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 tryto 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 thefigure. 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 constantvalues may vary and there may be some difference in the candidate's answers and modelanswer.
6) In case of some questions credit may be given by judgment on part of examiner of relevantanswer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

Q.1) A) Attempt any SIX of the following: **12M**

a. Write structure of C++ program.

(Each statement ½ M)

Ans: structure of C++ program:

Include Header Files
Class declaration
Member Functions Definitions
Main Function Program

b. How pointer is assigned to object? Explain with simple example.

(Explanation- 1M, example/program-1M)

Ans:

When address of an object of a class is stored into the pointer variable of the same class type then it is pointer to object. This pointer can be used to access the data member and member functions of same class.

Syntax: - class_name *pointer_variable;
pointer_variable=&object_name;

Example:
```
#include<conio.h>
#include<iostream.h>
class product
{
private:
```

```
int code;
float price;
public:
void getdata(void)
void display(void)
};
void main()
{
product p1;
product *ptr;
ptr=&p1;
ptr->getdata ();
ptr->display();
}
```

c. Write general form of member function, definition out of class.
(General form – 1M, definition – 1M)

Ans:

```
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 out of a class
 {
 Function body;
 }
```

d. Define constructor and destructor.
(Definition of each – 1M)

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

Destructor: A destructor is used to destroy the objects that are created by a constructor.

e. Define polymorphism. List types of polymorphism.
(Definition -1M, List – 1M)

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

f. List types of inheritance.
(List of any four types – 1/2M each)
Ans: List of inheritance:

1.Single Inheritance
2.Multiple Inheritance
3.Multilevel Inheritance
4.Hierarchical Inheritance
5.Hybrid Inheritance

g. Explain pointer operator and address operator with example.
(Explanation with example of pointer operator– 1M, address operator – 1M)
Ans:

Pointer operator:- * operator
It is used to declare pointer variable. Also used as ‘value at’ operator.
Example:-
int *p;// declares p as pointer variable
OR
printf(“%d”,*p); //Displays value at address stored in pointer p

Address operator:-&
It is used to return address of a variable. With address operator address of a variable can be stored in pointer variable.
Example:-
int *p, i ; // p is pointer variable, i is variable
p = &i; // p stores address of i

h. Write syntax of declare constructor.
(Correct syntax – 2M)
Ans:

Syntax:- constructor_name();

Note: Constructor has the same name as that of class name.

Q. 1) B) Attempt any TWO of the following: 08M
a. Explain multiple constructor in class with example.
(Explanation -2M, Any relevant example/program – 2M)
Ans:

Multiple constructor is a type of constructor in which a class can contain more than one constructor. This is known as constructor overloading. All constructors are defined with the same name as the class they belong to. All the constructors contain different number of arguments. Depending upon the number of arguments, the compiler executes appropriate constructor.
Multiple constructor can be declared in different ways:

```
integer();              // No arguments
integer(int, int);  // Two arguments
```

When the object is created the first constructor invoked.
In the first case, the constructor itself supplies the data values and no values are passed by the calling program.
In the second case, the function call passes the appropriate values from main ( ).
C++ permits us to use both these constructors in the same class.
For example, we could define a class as follows:

```
#include<iostream.h>
#include<conio.h>
class integer
{
int m, n;
public:
  integer()
  {
        m = 0;
        n = 0;
  }              // constructor 1
  integer(int a, int b)
  {
        m = a;
        n = b;
        cout<<"value of m="<<a;
        cout<<"value of n="<<b;
  }             // constructor 2
};
void main()
{
  clrscr();
  integer i1;
  integer i2(20,40);
  getch();
}
```

This declared three constructors for an integer object. The first constructor receives no arguments, the second receives two integer arguments and the third receives one integer object as an argument. *For example*, the declaration.
integer i1;
would automatically invoke the first constructor and set both m and n of i1 to zero. The statement
integer i2 (20, 40);
would call the second constructor which will initialize the data members m and n i2 to 20 and 40 respectively. Finally, the statement.

