UNIT-3

CLASS

The classes are the most important feature of C++ that leads to Object Oriented programming. The class is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.

The variables inside class definition are called as data members and the functions are called member functions.
For example Class of birds, all birds can fly and they all have wings and beaks. So here flying is a behavior and wings and beaks are part of their characteristics. And there are many different birds in this class with different names but they all possess this behavior and characteristics.

Similarly, the class is just a blueprint, which declares and defines characteristics and behavior, namely data members and member functions respectively. And all objects of this class will share these characteristics and behavior.

Properties of a Class

The class name must start with an uppercase letter (Although this is not mandatory). If the class name is made of more than one word, then the first letter of each word must be in uppercase.

Example, class Study, class CDGI etc

- Classes contain, data members and member functions, and the access of these data members and variable depends on the access specifiers (discussed in next section).
- Class's member functions can be defined inside the class definition or outside the class definition.
- Class in C++ are like structures in C , the only difference being, class defaults to private access control, whereas structure defaults to public.
- All the features of OOPS, revolve around classes in C++. Inheritance, Encapsulation, Abstraction etc.
- Objects of the class hold separate copies of data members. We can create as many objects of a class as we need.

- Classes do possess more characteristics like we can create abstract classes, immutable classes, all this we will study later.

Objects: -

The class is mere a blueprint or a template. No storage is assigned when we define a class. Objects are instances of the class, which holds the data variables declared in the class and the member functions work on these class objects.

```
#include <iostream>
using namespace std;
class Box {
   public:
      double length; // Length of a box
      double breadth; // Breadth of a box
      double height; // Height of a box
};
int main() {


   Box Box1; // Declare Box1 of type Box
   Box Box2; // Declare Box2 of type Box
   double volume = 0.0; // Store the volume of a box here
   // box 1 specification
   Box1.height = 5.0;
   Box1.length = 6.0;
   Box1.breadth = 7.0;
   // box 2 specification
   Box2.height = 10.0;
   Box2.length = 12.0;
   Box2.breadth = 13.0;
   // volume of box 1
   volume = Box1.height * Box1.length * Box1.breadth;
   cout << "Volume of Box1:" << volume <<endl;
   // volume of box 2
   volume = Box2.height * Box2.length * Box2.breadth;
   cout <<"Volume of Box2:" << volume <<endl;
   return 0;
```

```
}
```

Each object has different data variables. Objects are initialized using special class functions called Constructors.

```
class CDGI
{
int x;
void display(){} //empty function
};
int main()
{
CDGI obj; // Object of class CDGI created
}
```

Access Specifier in C++

Access specifiers in C++ class define the access control rules.

1. public
2. private
3. protected

Access specifiers in the program, are followed by a colon. You can use either one, two or all 3 specifiers in the same class to set different boundaries for different class members.

Public

Public means all the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes too. Hence there are chances that they might change them. So, the key members must not be declared public.

```
class PublicAccess
{
public: // public access specifier
int x; // Data Member Declaration
```

```
void display(); // Member Function decaration
}
```

Private

Private keyword means that no one can access the class members declared private outside that class. If someone tries to access the private member, they will get a compile-time error. By default, class variables and member functions are private.

```
class PrivateAccess
{
private: // private access specifier
int x; // Data Member Declaration
void display(); // Member Function declaration
}
```

Protected

Protected, is the last access specifier, and it is like private, it makes class member inaccessible outside the class. But they can be accessed by any subclass of that class. (If class A is inherited by class B, then class B is a subclass of class A.

```
class ProtectedAccess
{
protected: // protected access specifier
int x; // Data Member Declaration
void display(); // Member Function decaration
}
```

SCOPE Resolution Operator: -

The:: (scope resolution) operator is used to qualify hidden names so that you can still use them. You can use the unary scope operator if a namespace scope or global scope name is hidden by an explicit declaration of the same name in a block or class.

