Any four points for each point -1Mark)

[Note: any other relevant point can be considered]

Ans:

Compile time Polymorphism	**Runtime Polymorphism**
It means that an object is bound to its function call at compile time i.e. linking of function call to its definition at compile time.	It means that selection of appropriate function is done at run time i.e. linking of function call to its definition at run time.
Functions to be called are known well before	Function to be called is unknown until appropriate selection is made.
This does not require use of pointers to objects	This requires use of pointers to object
Function calls are faster	Function calls execution are slower
It is also referred as early binding or static binding.	It is also referred as late binding or dynamic binding.
e.g. overloaded function call It is implemented by function overloading or operator overloading	e.g. virtual function It is implemented by virtual functions.

(f) Write a program to show use of multilevel inheritance for following diagram to calculate the gross salary. gs = bs + 0.5 * bs + 0.6 * bs;



(Employee class definition-1Mark ,Perks class definition-1Mark,Salary class definition-1Mark,Main function-1Mark)
[Note: Any other correct implementation of multilevel inheritance can be considered.]

Ans:

```
#include<iostream.h>
class employee
{
char name [10], desig [10];
public:
void getdata()
{
cin>>name>>desig;
}
void showdata()
{
```

```
cout<<name<<desig;
}
};

class Perks: public employee
{
protected:
int bs;
public:
void getp()
{
cin>>bs;
}
void showp()
{
cout<<bs;
}
};

class Salary: public Perks
{
int gs;
public:
void clas()
{
gs = bs + (0.5* bs) +( 0.6 * bs);
}
void shows()
{
cout<<gs;
}
};

void main()
{
Salary sa;
sa.getdata();
sa.getp();
sa.clas();
sa.showdata();
sa.showp();
sa.shows();
}
```

3. Attempt any FOUR of the following: **MARKS 16**

(a) Explain structure of C++ program with example.
(Structure 1 Mark; Description -2 Marks; Example -1 Mark)

Ans:

General C++ program has following structure.

INCLUDE HEADER FILES
DECLARE CLASS
DEFINE MEMBER FUNCTIONS
DEFINE MAIN FUNCTION

Description:-

1. **Include header files**
   In this section a programmer include all header files which are require to execute given program. The most important file is *iostream.h* header file. This file defines most of the C++ statements like *cout* and *cin*. Without this file one cannot load C++ program.

2. **Declare Class**
   In this section a programmer declares all classes which are necessary for given program. The programmer uses general syntax of creating class.

3. **Define Member Functions**
   This section allows programmer to design member functions of a class. The programmer can have inside declaration of a function or outside declaration of a function.

4. **Define Main Functions**

This section the programmer creates object and call various functions writer within various class.

Example:

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

```
class example
    {
        int mark1, mark2;
    public:
        void accept( )
        {
                cout<<"Enter Marks for subject 1 and Subject 2";
                cin>>roll>>name;
        }
        void display( )
        {
                cout<<"Roll Number is "<<roll;
                cout<<"\nName is "<<name;
        }
    };
void main( )
    {
example d;
        clrscr( );
        d.accept( );
        d.display( );
        getch( );
    }
```

(b) Explain syntax for declaring the function inside the class and outside the class with example.

(Inside Declaration- 2 Marks; Outside Declaration- 2 Marks)

Ans:

- Syntax for Declaring Function Inside the class: -

```
class class_name
  {
        public:
        return_type function_name(argument(s) )
        {
                Function body;
        }
        };
```
In this method one simply declares and defines the function within a class. It does not require any special operator or explicit declaration. Entire function is written in a class.

- Syntax for Declaring Function Outside the Class

```
class class_name
  {
      public:
      return_type function_name(argument(s));        //Declaring a Function in class
};

return_type class_name :: function_name(argument(s))   //Defining a Function
  {
         Function body;
  }
```
In this method a programmer declare as many functions as he wants depending upon the need of a program. Once done with it a programmer can then define those functions outside of a class using scope resolution operator i.e. (**::**).

Advantage of having outside declaration is to have as many functions as we want and we can define it anywhere in a program.

(c) Explain the concept of parameterized constructor with example.

(Description of parameterized constructor- 2 Marks; Example- 2 Marks; Syntax is optional; any one method shall be considered)

Ans:

Description

Parameterized Constructor: - Constructor which accepts one or more value(s) as argument(s)/ parameter(s) is known as parameterized constructor. These constructors gets invoked when an object is created and this object shall be supplemented with appropriate numbers of arguments. Syntax for parameterized constructor is as follows.