The process of sharing the same name by two or more functions is referred to as function overloading.

b. What is virtual base class? Explain with example.
(Explanation 2M, Any relevant example/program – 2M)

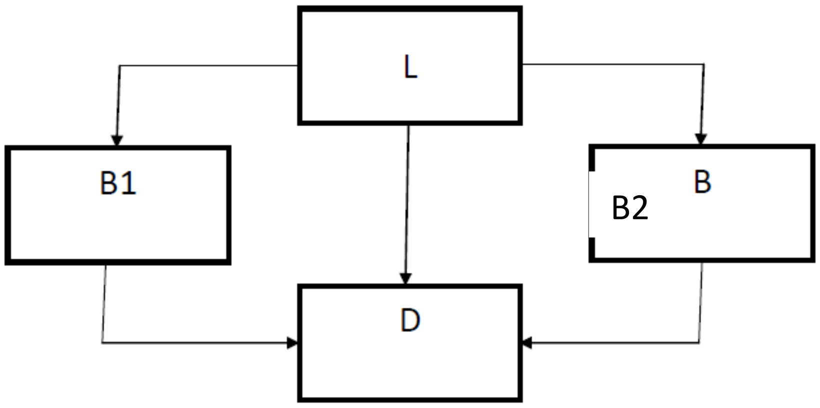
Ans:

An ambiguity can arise when several paths exist to a class from the same base class. This means that a child class could have duplicate sets of members inherited from a single base class.

C++ solves this issue by introducing a virtual base class. When a class is made virtual, necessary care is taken so that the duplication is avoided regardless of the number of paths that exist to the child class.

When two or more objects are derived from a common base class, we can prevent multiple copies of the base class being present in an object derived from those objects by declaring the base class as virtual when it is being inherited. Such a base class is known as virtual base class. This can be achieved by preceding the base class' name with the word virtual.

In the following example, an object of class D has two distinct sub objects of class L, one through class B1 and another through class B2. You can use the keyword virtual in front of the base class specifiers in the base lists of classes B1 and B2 to indicate that only one sub object of type L, shared by class B1 and class B2, exists.



Syntax of Virtual Base Class:

```
class L
{
/* ... */
};
// indirect base class
class B1 : virtual public L { /* ... */ };
class B2 : virtual public L { /* ... */ };
class D : public B1, public B2, public L { /* ... */ }; // valid
```

Using the keyword virtual in this example ensures that an object of class D inherits only one sub object of class L.

Example:

```
#include<iostream.h>
#include<conio.h>
class student
{
int rno;
public:
void getnumber()
{
cout<<"Enter Roll No:";
cin>>rno;
}
void putnumber()
{
cout<<"\n\n\t Roll No:"<<rno<<"\n";
}
};
class test:virtual public student
{
public:
int part1,part2;
void getmarks()
{
cout<<"Enter Marks\n";
cout<<"Part1:";
cin>>part1; cout<<"Part2:";
cin>>part2;
}
void putmarks()
{
cout<<"\t Marks Obtained\n";
cout<<"\n\t Part1:"<<part1;
cout<<"\n\tPart2:"<<part2;
}
};
class sports:public virtual student
{
public:
int score;
void getscore()
{
cout<<"Enter Sports Score:";
cin>>score;
}
void putscore()
{
cout<<"\n\t Sports Score is:"<<score;
}
};
class result:public test,public sports
{
int total;
public:
void display()
{
total=part1+part2+score;
putnumber();
putmarks();
putscore();
cout<<"\n\t Total Score:"<<total;
}
};
void main()
{
result obj;
clrscr();
obj.getnumber();
obj.getmarks();
obj.getscore();
obj.display();
getch();
}
```

c. List any four properties of constructor function.
(Any four properties – 1M each)

Ans:

1. It has the same name as that of the class they belongs to.
2. They should be declared in the public section.
3. They are executed when the objects are created.
4. Constructor has neither return value nor void.
5. They cannot be inherited, though a derived class can call the base class constructor.
6. Like other C++ functions, they can have default arguments and can be overloaded.
7. The main function of constructor is to initialize objects and allocate appropriate memory to objects.
8. Constructors cannot be virtual.
9. Cannot refer to their addresses.

10. An object with a constructor (or destructor) cannot be used as a member of a union.
11. They make implicit calls to the operations new and delete when memory allocation is required.
12. The constructor without argument is called as default constructor.

Q.2. Attempt any FOUR of the following. 16M

a. What is nesting of member function? Give one example.
(Explanation – 2M, Example/program – 2M)

Ans:

When a member function can be called by using its name inside another member function of the same class, it is known as nesting of member function.
A member function of a class can be called only by an object of that class using a dot operator. However, there is an exception to this. A member function can be called by using its name inside another member function of the same class. This is known as nesting of member functions.

Example:

```
# include<iostream.h>
class set
{
int m, n;
public:
void input (void);
void display (void);
int largest(void);
};
int set:: largest (void)
{
if (m >= n)
return (m);
else
return (n);
}
void set : : input (void)
{
cout<< “input values of m & n:”;
cin>> m >> n;
}
void set :: display(void)
{
cout<<“largest value = “<<largest();
}
main()
{
set a;
a.input();
a.display();
}
```

b. Describe virtual function with one example.
(Description- 2M, Example/program – 2M)
Ans:

A virtual function is a member function that is declared within a base class and redefined by a derived class. To create virtual function, precede the function's declaration in the base class with the keyword virtual. When a class containing virtual function is inherited, the derived class redefines the virtual function to suit its own needs. Base class pointer can point to derived class object. In this case, using base class pointer if we call some function which is in both classes, then base class function is invoked. But if we want to invoke derived class function using base class pointer, it can be achieved by defining the function as virtual in base class, this is how virtual functions support runtime polymorphism.

