What is a class? How can you define a class in C++? Explain with an example.

Object Oriented Programming encapsulates data (attributes) and functions (behavior) into packages called classes.

The class combines data and methods for manipulating that data into one package. An object is said to be an instance of a class. A class is a way to bind the data and its associated functions together. It allows the data to be hidden from externa l use. When defining a class, we are creating a new *abstract data type* that can be treated like any other built-in data type.

Generally, a class specification has two parts:

1. Class declaration
2. Class function definitions

The **class declaration** describes the type and scope of its members. The class function definitions describe how the class functions are implemented. The general form of a class declaration is:

```
class class_name
{
        private:
                variable declarations;
                function declarations;
        public:
                variable declarations;
                function declarations;
};
```

- ✓ The class declaration is similar to structure declaration in C. The keyword **class** is used to declare a class. The body of a class is enclosed within braces and terminated by a semicolon.

- ✓ The class body contains the declaration of variables and functions. These functions and variables are collectively called members. They are usually grouped under two sections i,e. private and public to denote which members are private and which are public. These keywords are known as **visibility labels**.

- ✓ The members that have been declared as private can be accessed only from within the class. On the other hand, public members can be accessed from outside the class also.

- ✓ The data hiding is the key feature of OOP. By default, the members are **private.** The variables declared inside the class are known as data members and the functions are known as *member functions*.

- ✓ Only the member functions can have access to the private members and functions. However, the public members can be accessed from outside the class.

- ✓ The binding of data and functions together into a single class type variable is referred to as *encapsulation.*

A Simple Class Example: **A typical class declaration would look like:**

```
class item
{
        int no;                 // variables declaration
        float cost;     // private by default
```

```
    public:
        void getdata(int a, float b);        // functions declaration
        void putdata(void);                  // using prototype
};
```

How can you declare objects to a class? Explain with an example.

Once a class has been declared, we can create variables of that type by using the class name. For example,

item x; // memory for x is created

Creates a variable x of type *item*.

In C++, the class variables are known as *objects.* Therefore, x is called an object of type item. We may also declare more than one object in one statement. Example:

item x, y, *z;

The declaration of an object is similar to that of a variable of any basic type. The necessary memory space is allocated to an object at this stage.

Note that class specification provides only a *template* and does not create any memory space for the objects.

How can you access the class members?

The object can access the public class members of a class by using dot operator or arrow operator. The syntax is as follows;

Objectname operator membername;

Example:

```
x.show();
z->show(); {z is a pointer to class item}
```

What is an access specifier? Explain about various access specifiers and their scope.

The access specifier specifies the accessibility of the data members declared inside the class. They are:

1. public
2. private
3. protected

public keyword can be used to allow object to access the member variables of class directly. The keyword **public** followed by colon (:) means to indicate the data member and member function that visible outside the class. Consider the following example:

```
class Test
{
    public:
        int x, y;       // variables declaration
        void show()
        {
                cout<<x<<y;
        }
```

```
};

int main()
{
    Test t;             //Object creation
    t.x = 10;
    t.y = 20;
    t.show();
};
```

Now, it display 10 and 20

Private keyword is used to prevent direct access to member variables or functions by the object. It is the **default access**. The keyword **private** followed by colon (:) is used to make data member and member function visible with in the class only.

Consider the following example:

```
class Test
{
     private:
         int x, y;        // variables declaration
};

int main()
{
    Test t;             //Object creation
    t.x = 10;
    t.y = 20;
};
```

Now it raises two common compile time errors:

„Test::x“ is not accessible
„Test::y“ is not accessible

Protected access is the mechanism same as private. It is frequently used in inheritance. Private members are not inherited at any case whereas protected members are inherited. The keyword **private** followed by colon (:) is used to make data member and member function visible with in the class and its child classes.

What is a member function? How can you declare member functions? Member functions are defined in two places. They are;

1. Inside the class
2. Outside the class

Inside the class

Member functions can be defined immediately after the declaration inside the class. These functions by default act as **inline functions**.

Example:

```
class student
{
        private:
                 int rno;
                char *sname;
        public:
```

```
        void print()
        {
                cout<<"rno="<<rno;
                cout<<"name="<<sname;
        }
        void read(int a, char *s)
        {
                rno = a;
                strcpy(snm, s);
        }
};
void main()
{
        student s;
        s.read(10, "Rama");
        s.print();
}
```

Output:

rno= 10

sname = Rama

Outside the class

- ✓ Member functions that are declared inside a class have to be defined separately outside the class. Their definitions are very much like the normal functions.
- ✓ The main difference is the member function incorporates a membership identity label in the header. This label tells the compiler which class the function belongs to.