Syntax: -
```
class class_name
        {
                        Data member(s);
```

```
                public:
                        class_name(arg1, arg2,… ) // parameterized constructor
                        {
                                ….
                                ….
                        }
        };
Example:-
class student
        {
                        int roll_number;
                        float per;
                public:
                        student(int r, float p )
                        {
                                roll_number = r;
                                per = p;
                        }
                        void display( )
                        {
                                cout<<"\n Roll Number is "<<roll_number;
                                cout<<"\n Percentage is "<<per;
                        }
        };
void main( )
        {
                student s(1, 75.00);            //Calling Parameterized Constructor
                int r_no;
                float per;
                clrscr( );
                cout<<"\n Enter Value of roll Number and percentage";
                cin>>r_no>>per;
                student s2(r_no,per);           //Passing user values to Constructor
                s.display( );
                s2.display( );
                getch( )
}
```

(d) Write a program to show use of single inheritance.

(Defining Base Class -1 Mark, Defining Derived Class -2 Marks, Creating Object and using derived members of base class -1 Mark)

Ans:

```
#include<iostream.h>
#include<conio.h>
class base
        {
                int roll;
                char name[25];
        public:
                void accept( )
                {
                        cout<<"Enter Roll Number and Name";
                        cin>>roll>>name;
                }
                void display( )
                {
                        cout<<"Roll Number is "<<roll;
                        cout<<"\nName is "<<name;
                }
        };
class derived : public base
        {
                int mark1, mark2;
        public:
                void accept_d( )
                {
                        cout<<"Enter Marks for subject 1 and Subject 2";
                        cin>>roll>>name;
                }
                void display_d( )
                {
                        cout<<"Roll Number is "<<roll;
                        cout<<"\nName is "<<name;
                }
        };
void main( )
        {
                derived d;
                clrscr( );
                d.accept( );
                d.accept_d( );
                d.display( );
                d.display_d( );
                getch( );
        }
```

(e) Write a program to find the length of string using pointer.
(Creating pointer variable -1 Mark, Setting pointer variable to first position- 1 Mark, Finding length- 2 Marks)

Ans:

```
#include<iostream.h>
#include<conio.h>
void main( )
        {
                char str[20], *p;
                int len=0;
                clrscr( );
                cout<<"\nEnter a String";
                cin>>str;
                p=&str[0];
                while(*p!='\0')
                        {
                                len++;
                                p++;
                        }
                cout<<"\nLength of a String "<<str<<" is "<<len;
                getch( );
        }
```

(f) Explain any four rules for virtual function.
(For each rule-1 Mark)

Ans:

Rules for virtual functions
When virtual functions are created for implementing late binding, we should observe some basic rules that satisfy the compiler requirements:

1. The virtual functions must be members of some class.
2. They cannot be static members.
3. They are accessed by using object pointers.
4. A virtual function can be friend of another class.
5. A virtual function in a base class must be defined, even though it may not be used
6. The prototype of the base class version of a virtual function and all the derived class versions must be identical. If two functions with the same name have different prototypes++ considers them as overloaded functions, and the virtual function mechanism is ignored.
7. We cannot have virtual constructors, but we can have virtual destructors.
8. While a vase pointer can point to any type of the derived object, the reverse is not true. That is to say, we cannot use a pointer to derived class to access an object of the base type.
9. When a base pointer points to a derived class, incrementing or decrementing it will not make it to point to the next object of the derived class. It is incremented or decremented only relative to its base type. Therefore, we should not use this method to move the pointer to the next object.
10. If a virtual function is defined in the base class, it need not be necessarily redefined in the derived class. In such cases, calls will invoke the base function.