Example:

```
class A
{
  int a;
  public:
     A()
     {
       a = 1;
     }
     virtual void show()
     {
       cout<<a;
     }
};

class B: public A
{
  int b;
  public:
     B()
     {
       b = 2;
     }
     virtual void show()
     {
       cout<<b;
```

```
    }
};
int main()
{
  A *pA;
  B oB;
  pA = &oB;
  pA→show();
  return 0;
}
```

Output is 2 since pA points to object of B and show() is virtual in base class A.

c. State and explain various visibility modifiers in inheritance.
(State- 1M, explanation of each mode -1M, [Example optional])

Ans:

Visibility modes:-

1. private
2. public
3. protected

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 becomesprotected members of derived class. “Private” members of base class are not inherited in derivedclass.
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();
}
```

Base class visibility	Derived Class visibility		
	Public derivation	Private derivation	Protected Derivation
Private	Not inherited	Not inherited	Not inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

d. Differentiate between procedure oriented and object programming languages.
(Any four relevant differences – 1M each)
Ans :

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

e. Explain with example object as member function argument.
(Any one method of explanation - 2M, Example/program – 2M)
Ans.
An object may be used as function arguments in two methods:-
i) A copy of the entire object is passed to the function.
ii) Only the address of the object is transferred to the function.
1) **Pass-by-value**
Since a copy of the object is passed to the function, any changes made to the object inside the function do not affect the object used to call the function.
2) **Pass-by-reference**
When an address of the object is passed the called function works directly on the actual object used in the call. This means that any changes made to the object inside the function will reflect in the actual object.

Example:
Following program illustrates the use of object as function arguments. It performs the addition of time in the hour & minute format.
```
# include<iostream.h>
class time
{
int hours;
int minutes;
public:
void gettime(int h, int m)
{
hours = h;
minutes = m;
}
void puttime(void)
{
cout<< hours << "hours and: ";
cout<< minutes << " minutes " << "\n";
}
void sum (time, time);
};
void time :: sum (time t1, time t2)
{
minutes =t1.minutes + t2.minutes ;
hours = minutes / 60;
minutes = minutes%60;
hours = hours = t1.hours + t2.hours;
}
main()
{
     time T1, T2, T3;
     T1.gettime(2, 30);
     T2.gettime(3, 45);
     T3.sum(T1, T2);
     cout<< " T1 = ";
     T1.puttime();
     cout<< " T2 = ";
     T2.puttime();
     cout<< " T3= ";
     T3.puttime();
}
```

An object can also be passed as argument to a non-memberfunction but, such functions can have access to the public memberfunction only through the object passed as arguments to it.

f. Write a program to declare class ‘city’ with data members cityname and state. Create array of object of size 5. Read and print data for array using pointer to object.

(Correct program 4M)

Note: Any relevant logic shall be considered.

Ans:

```
#include<iostream.h>
#include<conio.h>
class city
{
        protected:
        charc_name[40];
```