Usage of Scope Resolution Operator

1. To access a global variable when there is a local variable with same name:

```
// C++ program to show that we can access a global variable
// using scope resolution operator:: when there is a local
// variable with same name
#include<iostream>


using namespace std;
int x; // Global x
int main()
{
    int x = 10; // Local x
    cout << "Value of global x is " <<::x;
    cout <<"\nValue of localx is" << x;
    return 0;
}
```

2. To define a function outside a class.

```
// C++ program to show that scope resolution operator:: is used
// to define a function outside a class
#include<iostream>
using namespace std;
class A
{
public:
    // Only declaration
    void fun();
};
// Definition outside class using ::
void A::fun()
{
    cout << "fun() called";
}
int main()
{
```

```
    A a;
    a.fun();
    return 0;
}
```

3. To access a class's static variables.
4. In case of multiple Inheritance:

Constructor in C++

A class constructor is a special member function of a class that is executed whenever we create new objects of that class.

A constructor will have an exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.

Special characteristics of Constructors:

- They should be declared in the public section
- They do not have any return type, not even void
- They get automatically invoked when the objects are created
- They cannot be inherited though derived class can call the base class constructor
- Like other functions, they can have default arguments
- You cannot refer to their address
- Constructors cannot be virtual

```
#include <iostream>
using namespace std;
class Line {
   public:
      void setLength( double len );
      double getLength( void );
      Line(); // This is the constructor
   private:
      double length;
};
// Member functions definitions including constructor
```

```
Line::Line(void) {
    cout << "Object is being created" << endl;
}
void Line::setLength( double len ) {
    length = len;
}
double Line::getLength( void ) {
    return length;
}
// Main function for the program
int main() {
    Line line;
    // set line length
    line.setLength(6.0);
    cout << "Length of line:" << line.getLength() <<endl;
    return 0;
}
class A
{
int x;
public:
A(); //Constructor
};
```

Constructors are special class functions which perform initialization of every object. The Compiler calls the Constructor whenever an object is created. Constructors initialize values to object members after storage is allocated to the object.

Types of Constructors

Constructors are of three types:

- Default Constructor

The default constructor is the constructor which doesn't take any argument. It has no parameter.

- Parametrized Constructor

These are the constructors with a parameter. Using this Constructor, you can provide different values to data members of different objects, by passing the appropriate values as an argument.

- Copy Constructor

These are a special type of Constructors which takes an object as an argument and is used to copy values of data members of one object into another object. We will study copy constructors in detail later.

Destructors

The destructor is a special class function which destroys the object as soon as the scope of an object ends. The destructor is called automatically by the compiler when the object goes out of scope.

The syntax for destructor is same as that for the constructor, the class name is used for the name of the destructor, with a tilde ~ sign as a prefix to it.

```
class A
{
public:
~A();
};
```

Destructors will never have any arguments.

Friend Function: -

A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class. Even though the prototypes for friend functions appear in the class definition, friends are not member functions.

A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all its members are friends.

To declare a function as a friend of a class, precede the function prototype in the class definition with keyword friend as follows -

```
class Box {
double width;
public:
double length;
```

```
friend void printWidth ( Box box );
void setWidth( double wid );
};
```

To declare all member functions of class ClassTwo as friends of class ClassOne, place a following declaration in the definition of class ClassOne.

Inheritance in C++

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.

When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class.

The idea of inheritance implements the is a relationship. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.

Base and Derived Classes

A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes. To define a derived class, we use a class derivation list to specify the base class(es). A class derivation list names one or more base classes and has the form -

class derived-class: access-specifier base-class

Where access-specifier is one of public, protected, or private, and base-class is the name of a previously defined class. If the access-specifier is not used, then it is private by default.

```
#include <iostream>
using namespace std;
// Base class
class Shape {
   public:
      void setWidth(int w) {
```

```
        width = w;
    }
    void setHeight(int h) {
        height = h;
    }

protected:

    int width;
    int height;
};
// Derived class
class Rectangle: public Shape {
    public:
        int getArea() {
            return (width * height);
        }
};
int main(void) {
    Rectangle Rect;
    Rect.setWidth(5);
    Rect.setHeight(7);
    // Print the area of the object.
    cout << "Total area:" << Rect.getArea() << endl;
    return 0;
}
```

Summary of Access type in Inheritance using different access specifiers.

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

- Constructors, destructors and copy constructors of the base class.
- Overloaded operators of the base class.
- The friend functions of the base class.

