Constructor and Destructor

1 What is Constructor?
It is a member function which is automatically used to initialize the objects of the class type with legal initial values.

2 Why you should define a Constructor ?
Uninitialized member fields have *garbage* in them. This creates the possibility of a serious bug (eg, an uninitialized pointer, illegal values, inconsistent values, ...).

3 How can we declaring a constructor.
A constructor is similar to a function, but with the following differences.

- No return type.
- No return statement.

Example:

```
class Point {
    public:
        Point(); // parameterless default constructor
        Point(int new_x, int new_y); // constructor with parameters
        int getX();
        int getY();
    private:
        int x;
        int y;
};
```

4 Define any five characteristics of Constructor.

- They are called automatically when the objects are created.
- All objects of the class having a constructor are initialize before some use.
- The address of a constructor cannot be taken.
- These cannot be static.
- Return type cannot be specified for constructors.

5 What are the types of Constructor (Write any Four.)?
- Default constructors
- Parameterized Constructor

- Overloaded Constructor
- Copy Constructor

6 What is Default Constructor?
A *default constructor* is a constructor that either has no parameters, or if it has parameters, *all* the parameters have default values.

7 What is Parameterized Constructor?
A Constructor with arguments is called a parameterized constructor.

8 What is Overloaded Constructor?
Like function Overloading Constructor Overloading can also be done.

9 What is Copy Constructor? Explain with example.
Copy constructor is

- a constructor function with the same name as the class
- used to make deep copy of objects.

For ex:

```
class A //Without copy constructor
{
    private:
    int x;
    public:
    A() {A = 10;}
    ~A() {}
}
```

```
class B //With copy constructor
{
    private:
    char *name;
    public:
    B()
    {
    name = new char[20];
    }
    ~B()
    {
    delete name[];
    }
//Copy constructor
    B(const B &b)
    {
    name = new char[20];
    strcpy(name, b.name);
```

```
      }
   };
```

Let us Imagine if you don't have a copy constructor for the class B. At the first place, if an object is created from some existing object, we cannot be sure that the memory is allocated. Also, if the memory is deleted in destructor, the delete operator might be called twice for the same memory location.

This is a major risk. One happy thing is, if the class is not so complex this will come to the fore during development itself. But if the class is very complicated, then these kind of errors will be difficult to track.

10 Explain any three important places where a copy constructor is called.

- When an object is created from another object of the same type
- When an object is passed by value as a parameter to a function
- When an object is returned from a function

11 What is Dynamic Initialization of objects?

If we initialized class objects at run time, it is the case of Dynamic Initialization.

12 Define Destructors. With Syntax.

Destructors are less complicated than constructors. You don't call them explicitly (they are called automatically for you), and there's only one destructor for each object. The name of the destructor is the name of the class, preceeded by a tilde (~). Here's an example of a destructor:

```
Player::~Player() {
  strength = 0;
  agility = 0;
  health = 0;
}
```

13 What are the characteristics of Destructors? Any Five

- These are called automatically when the objects are destroyed.
- Destructor functions follow the usual access rules as other member functions.
- No arguments and return type permitted with destructors.
- These cannot be inherited.
- Address of a destructor cannot be taken.

Answer the questions (i) and (II) after going through the following class:

```
Class Maths
{
Char chapter[20];
int marks[20];
public:
Maths()                                    // Function 1
{
Strcpy(chapter, “Cons”);
Marks=90;
cout<<”Chapter Intialised”;
}
~Maths()                                    // Function 2
{
cout<<”Chapter Over”;
}
};
```

(i) Name the specific features of class shown by member function 1 and member function 2 in above example.

- Function 1: Constructor or Default Constructor
- Function 2: Destructor

(ii) How would member function 1 and function 2 get executed?

- When an object of class Maths is created Function 1 is executed.
- Function 2 is invoked automatically when the scope of an object of class Maths comes to an end.

INHERITANCE

1 What is Inheritance? Explain.

Inheritance is the process by which new classes called *derived* classes are created from existing classes called *base* classes. The derived classes have all the features of the base class and the programmer can choose to add new features specific to the newly created derived class.

For example, a programmer can create a *base* class named fruit and define *derived* classes as mango, orange, banana, etc. Each of these derived classes, (mango, orange, banana, etc.) has all the features of the *base* class (fruit) with additional attributes or features specific to these newly created derived classes. Mango would have its own defined features, orange would have its own defined features, banana would have its own defined features, etc.

This concept of *Inheritance* leads to the concept of *polymorphism*.

2 What are the features or Advantages of Inheritance:

Reusability:

Inheritance helps the code to be reused in many situations. The base class is defined and once it is compiled, it need not be reworked. Using the concept of inheritance, the programmer can create as many derived classes from the base class as needed while adding specific features to each derived class as needed.