```
            char state[40];
            public:
            void setName()
               {
                        cout<< "Enter city name:  ";
                        cin>>c_name;
                        cout<<"Enter state:";
                        cin>>state;
   }
    void printName()
       {
         cout<< "\n  City Name is:  " <<c_name;
         cout<< "\n State is : " << state;
       }
    };
    void main()
    {
    city* cPtr[100];    //array of pointers to cities
    int n = 0;                        //number of cities in array
    char choice;
    for(n=0;n<5;n++)
       {
         cPtr[n] = new city;                //make new object
         cPtr[n]->setName();
       }
    for(inti = 0; i< n; i++)
       {
         cout<< "\nCity number " << i+1;
         cPtr[i]->printName();
       }
    cout<<endl;
    getch();
    }
```

Q.3 .Attempt any FOUR of the following: 16M

a. List applications of Oop's technology.

(Any four applications each 1M)

Ans: Applications of OOP technology are:

1. Real-time systems
2. Simulation and modeling
3. Object-oriented databases
4. Hypertext, hypermedia and expert text

5. AI and expert systems
6. Neural networks and parallel programming
7. Decision support and office automation systems
8. CIM/CAM/CAD systems

b. State the use of default parameters in constructor with example.
(Use of default parameters in constructor 2M, Any other relevant example/program 2M)
Ans:

Default parameters in constructor
A constructor that accepts parameters in which some parameters can be declared with default value is called as constructor with default value.

Syntax:-
constrctor_name(datatype parameter1, datatype parameter2=value);

Example:-
```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class student
{
introll_no;
char name[10],course[30];
public:
student(char n[],intr,char c[]="Information Technology")
{
strcpy(name,n);
roll_no=r;
strcpy(course,c);
}
void display()
{
cout<<"\n\n Name of student : "<<name;
cout<<"\n Roll number of student : "<<roll_no;
cout<<"\n Course of student : "<<course;
}
};
void main()
{
```

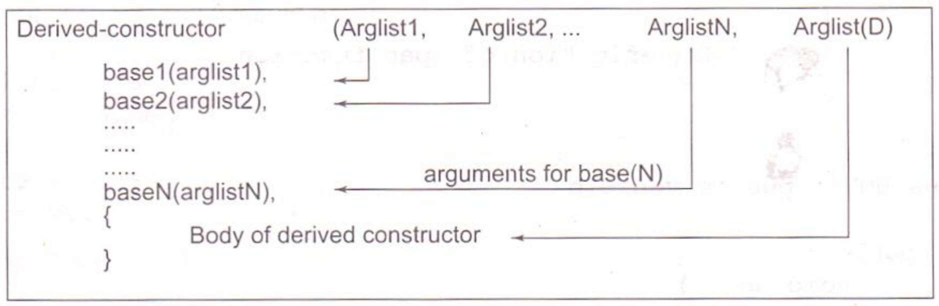
```
student s1("ABC",1);
student s2("XYZ",2,"Computer Engineering");
clrscr();
s1.display();
s2.display();
getch();
}
```

c. Explain constructors in derived class using one example.

(Explanation of constructor in derived class 2M, Any other relevant example/program 2M)

Ans: Constructor in derived class:

If a base class contains a constructor with one or more arguments then it is mandatory for the derived class to have a constructor and pass the argument to the base class constructor.


Derived-constructor (Arglist1, Arglist2, ... ArglistN, Arglist(D)
base1(arglist1),
base2(arglist2),
.....
.....
.....
baseN(arglistN),
{
Body of derived constructor
}
arguments for base(N)

A header line of derived constructor function contains two parts separated by a colon (:) First part provides the declaration of the arguments that are passed to the derived constructor and the second part lists the function calls to the base constructors.

Example:

```
#include<iostream.h>
#include<conio.h>
class base
{
int a;
public:
base(int x)
{
 a=x;
}
void displaybase()
{
```

```
cout<<a;
}
};
class derived:public base
{
int b;
public:
derived (intx,int y):base(x)
{
 b=y
}
void display()
{
cout<<b;
}
};
void main()
{
derived d(2,5);
d.displaybase();
d.display();
getch();
}
```

d. Write a program to overload operator '+' to add two complex numbers.
(Correct program 4M)
Note: Any relevant logic shall be considered.

Ans:

```
#include<iostream.h>
#include<conio.h>
class complex
{
floatx,y;
public:
complex()
 {
 }
complex(float real,floatimag)
```

```
 {
  x=real;
  y=imag;
 }
complex operator+(complex);
void display();
};
complex complex::operator+(complex c)
{
complex temp;
temp.x=x+c.x;
temp.y=y+c.y;
return(temp);
}
void complex::display()
{
cout<<x<<"+j"<<y<<"\n";
}
void main()
{
complex c1,c2,c3;
 c1=complex(2.5,3.5);
 c2=complex(1.6,2.7);
 c3=c1+c2;
cout<<"c1=";
c1.display();
cout<<"c2=";
c2.display();
cout<<"c3=";
c3.display();
getch();
}
```

e. Explain pointer arithmetic with suitable example.

(Pointer arithmetic operations- 2M (any two operations), relevant example - 2M)

Ans: Pointer 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.

Explanation:

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();
}
```

f. List characteristics of static data members.

(Any four Characteristics 1M each)

Ans:

Characteristics of static data members:

1. It is initialized to zero when the first object of its class is created. No other initialization is permitted.
2. Only one copy of that member is created for the entire class.
3. Created copy is shared by all the objects of that class, no matter how many objects are created.
4. It is visible only within the class, but its lifetime is the entire program.

Q.4.Attempt any FOUR of the following **16M**

a. Write a program to illustrate multiple inheritance. Write suitable data members and member functions.