Type of Inheritance

When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance.

We hardly use protected or private inheritance, but public inheritance is commonly used. While using different type of inheritance, following rules are applied -

Public Inheritance - When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class but can be accessed through calls to the public and protected members of the base class.

Protected Inheritance - When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.

Private Inheritance - When deriving from a private base class, public and protected members of the base class become private members of the derived class.

Polymorphism in C++

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. Real life example of polymorphism, a person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So, the same person possesses have different behavior in different situations. This is called polymorphism.

Polymorphism is considered as one of the important features of Object Oriented Programming.
In C++ polymorphism is mainly divided into two types:

- Compile time Polymorphism
- Runtime Polymorphism

Compile time polymorphism: This type of polymorphism is achieved by function overloading or operator overloading.
Function Overloading: When there are multiple functions with the same name but different parameters then these functions are said to be overloaded. Functions can be overloaded by change in number of arguments or/and change in type of arguments
Example

```
#include <iostream.h>
using namespace std;
class CDGI
{
public:
// function with 1 int parameter
void func(int x )
{
cout << "value of x is " << x ≪ endl;
}
// function with same name but 1 double parameter
void func(double x)
{
cout << "value of x is " << x ≪ endl;
}
// function with same name and 2 int parameters
void func(int 𝑥, int 𝑦 )
{
cout << "value of 𝑥 and 𝑦 is " ≪ 𝑥 ≪ "," << 𝑦 ≪ endl;
}
};
```

```
int main() {
    CDGI obj1;
    // Which function is called will depend on the parameters passed
    // The first 'func' is called
    obj1.func(7);
    // The second 'func' is called
    obj1.func(9.132);
    // The third 'func' is called
```

```
    obj1.func(85,64);
    return 0;
}
```

Operator Overloading: C++ also provide an option to overload operators. For example, we can make the operator ('+') for string class to concatenate two strings. We know that this is the addition operator whose task is to add to operands. So a single operator ' + ' when placed between integer operands, adds them and when placed between string operands, concatenates them.

Example:

```
// CPP program to illustrate
// Operator Overloading
#include<iostream>
using namespace std;
class Complex{
private:
    int real, imag;
public:
    Complex(int r = 0, int i =0) {real = r; imag = i;}
    // This is automatically called when ' }+\mathrm{ ' is used with
    // between two Complex objects
    Complex operator + (Complex const &obj) {
        Complex res;
        res.real = real + obj.real;
        res.imag = imag + obj.imag;
        return res;
    }
    void print() { cout << real <<" + i" << imag << endl; }
};
int main()
{
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2;// An example call to "operator+"
    c3.print();
}
```

Virtual Functions

Virtual Function is a function in a base class, which is overridden in the derived class, and which tells the compiler to perform Late Binding on this function.
Virtual Keyword is used to make a member function of the base class Virtual.

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for the function call.
- They are mainly used to achieve Runtime polymorphism
- Functions are declared with a virtual keyword in the base class.
- The resolving of the function call is done at Run-time.

Rules for Virtual Functions

- They Must be declared in public section of the class.
- Virtual functions cannot be static and cannot be a friend function of another class.
- Virtual functions should be accessed using pointer or reference of the base class type to achieve runtime polymorphism.
- The prototype of virtual functions should be same in the base as well as derived class.
- They are always defined in a base class and overridden in the derived class. It is not mandatory for a derived class to override (or re-define the virtual function), in that case, base class version of the function is used.
- A class may have virtual destructor but it cannot have a virtual constructor.

Runtime Polymorphism

Runtime polymorphism is achieved only through a pointer (or reference) of base class type. Also, a base class pointer can point to the objects of the base class as well as to the objects of the derived class.
Late binding(Runtime) is done in accordance with the content of pointer (i.e. location pointed to by pointer) an Early binding (Compile time) is done according to the type of pointer, since print() function is declared with virtual keyword so it will be binded at run-time (output is print derived class as pointer is pointing to object of derived class) and show() is non-virtual so it will be binded during compile time(output is show base class as pointer is of base type).