4. Attempt any FOUR of the following: **MARKS 16**

(a) Write a program to evaluate the largest element in entered array using pointer
(Creating pointer variable- 1 Mark, Setting pointer variable to first position- 1 Mark, Finding Largest Number -2 Marks)

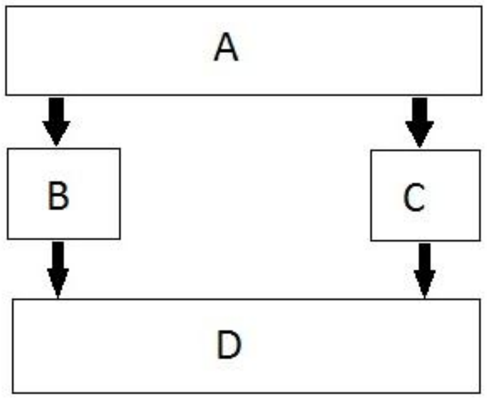
Ans:

```
#include<iostream.h>
#include<conio.h>
void main()
        {
                int arr[50],*ptr, lar=0,n,i;
                clrscr();
                cout<<"\nEnter number of elements in array";
                cin>>n;
                cout<<"\nEnter Elements in array";
                for(i=0;i<n;i++)
                        {
                                cin>>arr[i];
                        }
                ptr=&arr[0];
                for(i=0;i<n;i++)
                        {
                                if(lar<*ptr)
                                {
                                        lar = *ptr;
                                }
                                ptr++;
                        }
                cout<<"\nLargest Element in Array is "<<lar;
                getch();
        }
```

(b) Explain hybrid inheritance with example.
(Description -2 Marks, Example- 2 Marks; any other example/description can be considered; program is optional)

Ans:
Description:
"Hybrid Inheritance" is a method where one or more types of inheritance are combined together and used. Hybrid Inheritance is combination of Hierarchical and Mutilevel Inheritance. It is also known as Virtual Inheritance. We can use any combination to form hybrid inheritance only single level inheritance cannot be combined with multi-level inheritance as it will result into multilevel inheritance.

Example:-



Above figure shows graphical representation of hybrid inheritance which combines hierarchical inheritance with multiple inheritance.

Program:-

```
#include <iostream.h>
class mm
{
protected:
int rollno;
public:
void get_num(int a)
{ rollno = a; }
void put_num()
{ cout << "Roll Number Is:"<< rollno << "\n"; }
};
class marks : public mm
{
protected:
int sub1;
int sub2;
public:
void get_marks(int x,int y)
{
sub1 = x;
sub2 = y;
}
void put_marks(void)
{
cout << "Subject 1:" << sub1 << "\n";
cout << "Subject 2:" << sub2 << "\n";
}
};
class extra:public mm
{
protected:
float e;
public:
void get_extra(float s)
{e=s;}
void put_extra(void)
{ cout << "Extra Score::" << e << "\n";}
};
class res : public marks, public extra
{
protected:
float tot;
public:
void disp(void)
{
tot = sub1+sub2+e;
put_num();
put_marks();
put_extra();
cout << "Total:"<< tot;
}
};
int main()
{
res std1;
std1.get_num(10);
std1.get_marks(10,20);
std1.get_extra(33.12);
std1.disp();
return 0;
}
```

(c) Explain copy constructor with example.

(Explanation- 2 Marks, Example- 2 Marks)

Ans

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.

If a copy constructor is not defined in a class, the compiler itself defines one. If the class has pointer variables and has some dynamic memory allocations, then it is a must to have a copy constructor. The most common form of copy constructor is shown here:

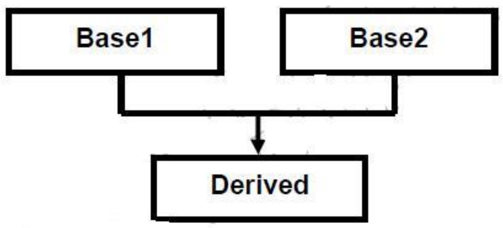
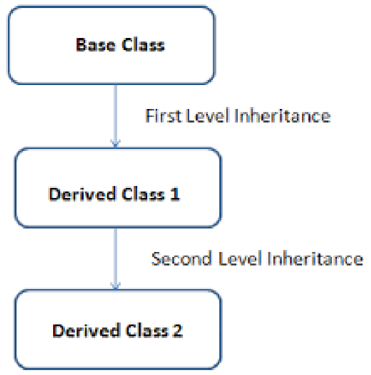
```
#include<iostream.h>
 class Point
{
private:
    int x, y;
public:
    Point(int x1, int y1) { x = x1; y = y1; }

    // Copy constructor
    Point(Point &p2) {x = p2.x; y = p2.y; }

    int getX()          {  return x; }
    int getY()          {  return y; }
};




int main()
{
    Point p1(10, 15); // Normal constructor is called here
    Point p2 = p1; // Copy constructor is called here
     // Let us access values assigned by constructors
    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
    cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();
     return 0;
}
```

(d) Differentiate between multiple inheritance and multilevel inheritance.
(Any four points of comparison each -1 Mark)

Ans:

Multiple inheritance:	**Multi-level**
Multiple inheritance refer to a class being derived from two or more classes.	Multilevel inheritance refers to a class inheriting from a parent class which is itself derived from another class.
A Class derived from at least more than one base class.	A class Extends or Derived from exactly one class & derived class can act as base class for another class
Multiple inheritances are supported by C++, but not by Java and C#.	Multilevel inheritance is supported by all OOPs languages.