(Any correct program with logic 4M)

Ans:

```
#include<iostream.h>
#include<conio.h>
class student
{
protected:
int roll_no,m1,m2;
public:
void get()
 {
cout<<"Enter the roll no:";
cin>>roll_no;
cout<<"\n Enter the marks of the student:";
cin>>m1>>m2;
 }
};
class sports
{
protected:
intsm;
public:
void getsm()
 {
cout<<"\n Enter the sports mark:";
cin>>sm;
 }
};
class Result:public student,public sports
{
float total,avg;
public:
void display()
 {
total=(m1+m2+sm);
avg=total/3;
cout<<"\n\n\t Roll No:"<<roll_no<<"\n\tTotal:"<<total;
cout<<"\n\t Average:"<<avg;
 }
};
void main()
{
 Result r;
clrscr();
r.get();
r.getsm();
r.display();
getch();
}
```

b. Explain overloaded constructor in class with example.
(Relevant explanation of overloaded constructor 2M, Any other relevant example/program 2M)

Ans:

Overloaded constructor is one where we can have more than one constructor in same class. Appropriate constructor will be called as per object creation.
All constructors are defined with the same name as the class they belong to. All the constructors contain different number of arguments. Depending upon the number of arguments, the compiler executes appropriate constructor.

Example:

```
#include<iostream.h>
#include<conio.h>
class demo
{
int num;
public:
demo() //Default Constructor
{
num=10;
cout<<"\n Values from Default Constructor"<<num;
}
demo(int x) //Parameterized Constructor
{
```

```
num = x;
cout<<"\n Values from Parameterized Constructor"<<num;
}
void display()
{
cout<<"\n Values of are"<<num;
}
};
void main()
{
clrscr();
demo d1,d2(30);
demo d3=d2; //Copy constructor
d3.display();
getch();
}
```

c. State scope resolution operator and memory management operator in C++.
(Scope resolution operator 2M, memory management operator (any one) 2M)

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

It takes the following form:

: :variable_name

Memory management operator

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

- new
- delete

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

d. Compare structure and class.
(Four relevant points of comparison each 1M)
Ans:

Structure	**Class**
1. Structure contains logically related data items which can be of similar type or different type.	1. Class is a way of binding data and functions together in one single unit.
2. In structure data is not hidden from external use.	2. It allows data and functions to be hidden from external use.
3. Body of structure contains declaration of variables.	3. Body of class contains declaration of variables and functions.
4. Structure does not contain function declaration.	4. Class contains function declaration.
5. Structure does not provide data hiding.	5. The basic purpose of class is to hide data from external use.

6. **Syntax:**
structstructure_name
{
datatype variable1;
datatype variable2;
..
..
}structure_variable;
	6. Syntax:
	class class_name
	{
	access specifier:
	declare data members;
	declare member functions;
	};

for e.g.
struct student
{
int roll_no;
char name[20]
}s;
	for e.g.
	class student
	{
	private:
	int roll_no;
	char name[20];
	public:
	void getdata();
	void putdata();
	};

e. What is hybrid inheritance? Give one example.
(Description -2M, any relevant example/program 2M)
Ans:

Description:

"Hybrid Inheritance" is a method where one or more types of inheritance are combined together and used. Hybrid Inheritance is combination of Single, Multiple, Hierarchical and Mutilevel 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.

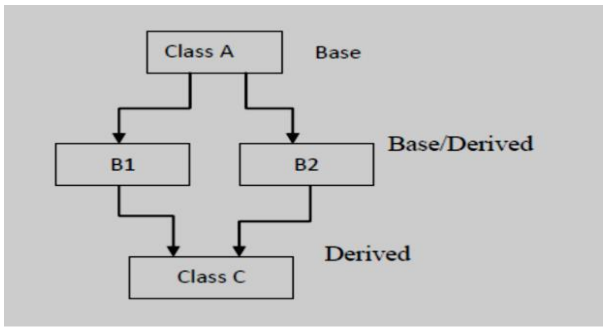
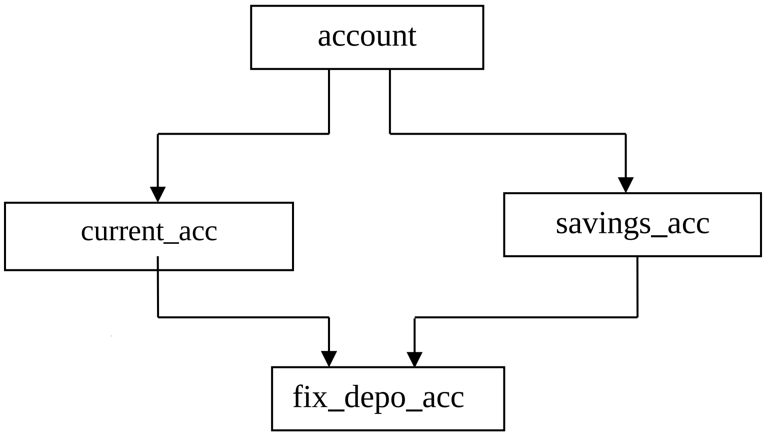


Fig: Hybrid Inheritance

Above figure shows graphical representation of hybrid inheritance.

