BOOP (4320702)

Important Question-Answer for GTU Exams

1) List C++ data types and derived data types.

❖ C++ offer the programmer a rich assortment of built-in as well as user defined data types. Following table lists down seven basic C++ data types:

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

❖ Several of the basic types can be modified using one or more of these type modifiers:

- signed
- unsigned
- short
- long

❖ Derived Data types

1) Array
2)Pointer
3)function
4) Reference

2) Write down an application of C++.

- 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.

3) Explain class and object.(3)

Class:

- A **class** in **C++** is a user defined type or data structure declared with keyword *class* that has data and functions as its members.
- A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces.
- Example:

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

Object :

- Object is a basic run-time entity in object oriented system.
- Object is a one type of class variable.
- Object can be declared as same as the variable declaration.
- e.g. Person,Cycle,Chair, Pen, Window in computer software
- Objects can be either

1)Physical Object :Person,Table
2) Conceptual Object: vehicle, bird
3) Abstract Object: God

- An object is represented as its property(or attributes) and operations performed on it.

Object: Window
Properties:
size
type

position

Operations:

change_size( )
move( )
minimize( )
maximize( )
close( )

4) Write the important features of Object Oriented Programming.

- ❖ Emphasis on data rather than procedure.
- ❖ Programs are divided into objects.
- ❖ Data structures are 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 and can't be accessed by external function.
- ❖ Objects may communicate with each other through functions.
- ❖ New data and functions can be easily added whenever necessary.

5) Explain Reference Variable.

- ❖ A reference variable is a one type of variable.
- ❖ It provides an alias or alternative name for a previously defined variable.
- ❖ A reference variable must be initialized at the time of declaration.
- ❖ **Syntax :**

```
data-type  &reference-name = variable-name;
```

- ❖ **Example:**

```
int a=15,b=25,total;
total = a + b;
int & sum = total;
cout<<total;          // it will print 40
cout<<sum;            // it will print 40
sum = 20;
cout<<total;          // it will print 20
cout<<sum;            // it will print 20
```

6) Explain Scope Resolution operator with example.

- ❖ Local variable: A variable which is declared inside the function or block is called as the local variable for that function and block.
- ❖ Global variable: A variable which is declared outside the function or block is called as the global variable for that function and block.
- ❖ In C, to access the global version of the variable inside the function and block that is not possible.

- In C++, the global version of variable is accessed inside the function and block using the scope resolution operator.
- Scope resolution operator is used to uncover a hidden variable.
- Syntax:

  :: variable-name;
- Example:

```
int a = 10;
void main( )
{
        int a = 15;
        cout<< " Value of a =  " << a<<endl;
        cout<< " Value of a =  " << ::a;
}
```

Output:

```
Value of a = 15
Value of a = 10
```

7) Define Object with example.(2)

- Object is a basic run-time entity in object oriented system.
- Object is a one type of class variable.
- Object can be declared as same as the variable declaration.
- **Syntax of object declaration:**

```
class-name object-name;
```

- **Example** :

```
student  s1,s2,s3;
```

where, student is the class name and s1,s2 and s3 are the object name.

- Objects can be created when a class is defined by placing their names immediately after the closing brace, same as structures.

- **Example**:

```
class student
{
        . . . . .
        . . . . .
}s1,s2,s3;
```

Where s1,s2 and s3 are the object name.

8) Explain Function Prototyping with example. (3)

- The prototype describes the function interface to the compiler by giving details such as the number and type of arguments and the type of return values.
- With function prototyping, a template is always used when declaring and defining a function.
- When a function is called, the compiler uses the template to ensure that proper arguments are passed, and return value is treated correctly.
- Function prototype is a declaration statement in the calling program.

Syntax:

```
return-type function-name(argument list);
```

- The argument list contains the types and names of arguments that must be passed to the function.

Example:

```
int sum(int x, int y);
```

- In the function prototype, all the argument variable must be declared independently inside the parenthesis.
- In the function declaration, the names of the arguments are dummy variables and therefore they are optional.

Example:

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

- The variable names in the prototype act as placeholders. So, if the names are used they don't have to match the names used in the function call or function definition.
- In the definition of the function, names are required because the arguments must be referenced inside the function.