Syntax in C++.
class base1
{ .... };
class base2
{ .... };
class derived : public base1, public base2
{.... ... .};
In this example, class derived is derived from two base classes base1 & base2
	Syntax in C++.
	class A
	{ .... };
	class B : public A
	{ .... };
	class C : public B
	{ .... };
	In this example, class B is derived from class A and class C is derived from derived class B.

	 Base Class First Level Inheritance Derived Class 1 Second Level Inheritance Derived Class 2

(e) Write a program to show use of passing object as a parameter to function showdata ( ) for a class "student" having data member as name and roll no. & member function as getdata ( ).
(Creating class -1 Mark, Creating Functions- 1 Mark, Passing Object as Argument- 2 Marks)
(Note:- Any concept which justifies object as function argument shall be considered)

Ans:

```
#include<iostream.h>
#include<conio.h>
class student
        {
              int roll;
              char name[25];
        public:
              void getdata( )
              {
                        cout<<"Enter Roll Number and Name";
                        cin>>roll>>name;
              }
              void showdata(student s1)
              {
                        cout<<"Roll Number is "<<s1.roll;
                        cout<<"\nName is "<<s1.name;
              }
        };

void main ()
{
student wl,w2;
wl.getdata();
w2.showdata(wl); //passing object as parameters
return 0;
}
```

(f) Explain:
(i)Scope resolution operator (ii) Memory management operator
(Scope resolution operator Explanation- 1Mark Example -1Mark)
(Memory management operator Explanation- 1Mark Example -1Mark)

Ans:
(i)Scope resolution operator

In C, the global version of a variable cannot be accessed from within the inner block. C++ resolves this problem by introducing a new operator :: called scope resolution operator. This can be used to uncover a hidden variable. It takes the following form:

```
:: variable-
```

This operator allows access to the global version of a variable.
Example:

```
int student :: roll_no;          // Using data members with the help of scope resolution operator
or
void student :: performance ( ) // Using member function with the help of scope resolution operator
{           //Function Body     }
```

(ii) Memory management operator

There are two types of memory management operators in C++:

- new
- delete

These two memory management operators are used for allocating and freeing memory block in efficient and convenient ways.

New operator:
The new operator in C++ is used for dynamic storage allocation. This operator can be used to create object of any type.

General syntax of new operator in C++:
The general syntax of new operator in C++ is as follows:

```
pointer variable = new datatype;
```

In the above statement, new is a keyword and the pointer variable is a variable of type datatype.

For example:
int *a=new int;

Delete operator:

The delete operator in C++ is used for releasing memory space when the object is no longer needed. Once a new operator is used, it is efficient to use the corresponding delete operator for release of memory.

General syntax of delete operator in C++:
The general syntax of delete operator in C++ is as follows:

delete pointer_variable;

```
#include <iostream>
using namespace std;
void main()
{
    //Allocates using new operator memory space in memory for storing a integer datatype
    int *a= new int;
    *a=100;
    cout << " The Output is:a= " << *a;
    //Memory Released using delete operator
    delete a;

}
```

5. Attempt any FOUR of the following: **MARKS 16**

(a) State any four features of object oriented programming.
(Any four each- 1Mark)

Ans:

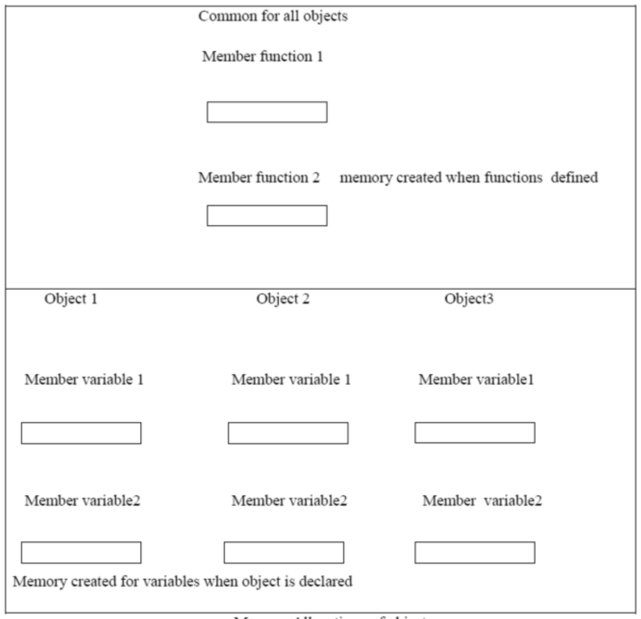
- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects
- Data structure designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together in the data structure.