Example:



Above diagram is an example of hybrid inheritance. Class current_acc and class savings_acc derive common members such as name and acc_no from class account. Then all the data of account is combined in class fix_depo_acc.

OR

Program as an example:

```
#include<iostream.h>
#include<conio.h>
class account
{
protected:
characc_name[10];
longph_no;
};
class savings_acc:virtual public account
{
protected:
ints_acc_no;
};
class current_acc:virtual public account
{
protected:
intc_acc_no;
};
class fix_depo_acc:public savings_acc,current_acc
{
int fix_depo_acc_no;
public:
void getdata()
{
cout<<"\n Enter account holder name : ";
cin>>acc_name;
cout<<"\n Enter phone number : ";
cin>>ph_no;
cout<<"\n Enter savings account number : ";
cin>>s_acc_no;
cout<<"\n Enter current account number : ";
cin>>c_acc_no;
cout<<"\n Enter fixed deposit account number : ";
cin>>fix_depo_acc_no;
}
void putdata()
{
cout<<"\n\n Account holder name : "<<acc_name;
cout<<"\n phone number : "<<ph_no;
cout<<"\n Savings account number : "<<s_acc_no;
cout<<"\n Current account number : "<<c_acc_no;
cout<<"\n Fixed deposit account number : "<<fix_depo_acc_no;
}
};
void main()
```

```
{
fix_depo_acc f;
clrscr();
f.getdata();
f.putdata();
getch();
}
```

f. Describe the concept of 'this' pointer. Give one example.
(Description of 'this' pointer 2M, Any relevant example/program 2M)

Ans.

this pointer: A unique keyword called *this* to represent an object that invokes a memberfunction. It is predefined pointer in any class.

```
classabc
{
int a;
…..
…..
};
```

The private variable 'a' can be directly used inside a member function,likea=123;
Using 'this' can do the same job like:
this->a=123;
this-getdata( ); // call getdata function;

Example:

```
#include<iostream.h>
#include<conio.h>
class distance
{
int feet,inches;
public:
void getdata()
{
cout<<"\n Enter distance in Feet : ";
cin>>this->feet;
cout<<"\n Enter distance in inches : ";
cin>>this->inches;
}
void operator-()
{
```

```
this->feet=-this->feet;
this->inches=-this->inches;
}
void putdata()
{
cout<<"\n\n Distance in feet : "<<this->feet;
cout<<"\n Distance in inches : "<<this->inches;
}
};
void main()
{
distance d;
clrscr();
d.getdata();
cout<<"\n Before Negation : ";
d.putdata();
-d;
cout<<"\n\n After Negation : ";
d.putdata();
getch();
 }
```

Q.5 Attempt any FOUR of the following **16M**

a. State the difference between runtime and compile time polymorphism.
(Any four relevant points- 1M each)

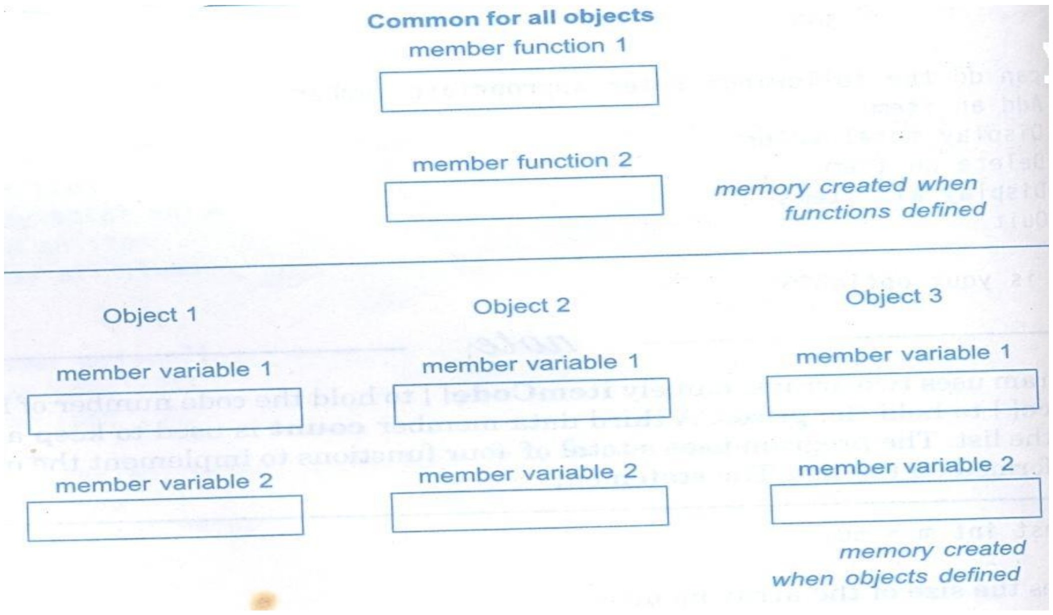
Ans:

Runtime polymorphism	**Compile time polymorphism**
In this polymorphism, selection of appropriate function is done at run time.	In this polymorphism, an object is bound to its function call at compile time.
Function to be called is unknown until appropriate selection is made.	Functions to be called are known well before.
This requires use of pointers to object	This does not require use of pointers to objects
Function calls execution are slower	Function calls execution are faster
Also called as late binding, dynamic binding.	Also called as early binding, Static binding.
It is implemented with virtual function.	It is implemented with operator overloading or function overloading

b. Explain how memory is allocated for the objects of class in C++.
(Description 2M, Diagram 2M)

Ans.

Memory space for objects is allocated when they are declared. The member functions are created and placed in the memory space only once when they are defined as a part of class specification. 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 variables 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.



c. Write a program to overload '_ _' unary operator to decrement the data members of object by one. Assume suitable data.
(Correct program 4M)
Note: Any relevant logic shall be considered.

Ans:

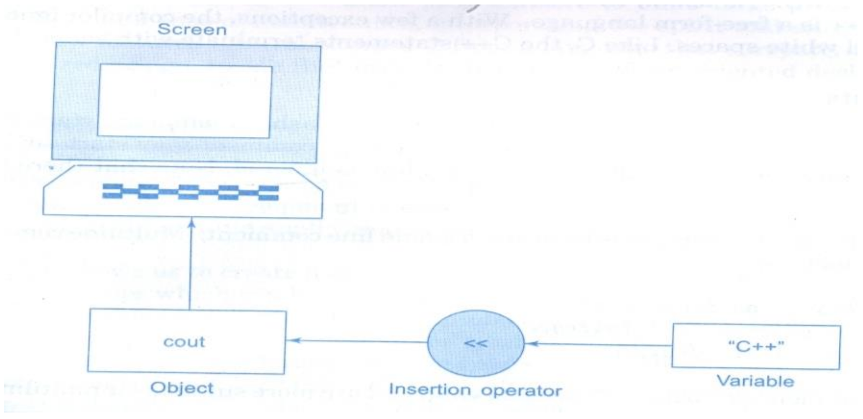
```
#include<iostream.h>
#include<conio.h>
class example
{
int a,b;
public:
void getdata()
{
cout<<"Enter the two numbers:";
cin>>a>>b;
}
void operator--()
{
a=--a;
b=--b;
}
void putdata()
{
cout<<"a="<<a<<"b"<<b;
}
}e;
void main()
{
clrscr();
e.getdata();
e--;
e.putdata();
getch();
}
```

d. Explain insertion and extraction operators.
(Explanation of insertion operators 2M, extraction operators 2M)

Ans.

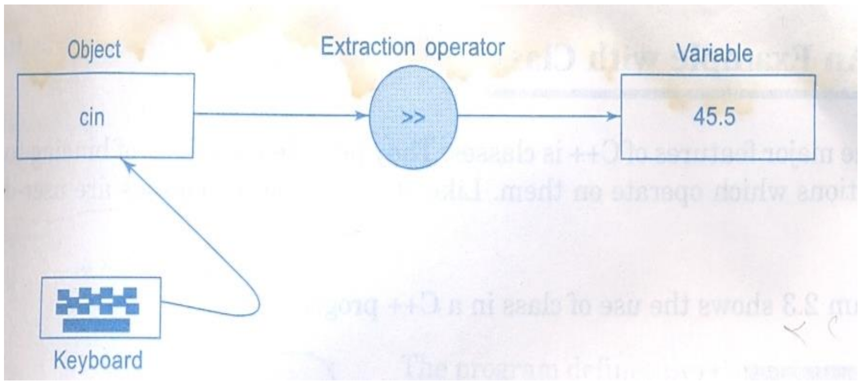
Insertion operators: The operator << is called the put to operator. It inserts (or sends) the contents of the variable on its right to the object on its left.

Example: cout<< string;



Extraction operators: The operator >> is known as extraction or get from operator. It extracts (or takes) the value from the keyboard and assigns it to the variable on its right.