Saves Time and Effort:

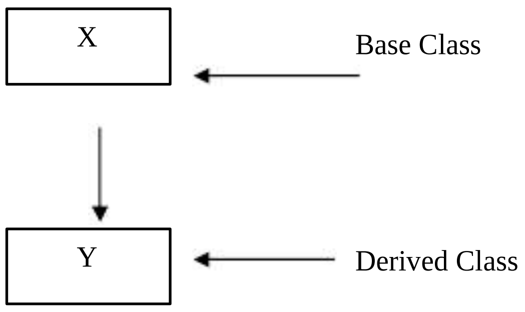
The above concept of reusability achieved by inheritance saves the programmer time and effort. Since the main code written can be reused in various situations as needed.
Increases Program Structure which results in greater reliability.

3 Write general form for implementing the concept of Inheritance:
class derived_classname: access specifier baseclassname
For example, if the *base* class is *exforsys* and the derived class is sample it is specified as:

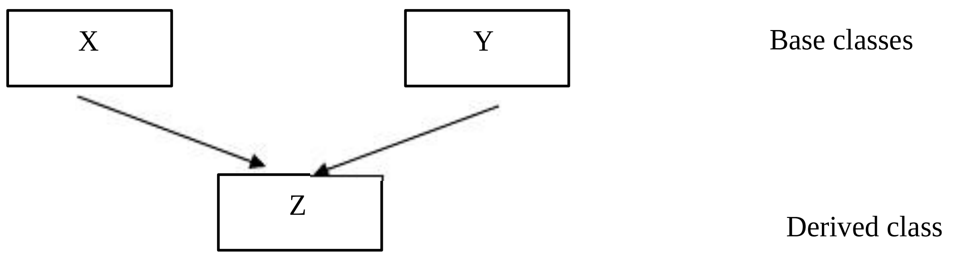
```
        class sample: public exforsys
```

The above makes sample have access to both *public* and *protected* variables of base class *exforsys*. Reminder about public, private and protected access specifiers:
If a member or variables defined in a class is private, then they are accessible by members of the same class only and cannot be accessed from outside the class.
.
Public members and variables are accessible from outside the class.
.
Protected access specifier is a stage between private and public. If a member functions or variables defined in a class are protected, then they cannot be accessed from outside the class but can be accessed from the derived class.

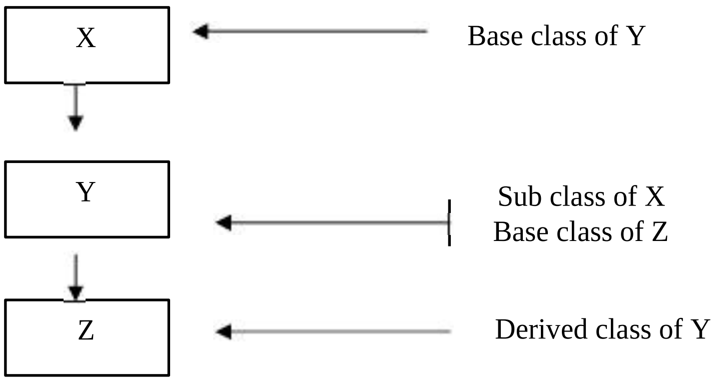
4 What are the types of Inheritance? Explain Each.
Single inheritance
When a derived class inherits only from one base class, it is known as single inheritance.



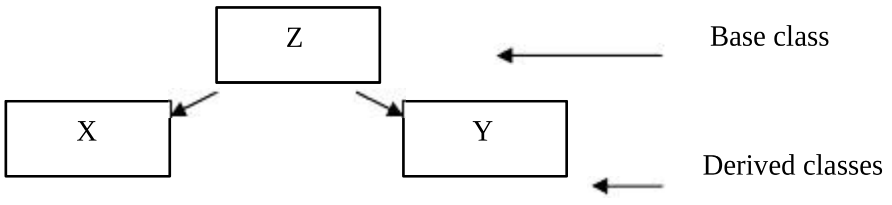
Multiple inheritance
When the derived class inherits from multiple base classes, it is known as Multiple inheritance.



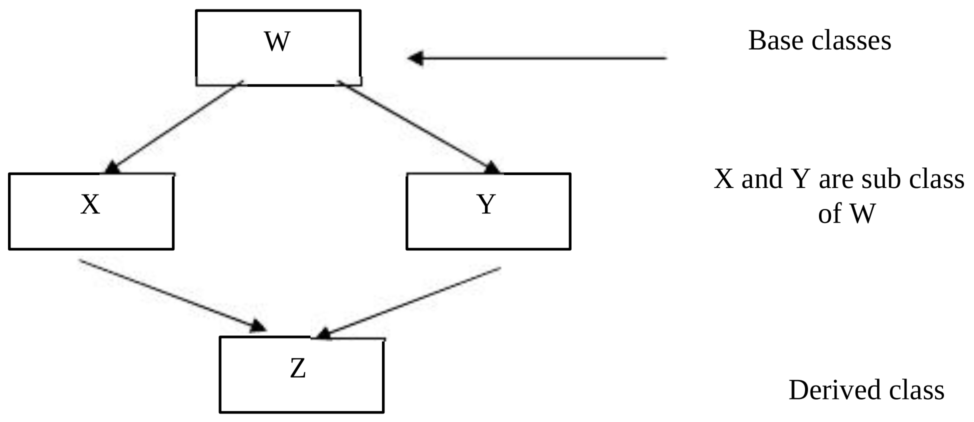
Multilevel inheritance
When a derived class inherits from a class that itself inherits from another class, it known as multilevel inheritance.