Example:

```
int sum(int a, int b)
{
        int total=a+b;
        . . . . . .
        . . . . . .
}
```

9) Explain Call by Reference.

Call By Reference:

- The c++ provides easy and effective solution using the reference variables.
- When function is called, it creates a reference for each argument and using references the actual arguments are accessed.
- This method of passing the arguments or parameters to the function is called call by reference.
- The call by reference function is written as

```
int exch (int& a, int& b)
{
     int temp;
        Temp= a;
        a =b;
        b=temp;
 }
```

- Note that the function prototype should also mention the type of arguments as reference when arguments are to be passed as reference. The prototype of above function is as follows.
  void exch(int&,int&); //function prototype with reference arguments

  When this function is called as
  exch(x,y); //function call

- It passes the arguments as
  int & a=x; and
  int &b=y;

Which makes **a** alias of **x** and **b** alias of y and hence while function body exchanges **a** and **b**, it is same as exchanging **x** and **y**.

Example:

```
#include<iostream.h>
void exch(int&, int&);                    //function prototype
void main()
{
        int x,y;
        cout<<"Enter two numbers:"<<endl;
        cin>>x>>y;
        exch(x,y);
        cout<<"After exchange:"<<endl;
        cout<<x<<" "<<y;
}
```

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

Enter two numbers:
3  6
After exchange:
6  3
```

10) Explain inline function.

- Functions provide the flexibility to use the same code many places in a program.
- This reduces the size of a program as well as reduces memory requirements at run time.
- This advantage becomes overhead if the function needs time to call (passing arguments and transferring control) and return(returning value and control back).
- C++ goes one step ahead and provides inline function which are capable of acting as both macros as well as functions.
- The syntax for defining inline function is as follows.

```
inline return_type function_name(arguements)
{
        //body of the function
}
```

- The word inline is reserved word and preceding function header by it makes function inline.
- The word inline requests the compiler to make the function inline means expand every call to this function by its body i.e. definition.

Example:

```
#include<iostream.h>
using namespace std;
const float pie=3.14;
inline float area(float r)
{
        return (pie*r*r);
}
void main()
```

```
{
        float radius;
        cout<<"Enter radius:";
        cin>>radius;
        cout<<"Area="<<area(radius)<<endl;
}
```

11) Explain Static Member Function with example.

Static Member Functions

- The member function of the class which is declared with the static keyword, it is called as static member functions.

Static member has the following characteristics:

- A static member function can have access to only other static members declared in the same class. Static members can be either data member or member function.
- A static member function can be called using the class name as follows:

class-name :: function-name;

Example:

```
#include<iostream.h>
#include<conio.h>
class test
{
        int code;
        static int count;
        public:
                void setcode( );
                void showcode( );
                static void showcount( )
                {
                        cout<<"Count : "<<count<<endl;
                }
};
int test::count;
void main( )
{       --------
        --------
        test::showcount( );
        --------
```

```
        --------
        test::showcount( );
        ----------------
    }
```

12) Define a class Emp which include following data member and member Function. (4)
Data Member : 1) Emp_no 2) Name of Employee 3)Name of department 4) Salary
Member Function :
To read a Employee Number, Name, Department and salary.
To Display Employee Number, Name, Department and salary

```
#include<conio.h>
#include<iostream.h>
class employee
{
    private:
        int empid;
        char empname[15],dept[10];
        float salary;
    public:
        void insertdata()
        {
            cout<<"Enter Employee ID : ";
            cin>>empid;
            cout<<"Enter Employee Name : ";
            cin>>empname;
            cout<<"Enter Department Name : ";
            cin>>dept;
            cout<<"Enter Salary : ";
            cin>>salary;
        }
        void displaydata()
        {
            cout<<"Employee ID : "<<empid<<endl;
            cout<<"Employee Name : "<<empname<<endl;
            cout<<"Department Name : "<<dept<<endl;
            cout<<"Salary : "<<salary<<endl;
        }
};
void main()
{
    clrscr();
    employee e;
    e.insertdata();
    e.displaydata();
    getch();
}
```

13) Explain Function overloading with example. (7)**

- A program needs to compute the area of circle and area of rectangle.
- We can do is write two functions with names circle_area() and rect_area() and call circle_area whenever we want to compute area of circle and rect_area() whenever we want to compute area of rectangle.
- Programmer needs to remember separate names for each function and see the prototype each time whenever function call is required.
- As C does not allow same names to multiple functions , there is no way except to give separate names.
- C++ solves this problem by allowing multiple function to have same names. This is known as function overloading.
- For above example of area, we can write two functions with name area, but with different implementation.
- Complier differentiates it by number and type of the arguments. The area functions for above example is defined as