- Data is hidden & cannot be accessed by external functions.
- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- Follows bottom-up approach in program designing.

(b) Explain memory allocation for object with example.
(Explanation- 2Marks, Diagram- 2Marks)

Ans:

The memory space for object is allocated when they are declared & not when the class is specified. Actually, the member functions are created & placed in memory space only once when they are defined as a part of a class definition. Since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions. When the objects are created only space for member variable is allocated separately for each object. Separate memory locations for the objects are essential because the member variables will hold different data values for different objects this is shown in fig



Memory Allocation of objects

(c) Explain various pointer arithmetic operations with examples.
(Operations-2Marks,Suitable example- 2Marks. Note: Any relevant example shall be considered)

Ans: Pointer arithmetic
C++ allows pointers to perform the following arithmetic operations:

1 A pointer can be incremented (++) or decremented (--)
2 Any integer can be added to or subtracted from a pointer.
3 One pointer can be subtracted from another.

Eg:
int a[6];
int *ptr;
ptr=&a;
ptr refers to the address of the variable a.
ptr++ or ++ptr-This statement moves the pointer to the next memory address .similarly we can decrement the pointer variable as follows:
Ptr-- or --ptr-This statement moves the pointer to the previous memory address. Also, if two pointer variables points to the same array can be subtracted from each other.

Example:
```
#include<iostream.h>
#include<conio.h>
Void main()
{
Int num[5]={56,75,22,18,90},;
Int ptr;
Int i;
Cout<<"array elements are::";
For(i=0;i<5;i++)
Ptr=num;
Cout<<"value of ptr::"<<*ptr;
Cout<<"\n";
Ptr++;
Cout<<"value  of ptr++::"<<*ptr;
Cout<<"\n";
Ptr--;
Cout<<"value  of ptr--::"<<*ptr;
Cout<<"\n";
Ptr=ptr+2;
Cout<<"value of ptr+2::"<<*ptr;
Cout<<"\n";
Ptr=ptr-1;
Cout<<"value of ptr-1::"<<*ptr;
Cout<<"\n";
Ptr+=3;
Cout<<"value of ptr+=3::"<<*ptr;
Cout<<"\n";
getch()
}
```

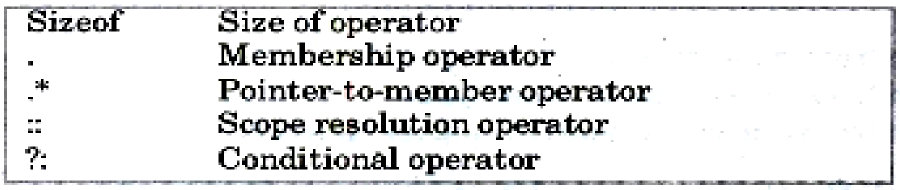
(d) Explain various rules for overloading operators.
(Any four points each- 1Mark)

Ans:

1) Only existing operators can be overloaded. New operators cannot be created.
2) The overloaded operator must have at least one operand that is of user-defined type.
3) We cannot change the basic meaning of an operator. That is to say, we cannot redefine the plus (+) operator to subtract one value from the other.
4) Overloaded operators follow the syntax rules of the original operators. They cannot be overridden.
5) There are some operators that cannot be overloaded (see table7.1)
6) We cannot use friend functions to overload certain operators. (see table 7.2) However member functions can be used to overload them.
7) Unary operators, overloaded by means of member function, take no explicit arguments and return no explicit values, but, those overloaded by means of a friend function, take one reference argument (the object of the relevant class).
8) Binary operators overloaded through a member function take one explicit argument and those which are overloaded through a friend function take two explicit arguments.