General form

```
return-type classname :: function-name (list of arguments)
{
        // function body
}
```

Example:

```
class student
{
        private:
                int rno;
                char *sname;
        public:
                void read(int , char*);
                void print();
};
void student :: read (int a, char *s)
{
        no = a;
        strcpy(sname, s);
}
void student :: print()
{
        cout<<"rno="<<rno;
        cout<<"name="<<sname;
}
void main()
{
        student s;
        s.read(10, "Rama");
        s.print();
}
```

Output:

rno= 10

sname = Rama

What are the characteristics of member functions?

- ✓ The member functions are accessed only by using the object of the same class.
- ✓ The same function can be used in any number of classes.
- ✓ The private data or private functions c an be accessed only by public member functions.

How can you declare outside member function as inline? Explain with an example. (OR) Explain about inline keyword.

- ✓ Inline function mechanism is useful for small functions.
- ✓ The inline functions are similar to macros. By default, all the member functions defined inside the class are inline functions.
- ✓ The member functions defined outside the class can be made inline by prefixing "inline" to the function declaration.

General form:

```
inline Return-type classname :: function-name (list of arguments)
{
        //function body
}
```

Example:

```
class student
{
    private:
        int rno;
        char *sname;
    public:
        void print();
        void read(int a, char *s)
        {
            no = a;
            strcpy(snm, s);
        }
};
inline void student :: print()
{
    cout<<"no="<<no;
    cout<<"name="<<snm;
}
void main()
{
    student s;
    s.read(10, "Rama");
    s.print();
}
```

Output:
rno= 10
sname = Rama

What is meant by data hiding? (OR) How data hiding is accomplished in C++? (OR) Explain about classes. (OR) Explain about data encapsulation.

Data hiding is also called as data encapsulation. An encapsulated object is called as abstract data type. Data hiding is useful for protecting data. It is achieved by making the data variables of class as private. Private variables cannot directly accessible to the outside of the class. The keywords private and protected are used to protect the data.

Access Specifier	Members of the Class	Class Objects
Public	Yes	Yes
Private	Yes	No
Protected	Yes	No

Example:

```
class ex
{
        private:
                 int p;
        protected:
                 int q;
        public:
                int r;
        void getp()
        {
                p=10;
                cout<<"private="<<p;
        }
        void getq()
        {
                q=10;
                cout<<"protected="<<q;
        }
        void getr()
        {
                r=10;
                cout<<"public ="<<r;
        }
};

void main()
{
        ex e;
        e.getp();
        e.getq();
        e.getr();
        e.r=100;
        e.getr();
}
```

Output:

```
private =10
protected=10
public =10
public =100
```

How can you overload a member function? Explain with an example.

Member functions can be overloaded like any other normal functions. Overloading means one function is defined with multiple definitions with same functions name in the same scope.

The following program explains the overloaded member functions

```
class absv
{
        public:
                int num(int i);
                double num(double d);
};
int absv:: num(int i)
```

```
{
        return (abs(i));
}
double absv:: num(double d)
{
        return (fabs(d));
}
void main()
{
        absv a;
        cout<<"\nThe absolute value of -10 is "<<n.num(-10);
        cout<<"\nThe absolute value of -12.35 is "<<n.num(-12.35);
}
```

Output:
The absolute value of -10 is 10
The absolute value of -12.35 is 12.35

What is a nested class? Explain with an example.
When a class defined in another class, it is known as nesting of classes. In nested classes, the scope of inner class is restricted by outer class

The following program illustrates the nested classes:

```
class A
{
public :
        class B
        {
                void show()
                {
                        cout<<"\nC++ is wonderful language";
                }
         };
};

int main(void)
{
        A::B x;
        x.show();
}
```

Output:
C++ is wonderful language

What is a constructor? What is a destructor? What are their properties?

A constructor is a special member function used for automatic initialization of an object. Whenever an object is created, the constructor is called automatically. Constructors can be overloaded. Destructor destroys the object. Constructors and destructors having the same name as class, but destructor is preceded by a tilde (~) operator. The destructor is automatically executed whenever the object goes out of scope.

Characteristics of C*onstructors*

- ✓ Constructors have the same name as that of the class they belongs to.
- ✓ Constructors must be declared in public section.
- ✓ They automatically execute whenever an object is created.

- ✓ Constructors will not have any return type even void.
- ✓ Constructors will not return any values.
- ✓ The main function of constructor is to initialize objects and allocation of memory to the objects.
- ✓ Constructors can be called explicitly.
- ✓ Constructors can be overloaded.
- ✓ A constructor without any arguments is called as default constructor.

What are the applications of constructors? (Default constructor)

Constructors are used to initialize member variables of a class. Constructors allocate required memory to the objects. Constructors are called automatically whenever an object is created. A constructor which is not having any arguments are said to be default constructor.

Example:

```
class num
{
        int a, b;
    public:
        num()
        {
                a=5; b=2;
        }
        void show()
        {
                cout<<a<<b;
        }
};
Void main()
{
        num n;
        n.show();
}
```

Output:

5 2

Explain about parameterized constructors with an example.