Example: cin>> number1;



e. Differentiate between call by value and call by reference.
(Any four relevant points 1M each)
Ans:

Call by Value	Call by reference
Call by value method of calling function pass value of parameters as arguments.	Call by reference method of calling function pass address of parameters as arguments.
Call by value method copy of actual parameter is created.	Call by value method no copy of actual parameter is created, address of actual parameters is passed.
Processing inside function does not affect actual parameters. Function works only on copy of parameters.	Processing inside the function affects actual parameters as operations are done only on actual parameters.

Example:
swap(a,b); //function call
 	
void swap(int a,int b)// function definition
{
}
	Example:
	swap(&a,&b); //function call
	
	void swap(int *a,int *b)// function definition
	{
	}

f. Write any four rules for operator overloading.
(Any four rules; 1M each)
Ans:

Rules for overloading operators:

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 data type.
3. We can’t change the basic meaning of an operator. That is to say, we can’t redefine the plus(+) operator to subtract one value from other.
4. Overloaded operators follow the syntax rules of the original operators. They can’t be overridden.
5. There are some operators that can’t be overloaded.
6. We can’t use friend functions to overload certain operators. 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 the 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.

Q.6 Attempt any TWO of the following 16M

a. Write a program to declare class 'complex' with data members x and y. Write member function to read and print data members. Read x and y for two objects and add these two objects into third object using friend function.

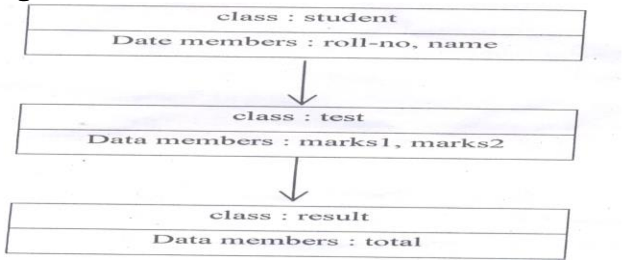
(Class definition with read and print functions 4M, friend function definition 2M, main function definition 2M)

Ans:

```
#include<iostream.h>
#include<conio.h>
class complex
{
intx,y;
public:
void read()
{
cout<<"Enter values for x and y:";
cin>>x>>y;
}
void print()
{
cout<<"x="<<x<<"y="<<y;
}
friend complex add(complex,complex);
};
complex add(complex a,complex b)
{
complex d;
d.x=a.x+b.x;
d.y=a.y+b.y;
return(d);
}
void main()
```

```
{
complex c1,c2,c;
clrscr();
c1.read();
c2.read();
c=add(c1,c2);
c.print();
getch();
}
```

b. Implement inheritance using following figure with member functions for reading and printing data.



(Correct declaration and definition of class student-2M, class test-2M, class result-2M, and main function for calling class function-2M)

Ans.

```
#include<iostream.h>
#include<conio.h>
class student
{
protected:
int roll_no;
char name[10];
public:
void getstudent()
{
cout<<"enter  roll number and name";
cin>>roll_no>>name;
}
void putstudent()
{
cout<<roll_no<<name;
}
};
class test:public student
{
```

```
protected:
int marks1,marks2;
public:
void gettest()
{
cout<<”enter marks”;
cin>>marks1>>marks2;
}
void puttest()
{
cout<<”marks1=”<<marks1<<”marks2=”<<marks2;
}
};
class result:public test
{
int total;
public:
void display()
{
total=marks1+marks2;
putstudent();
puttest();
cout<<”total=”<<total;
}
};
void main()
{
result r;
clrscr();
r.getstudent();
r.gettest();
r.display();
getch();
}
```

c. Write a program to compare two strings and concatenate two strings using pointer to string.
(String variable, pointer declaration 1M, Pointer initialization 1M, logic for comparison of two strings with pointer 3M, Logic for concatenate of two strings with pointer 3M)

Ans:

```
#include<iostream.h>
#include<conio.h>
void main()
```

```
{
char str1[10],str2[10],*ptr1,*ptr2;
int flag;
cout<<"enter two strings:";
cin>>str1>>str2;
ptr1=str1;
ptr2=str2;
while(*ptr1!='\0')
{
if(*ptr1==*ptr2)
{
ptr1++;
ptr2++;
flag=1;
}
else
{
break;
}
}
if(flag!=1)
{
while(*ptr1!='\0')
ptr1++;
ptr2=str2;
while(*ptr2!='\0')
{
*ptr1=*ptr2;
ptr1++;
ptr2++;
}
*ptr1='\0';
}
cout<<str1;
getch();
}
```