Hierarchical inheritance
When the properties of one class may be inherited by more than one class, it is known as Hierarchical inheritance



Hybrid inheritance
When a derived class inherits from multiple base classes and all of its base classes inherits from a single base class, this form of inheritance is known as hybrid inheritance.



5 Differentiate between private and protected members.
The Protected and private members are hidden from outside world of a class ie both cannot be accessed by non member and non friend functions. However, the difference is that private members are not inherited by derived class whereas protected members are inherited by the derived class.

Write the output of given example of Inheritance Example:

```
class exforsys
{public:
exforsys(void) { x=0; }
void f(int n1)
{x= n1*5;
}
void output(void) { cout<<x; }
private:
int x; };
class sample: public exforsys
{ public:
sample(void) { s1=0; }
void f1(int n1)
{s1=n1*10;}
void output(void)
{exforsys::output();
cout << s1; }
private:
int s1;};
int main(void)
{ sample s;
s.f(10);
s.output();
s.f1(20);
s.output();
}
```

The output of the above program is

```
50
200
```

Explanation

In the above example, the derived class is sample and the base class is *exforsys*. The *derived* class defined above has access to all *public* and *private* variables. *Derived* classes cannot have access to base class *constructors* and *destructors*. The derived class would be able to add new member functions, or variables, or new constructors or new destructors. In the above example, the derived class sample has new member function f1( ) added in it. The line:

```
sample s;
```

creates a derived class object named as s. When this is created, space is allocated for the data members inherited from the base class *exforsys* and space is additionally allocated for the data members defined in the derived class *sample*.

The *base* class constructor *exforsys* is used to initialize the base class data members and the *derived* class *constructor* sample is used to initialize the data members defined in *derived* class.

6 What is Visibility Mode? Write their types too.

These control the access specifier to be inheritable members of the base class, in he derived class.

The types of Visibility modes are

- The Public
- The Protected
- The Private

7 Explain the role of each Visibility Mode?

- The Public

  In this mode, the derived class can access the public and protected members of the base class publically.

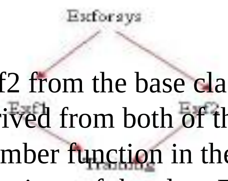
- The Protected

  In this mode, the derived class can access the public and protected members of the base class protectedly

- The Private

  In this mode, the derived class can access the public and protected members of the base class privately.

8 Explain Virtual Base Class.
Virtual Base Classes



In the above example, there are two derived classes Exf1 and Exf2 from the base class Exforsys. As shown in the above diagram, the Training class is derived from both of the derived classes Exf1 and Exf2. In this scenario, if a user has a member function in the class Training where the user wants to access the data or member functions of the class Exforsys it would result in error if it is performed like this:

```
class Exforsys
{
protected:
int x;
};

class Exf1:public Exforsys
{ };

class Exf2:public Exforsys
{ };

class Training:public Exf1,public Exf2
{
public:
int example()
{
return x;
}
};
```

The above program results in a compile time error as the member function example() of class Training tries to access member data x of class Exforsys. This results in an error because the derived classes Exf1 and Exf2 (derived from base class Exforsys) create copies of Exforsys called subobjects.

This means that each of the subobjects have Exforsys member data and member functions and each have one copy of member data x. When the member function of the class Training tries to access member data x, confusion arises as to which of the two copies it must access since it derived from both derived classes, resulting in a compile time error.

When this occurs, Virtual base class is used. Both of the derived classes Exf1 and Exf2 are created as virtual base classes, meaning they should share a common subobject in their base class.

For Example:

```
class Exforsys
{ protected:
int x;
;
class Exf1:virtual public Exforsys
{ };
class Exf2:virtual public Exforsys
{ };
class Training:public Exf1,public Exf2
{
public:
int example()
{ return x; } };
```

Because a class can be an indirect base class to a derived class more than once, C++ provides a way to optimize the way such base classes work. Virtual base classes offer a way to save space and avoid ambiguities in class hierarchies that use multiple inheritance.

14 **Consider the following declarations and answer the questions given below:**

```
class LB
{ char name[20];
protected:
int jaws;
public:
void inputdata(char, int);
void outputdata();
};
class animal:protected LB
{
int tail;
protected:
int legs;
public:
void readdata(int,int);
void writedata();
};
class cow:private animal
{
int horn_size;
public:
void fetchdata(int);
void displaydata();
};
```

(i) **Name the base class and derived class of class animal.**

(ii) **Name the data member that can be accessed from function displaydata().**

(iii) **Name the data members that can be accessed by an object of cow class.**

(iv) Is the member function outputdata() accessible to the object of animal class.

Ans i Base class of animal is LB, Derived class of animal is cow.

Ans ii The data members which can be accessed from function displaydata() are horn_size, legs and jaws.

Ans iii The object of cow class cannot be directly access any data member.

Ans iv No