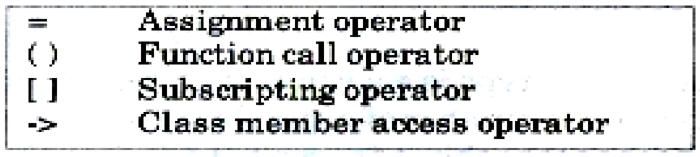
9) When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class.
10) Binary arithmetic operators such as +,-,*, and / must explicitly return a value. They must not attempt to change their own arguments.

Table 7.1 Operators that cannot be overloaded



Sizeof	Size of operator
.	Membership operator
.*	Pointer-to-member operator
::	Scope resolution operator
?:	Conditional operator

Table 7.2 Where a friend cannot be used



=	Assignment operator
( )	Function call operator
[ ]	Subscripting operator
->	Class member access operator

(e) Write a program to show use of virtual function.

(Correct program-4Marks)

Ans:

Virtual function

```
 #include<iostream.h>
#include<conio.h>
class base
{
     public:
     void display()
     {
       cout<<”Display Base”;
      }
      virtual void show()
       {
         cout<<”Show Base”;
        }
```

```
                        };
class derived: public base
{
  public:
  void display()
   {
      cout<<"Display Derived";
    }
    void show()
    {
      cout<<"Show Derived";
     }
};
void main()
{
  base b1;
  derived d1;
  base *bptr;
  bptr=&b1;
  bptr->display();
  bptr->show();
  bptr=&d1;
  bptr->display();
  bptr->show();
  getch();
}
```

(f) Define polymorphism. Explain any two types with example.

(Definition -1 Mark, Two types -3 Marks)

Ans:

Polymorphism- It is the ability to take more than one form. An operation may exhibit different behaviors in different instances.

Types- 1) Compile time polymorphism
2) Run time polymorphism

Compile time polymorphism: - It means a compiler is able to select the appropriate function for a particular function call at the compile time itself i.e. linking of function call to its definition at compile time.
It can be achieved by function overloading & operator overloading.
Function overloading- We can use the same function name to create functions that perform a variety of different tasks. i.e. same name functions may behave differently.
Operator overloading: The ability to provide the operators with special meaning for a data type is known as operator overloading.

***Runtime polymorphism:* -** It means a selecting an appropriate member function for a particular function call at the run time i.e. linking of function call to its definition at run time.
It can be achieved by implementing virtual functions in a program.
Virtual function: When base class and its derived class both contains same function name then the function in base class is declared as virtual using keyword virtual preceding its normal declaration.

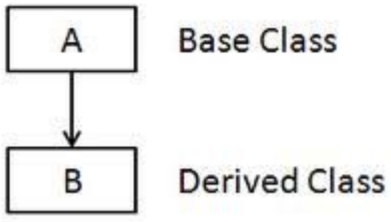
6. Attempt any TWO of the following: **MARKS 16**

(a) Explain various types of inheritance with example.

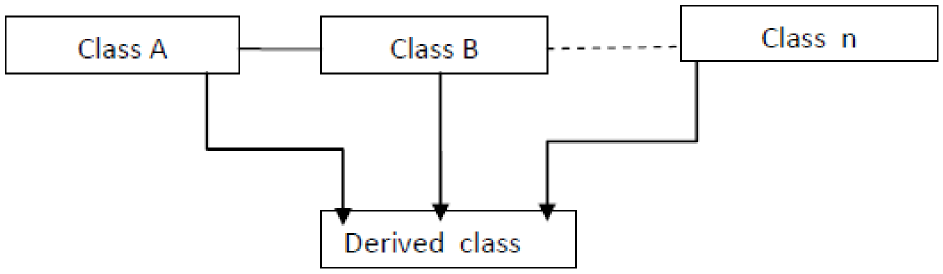
(Any four types each- 2 Marks; syntax shall be considered as example)

Ans:

1.Single inheritance: It includes single base class which can allow only one derived class to inherit its properties.