```
float area(float r);              //area of circle
float area(float l, float b);     //area of rectangle
```

Example:

```
#include<iostream.h>
Using namespace std;
Const float pie=3.14;
float area(float r)  //function to compute area of circle
{
        Return(pie*r*r);
}
float area(float 1,float b)   //function to compute area  of rectangle
{
        Return(l*b);
}
void main()
{
        float radius;
        cout<<"Enter radius:";
        cin>>radius;
        cout<<"Area of circle="<<area (radius)<<endl;
        float length, breadth;
        cout<<"Enter length and breadth:";
        cin>>length>>breadth;
        cout<<"Area of rectangle="<<area(length, breadth)<<endl;
}
```

Output:

Enter radius:3.4
Area of circle:36.298404
Area of rectangle:16.320002

14) Explain Default Argument with example.

- Function defined with three arguments can not be called with one or two arguments in C, but should be called with always three arguments.
- C++ allows calling of function with less number of arguments then listed in its header.
- C++ makes it possible by allowing default arguments i.e. arguments having default values.
- Whenever function is called and value for such argument is not passed, compiler uses the default value.

Example:

```
#include<iostream>
Using namespace std;
void set_point(int x,int y=0);  //default arguments
void main()
{
        int p,q;
        cout<<"Enter x coordinate:";
        cin>>p;
        set_point(p);
        cout<<"Enter x and y coordinates:"
        cin>>p>>q;
        set_point(p,q);
}
        //function to set the point
void set_point(int x,int y)
{
        Cout<<"("<<x<<","<<y<<")"<<endl;
}
```

Output:

Enter x coordinate:5
(5,0)
Enter x and y coordinates:6 8
(6,8)

15) Explain Parameterized Constructor with suitable example. (4)***

- The constructor which can take the arguments that is called as parameterized constructors.
- It follows all properties of the constructor.
- It takes parameters to initialize the data.
- The parameterized constructor is called by two ways:
  - **Implicit calls**
  - **Explicit calls**

Example of implicit call is

point p2(5,7);

Where, p2 is the object name of the class point.

Example of explicit call is

point p1 = point(5,7);

Syntax:

```
class class-name
{
   private :
      data Members;
   public:
       class-name(variables)
      {
      // Constructor code
      }
   //... other Variables & Functions
};
```

Example:

```
class point
{
        private:
                int x;
                int y;
        public:
                point( )            //default constructor
                {
                        x=0;
                        y=0;
                }
                point(int x1,int y1)            //parameterized constructor
                {
```

```
                x=x1;
                y=y1;
        }
        void putpoint( )
        {
        cout<<" ( "<<x<<" , "<<y<<" ) "<<endl;
        }
};

void main( )
{
    point p1;
    point p2(5,7);
    p1.putpoint( );
    p2.putpoint( );
}
```

Output:
(0,0)
(5,7)

16) Explain Copy Constructor with example.

- 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.
- It is used to initialize one object from another of the same type.
- It Copy an object to pass it as an argument to a function.
- It Copy an object to return it from a function.

Example:

```
class point
{
    private:
        int x;
        int y;
    public:
        point( )            //default constructor
        {
            x=0;
            y=0;
        }
        point(int x1,int y1)        //parameterized constructor
        {
            x=x1;
            y=y1;
        }
        point(point& p)             //copy constructor
```

```
                {
                        x=p.x;
                        y=p.y;
                }

                void putpoint( )
                {
                cout<<" ( "<<x<<" , "<<y<<" ) "<<endl;
                }
};

void main( )
{
        point p1;
        cout<<"p1= ";
        p1.putpoint( );

        point p2(5,7);
        cout<<"p2= ";
        p2.putpoint( );

        point p3(p2);   //Using copy constructor   // implicit call
        cout<<"p3= ";
        p3.putpoint( );

        point  p4=p3;  //Using copy constructor          //explicit call
        cout<<"p4= ";
        p4.putpoint( );

        point p5;
        p5=p4;
        cout<<"p5= ";
        p5.putpoint( );
}
```

Output:

```
p1= (0,0)
p2=(5,7)
p3=(5,7)
p4=(5,7)
p5=(5,7)
```

17) What is friend function with example.(4)***

- To make an outside function friendly to a class. We have to declare this function as a friend of the class.
- The function declaration should be preceded by the keyword friend.

- The functions that are declared with the keyword friend are known as friend function.
- The function is defined elsewhere in the program like a normal C++ function.

➢ **Characteristics of the Friend Function:**

- It is not in the class to which it has been declared as friend.
- Since it is not in the scope of the class, it cannot be called using the object of that class.
- It can be invoked like a normal function without the help of the object.
- It cannot access the member names directly and has to use an object name and membership operator with each member name.
- It can be declared either in the public or private part of a class.
- It has the objects as arguments.

Example:

```
#include<iostream.h>
#include<conio.h>
class ABC;      //Forward declaration of class
class XYZ
{
        int a;
public:

        void setvalue(int x)
        {
                a=x;
        }
        friend void max(ABC,XYZ);
};
class ABC
{
        int b;
        public:
        void setvalue(int y)
        {
                b=y;
        }
        friend void max(ABC,XYZ);
};
void max(ABC p,XYZ q)
{
        if(p.b>q.a)
        {
```

```
                cout<<"b is max";
        }
        else
        {
                cout<<"a is max";
        }
}
void main()
{

        ABC p;
        XYZ q;
        clrscr();
        p.setvalue(20);
        q.setvalue(10);
        max(p,q);
        getch();
}
```

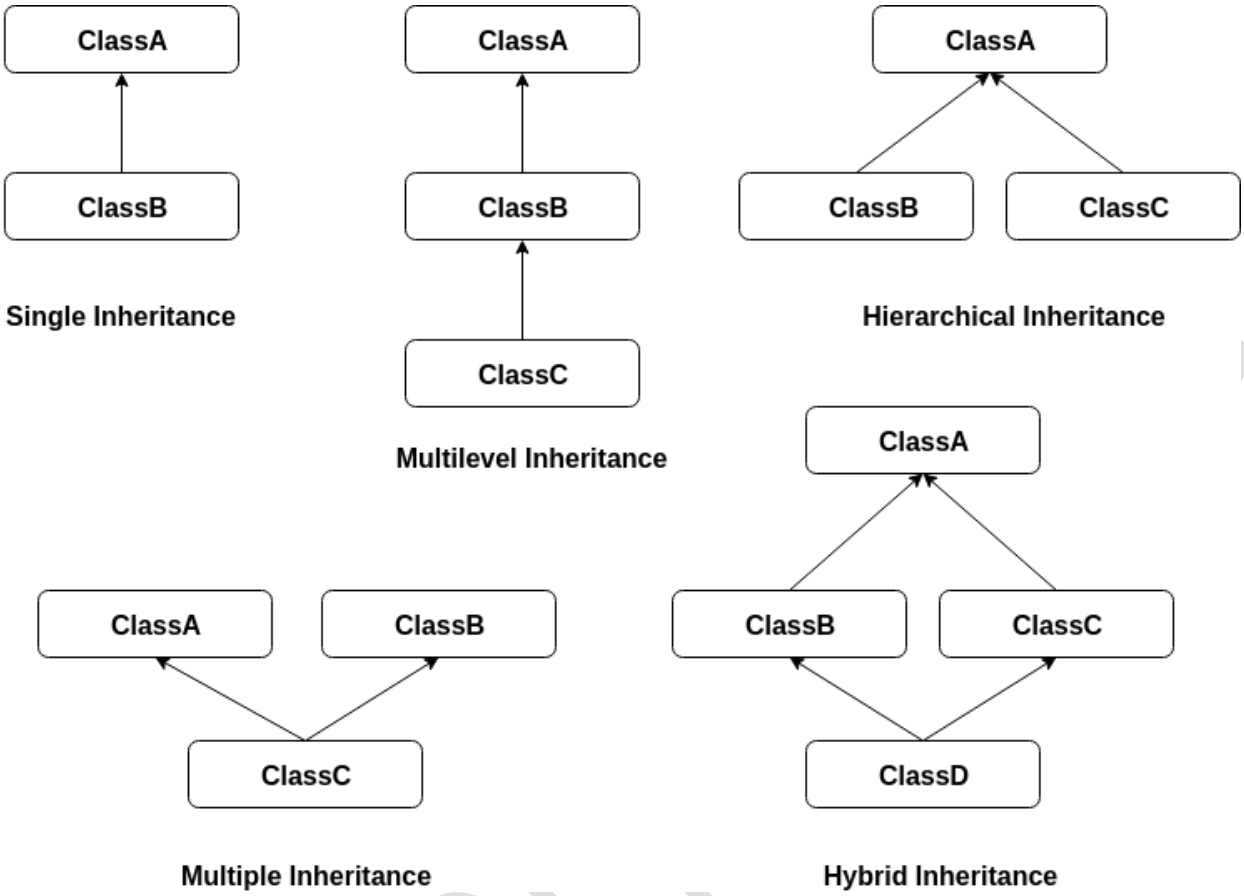
Output of the Program:

b is max

18) Define Inheritance ? List the types of Inheritance and also draw the diagram s of it.(7)*

- Inheritance is the process the by which object of one class derived (acquired) the properties of object of another class.
- **Types of inheritance:**

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



19) Write a simple C++ program that show the use of single Inheritance(4)

- In single level inheritance there is only one base class and one derived class.
- Figure of single level inheritance given below:

Single Inheritance

- In above figure, class A is called as Base Class (Old Class) and class B is called as Derived class (New Class).
- The derived class B derived all or some properties (members) of the base class A.
- The derived class B has its own properties as well as derived properties of base class A.
- Using the object of derived class B we can call the members of Derived class as well as members of base class A which are derived inside the derived class.
- Example:

```
class A
{
        - - - - -
        - - - - -
        - - - - -
};
 class B : public A
{
        - - - - -
        - - - - -
        - - - - -
};
```

Program:

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

class circle
{
        protected:
                float a;
                int r;
        public:
        void getdata()
        {
                cout<<"Enter r=";
                cin>>r;
        }
        void area()
        {
                a=3.14*r*r;
                cout<<"Area="<<a;
        }
};
```

```
class circle_ext:public circle
{
        protected:
                float p;
        public:
                void peri()
                {
                        p=2*3.14*r;
                        cout<<"Peripheral="<<p;
                }
};
void main()
{
        clrscr();
        circle c1;      //base class
        c1.getdata();
        c1.area();

        circle_ext c2;  //derived class
        c2.getdata();
        c2.area();
        c2.peri();
        getch();
}
```

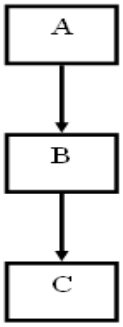
Output:

```
Enter r=2
Area=12.56

Enter r=3
Area=28.26
Peripheral=18.84
```

20) What is Inheritance? Explain Multilevel inheritance with example (4)*

- When a class is derived from another derived class i.e., derived class act as a base class, such type of inheritance is known as multilevel inheritance.
- In multilevel inheritance, the class is derived from another derived class.
- Figure of multilevel inheritance given below:



Multilevel Inheritance

- In above figure, class A is called as Base Class (Old Class) for class B. And class B is called as Derived class (New Class) for class A.
- Also class B is called as base class for class C. And class C is called as derived class for class B.
- The class B is act as base class as well as derived class. So, class B is called as intermediate class.
- Here, class B has its own properties (members) as well as derived properties (members) of class A. And class C has its own properties (members) as well as derived properties (members) of Class B.
- Using the object of derived class B we can call the members of derived class B as well as members of base class A which are derived inside the derived class
- Using the object of derived class C we can call the members of derived class C as well as members of base class A and B which are derived inside the derived class.