- ✓ It may be necessary to initialize the various data elements of different objects with different values when they are created. This is achieved by passing arguments to the constructor function when the objects are created.
- ✓ The constructors that can take arguments are called parameterized constructors.
- ✓ They can be called explicitly or implicitly.

Example:

```
class num
{
        int a, b ,c;
    public:
        num(int x, int y)
        {
                a=x; b=y;
        }
        void show()
        {
                cout<<a<<b;
        }
```

Output:

10 20

1 2

```
};
void main()
{
        num n(10,20); //implicit call
        n.show();
        num x= num(1,2); //explicit call
        x.show();
}
```

What is meant by constructor overloading? Explain with an example.
(Multiple Constructors)

- ✓ Similar to normal functions, constructors also overloaded. C + + permits to use more than one constructors in a single class.

- ✓ If a class contains more than one constructor. This is known as constructor overloading.

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

Example:

```
class num
{
        int a, b;
     public:
        num()          // Default constructor
        {
                a=10; b=20;
        }
        num(int x, int y)      // Parameterized constructor

        {
                a=x;
                b=y;
        }
        void add()
        {
                cout<<a+b;
        }
};
void main()
{
        num n;

        n.add();

        num x(1,2);

        x.add();
}
```

Output:

30

3

The num class has two constructors. They are;

- ✓ Default constructor
- ✓ Parameterized constructor

Whenever the object "n" is created the default constructor is executed automatically and a, and b values are initialized to 10, 20 respectively.

Whenever the object "x" is created the parameterized constructor is executed automatically and a, and b values are assigned with 1 and 2 respectively.

Explain about constructors with default argument with an example.

- ✓ Similar to functions, It is possible to define constructors with default arguments.
- ✓ Consider power(int n, int p= 2);
  - The default value of the argument p is two.
  - power p1 (5) assigns the value 5 to *n* and 2 to *p*.
  - power p2(2,3) assigns the value *2* to n and 3 to *p*.

Example:

```
class power
{
        int b,p;
     public:
        power(int n=2, int m=3)
        {
                b=n;
                p=m;
                cout<<pow(n,m);
        }
};
void main()
{
        power x;
        Power y(5);
        power z(3,4);

}
```

Output:

```
8    125  81
```

Explain about copy constructor with an example.

- ✓ Copy constructor is used to declare and initialize an object from another object.
- ✓ A copy constructor takes a reference to an object of the same class as itself as an argument.

Ex:

```
class Test
{
        int i;
     public:
        Test()                  // Default constructor
        {
                i=0;
        }
        Test (int a)            // Parameterized constructor
        {
                i = a;
        }
        Test (code &x)          //Copy Constructor
        {
                i = x.i;
        }
        void show()
        {
```

```
            cout<<i<<endl;
        }
};
void main()
{
        Test a(100);
        a.show();
        Test b(a);          //Copy Constructor invoked
        b.show();
}
```

Output:
100 100

Explain about Destructors with an example.

- ✓ Destructor is a special member function like a constructor. Destructors destroy the class objects that are created by constructors.
- ✓ The destructor have the same name as their class, preceded by a ~.
- ✓ The destructor neither requires any arguments nor returns any values.
- ✓ It is automatically executed when the object goes out of score.
- ✓ Destructor releases memory space occupied by the objects.

Characteristics of Destructors

- ✓ Their name is the same as the class name but is preceded by a tilde(~).
- ✓ They do not have return types, not even void and they cannot return values.
- ✓ Only one destructor can be defined in the class.
- ✓ Destructor neither has default values nor can be overloaded.
- ✓ We cannot refer to their addresses.
- ✓ An object with a constructor or destructor cannot be used as a member of a union.
- ✓ They make „implicit calls" to the operators ***new*** and ***delete*** when memory allocation/ memory de-allocation is required.
- ✓

Example:

```
class Test
{
    public:
        Test()
        {
            cout<<"\n Constructor Called";
        }
        ~Test()
        {
            cout<<"\n Destructor Called";
        }
};
void main()
{
        Test t;
}
```

Output:
Constructor Called
Destructor Called

What is meant by anonymous objects? Explain.

It is possible to declare objects without any name. These objects are said to be anonymous objects. Constructors and destructors are called automatically whenever an object is created and destroyed respectively. The anonymous objects are used to carry out these operations without object.

Ex:

```
class noname
{
        int x;
      public:
        noname()
        {
                 cout<<"\n In Default Constructor";
                x=10;
                cout<<x;
        }
        noname(int i)
        {
                 cout<<"\n In Parameterized Constructor";
                x=i;
                cout<<x;
        }
        ~noname()
        {
                cout <<"\n In Destructor";
        }
};
void main()
{
        noname();
        noname(12);
}
```

Output:
In Default Constructor 10
In Parameterized Constructor 12
In Destructor
In Destructor