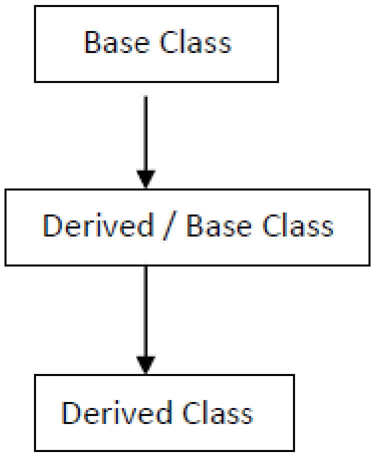


2.Multiple inheritance: In this inheritance, a single derived class can inherit properties of more than one base class

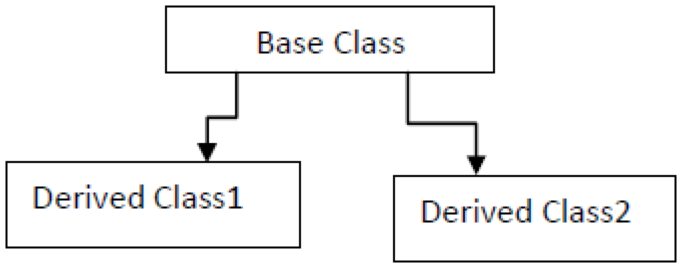


Multiple inheritance

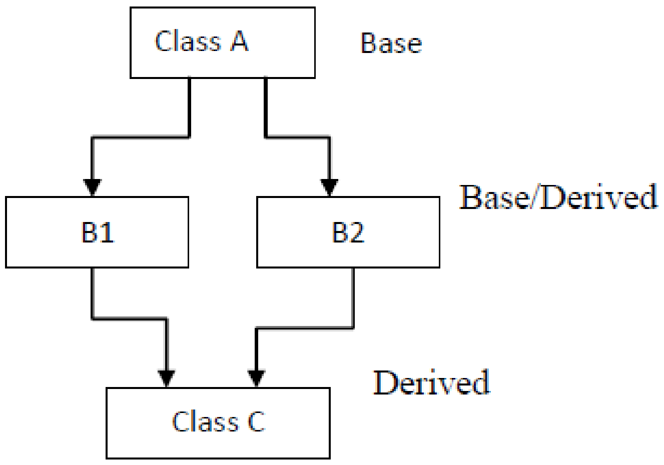
3.Multi-level inheritance: A single class can be derived from a single base class. We can derive a new class from as already derived class. It includes different levels for defining class. A child class can share properties of its base class (parent class) as well as grandparent class



4.Hierarchical inheritance: In this inheritance, multiple classes can be derived from one single base class. All derived classes inherit properties of single base class



5.Hybrid Inheritance: In this inheritance, it combines single inheritance, multiple inheritance, multi – level inheritance & hierarchical inheritance.



(b) Write a program to create a class "employee" with data member as name, designation and basic salary & gross salary. Create member functions as getdata () to read and showdata () to display details. Create sum () as friend function to calculate gross salary.

gs=bs+O.5*bs+9.O*bs;

(Class definition-2 Marks, Friend declaration-2 Marks, Friend definition-3Marks, Output -1 Mark)

Ans:

```
#include<iostream.h>
#include<conio.h>
class employee
{
private:
char emp_name[10];
char designation[7];
float bas_sal;
float gr_sal;
public:
void getdata()
{
cout<<"enter employee name,designation and basic salary";
cin>>emp_name>>endl;
cin>>designation>>endl;
cin>>bas_sal>>endl;
}
void showdata()
{
cout<<"Employee Name="<<emp_name<<endl;
cout<<"Employee designation="<<designation<<endl;
cout<<"Employee basic salary ="<<bas_sal<<endl;
}
friend void sum(employee e);
};
void sum(employee e)
{
```

```
gs_sal=e.bas_sal+(0.5*e.bas_sal)+(0.9*e.bas_sal);
cout<<"Total gross salary of employee="<<gs_sal;
}
void main()
{
employee e1;
clrscr();
e1.getdata();
e1.showdata();
sum(e1);
getch();
}
```

(c) Create a class "Account" with data member as acct.no. and balance. Create member function as getdata () and showdata (). Write a program to create a pointer for getdata () and showdata () function and access them using object of class "Account".
(Class definition- 4 Marks, Main Function- 4 Marks)

Ans:

```
#include<iostream.h>
#include<conio.h>
class account
{
private:
int acct_no;
float balance;
public:
void getdata()
{
cout<<"Enter account number and balance::";
cin>>acct_no>>endl;
cin>> balance >>endl;
}
void showdata()
{
cout<<"Account number="<<acct_no<<endl;
cout<<"balance="<< balance <<endl;
}
};
```

```
void main()
{
account a;
account *ptr;
ptr=&a;
ptr->getdata();
ptr->showdata();
getch();
}
```