Example of multilevel inheritance.

```
class A
{
        - - - - -
        - - - - -
        - - - - -
};
 class B : public A
{
        - - - - -
        - - - - -
        - - - - -
};
class C : public B
{
        - - - - -
        - - - - -
        - - - - -
};
```

Program:

```
#include<conio.h>
#include<iostream.h>
class base
{
        protected:
                int a;
        public:
                base()
                {
                        a=0;
                }
                base(int a1)
                {
                        a=a1;
                }

                void display( )
                {
                        cout<<"A="<<a;
                }
};
class derived1:public base
{
        protected:
                int b;
        public:
                derived1()
                {
                        b=0;
                }
                derived1(int a1,int b1):base(a1)
                {
                        b=b1;
                }
                void display()
                {
                        cout<<"A="<<a<<"B="<<b<<endl;
                }
};
class derived2:public derived1
{
        protected:
                int c;
        public:
                derived2()
```

```
                {
                        c=0;
                }
                derived2(int a1,int b1,int c1):derived1(a1,b1)
                {
                        c=c1;
                }
                void display()
                {
                        cout<<"A="<<a<<"B="<<b<<"C="<<c;
                }
};
void main()
{
        clrscr();
        base a1(10);
        a1.display();
        derived1 b1(20,30);
        b1.display();
        derived2 c1(40,50,60);
        c1.display();
        getch();
}
```

Output:

```
A=10
A=20    B=30
A=40    B=50   C=60
```

21) Explain 'this' pointer with example.(4)**

- this pointer is used to represent an object that invokes a member function.
- Example: the function call A.max( ) will set the pointer this to the address of the object A.
- The pointer this acts as an implicit argument to all the member function.
- **Example:**

```
class ABC
{
        int a;
        . . . . . .
        . . . . . .
};
```

- The private variable can be directly inside a member function. Like, a = 150;
- It can be done using the this pointer like, this->a = 150;
- The application of the pointer this is to return the object it points to.

```
return *this;
```

Example:

```
#include<iostream.h>
#include<conio.h>
Class person
{
        private:
                char name[10];
                int age;
public:
    void getdata( )
    {
                        cout<<"Enter  Name and Age :"
                        cin>>Name;
                        cin>>age;
    }
    void putdata( )
    {
                cout<<"Name:"<<name;
                cout<<"Age:"<<age;
    }
    person &  elder (person  &p)
    {
                if (p.age > age)
                                return  p;
                else
                                return  *this;
            }
};
void main( )
{
    person  p1,p2;

    p1.getdata( );
    p2.getdata( );
    p1.putdata( );
    p2.putdata( );

    person& p3=p1.elder(p2);
    cout<<" Elder is : ";
    p3.putdata( );
    getch( );
}
```

Output:

```
Enter  Name and Age : Sunita   20
Name: Sunita
```

```
Age: 20
Enter  Name and Age : Anita    18
Name: Anita
Age: 18
Elder is :
Name: Sunita
Age: 20
```

22) Explain Virtual Function with suitable example. (4)*

- ❖ When we use the same function name in both the base and derived classes, the function in base class is declared as virtual using the keyword virtual preceding its normal declaration.
- ❖ When a function is made virtual, C++ determines which function to use at run time based on the type of object pointed to by the base pointer, rather than the type of the pointer.
- ❖ So, by making the base pointer to point to different objects, we can execute different versions of the virtual function.

Rules for virtual function

1. The virtual function 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 a friend of another class.
5. A virtual function in a base class must be defined, even though it may not be used.
6. The prototypes 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, C++ considers them as overloaded functions.
7. We cannot have virtual constructors, but we can have virtual destructors.
8. While a base pointer can point to any type of the derived object, the reverse is not true.
9. 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.

Example:

```
#include<iostream.h>
#include<conio.h>
class base
{
        protected:
                int b;
        public:
                base(int b1)
                {
                        b=b1;
                }
```

```
                virtual void display ( )
                {
                        cout<<"b= "<<b;
                }
};

class derived: public base
{
        protected:
                int d;
        public:
                derived(int b1, int d1):base(b1)
                {
                        d=d1;
                }
                void display ( )
                {
                        cout<<"b= "<<b;
                        cout<<"d= "<<d;
                }

};

void main()
{
        base  *b;
        base b(5);
        b=&b1;
        b->display( );

        derived d1(10,20);
        derived d2(30,40);

        b=&d1;          //Display data of derived class object d1
        b->display( );
        b=&d2;          //Display data of derived class object d2
        b->display( );
}
```

Output:

```
b=5
b=10   d=20
b=30   d=40
```