Introduction to Data Structure

Data Structure is a way of collecting and organizing data in such a way that we can perform operations on these data in an effective way. Data Structures is about rendering data elements in terms of some relationship, for better organization and storage. For example, we have data player's name "Virat" and age 26. Here "Virat" is of String data type and 26 is of integer data type.

We can organize this data as a record like Player record. Now we can collect and store player's records in a file or database as a data structure. For example "Dhoni" 30, "Gambhir" 31, "Sehwag" 33

In simple language, Data Structures are structures programmed to store ordered data, so that various operations can be performed on it easily. It represents the knowledge of data to be organized in memory. It should be designed and implemented in such a way that it reduces the complexity and increases the efficiency.

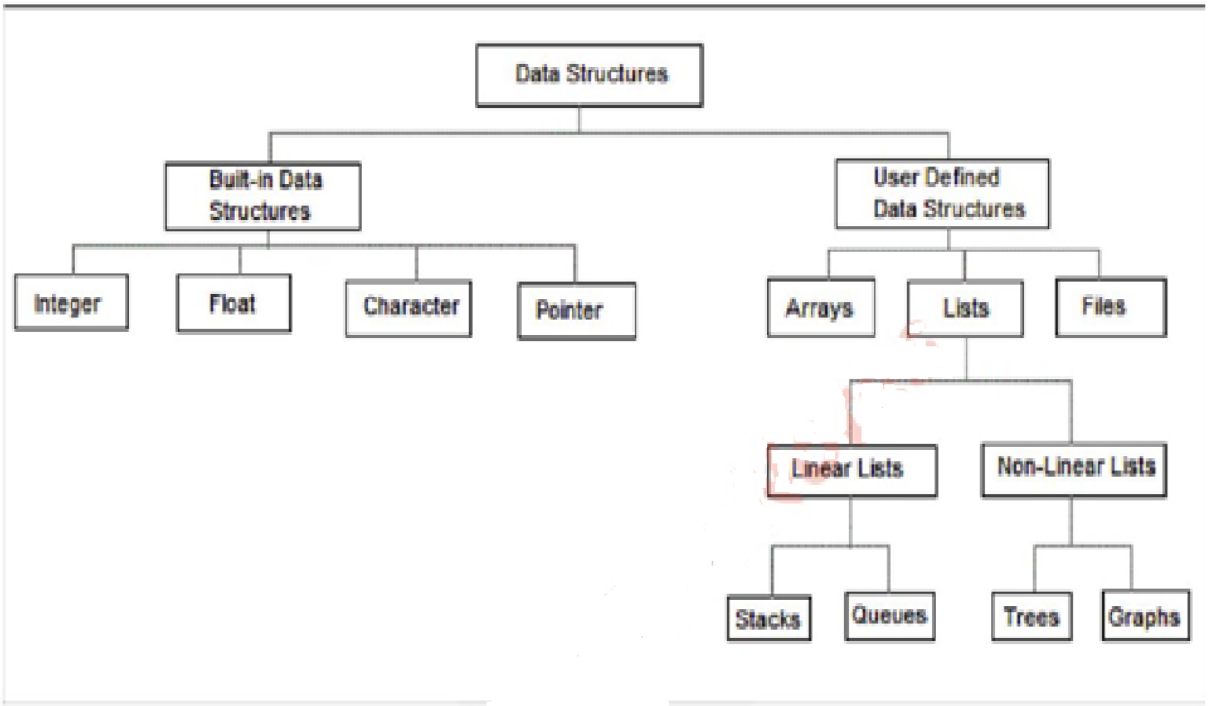
Basic types of Data Structures

As we have discussed above, anything that can store data can be called as a data structure, hence Integer, Float, Boolean, Char etc, all are data structures. They are known as Primitive Data Structures.

Then we also have some complex Data Structures, which are used to store large and connected data. Some example of Abstract Data Structure is :

- Linked List
- Tree
- Graph
- Stack, Queue etc.

All these data structures allow us to perform different operations on data. We select these data structures based on which type of operation is required. We will consider these data structures in more details in our later lessons.



The data structures can also be classified based on the following characteristics:

Characteristic	Description
Linear	In Linear data structures, the data items are arranged in a linear sequence. Example: Array
Non-Linear	In Non-Linear data structures, the data items are not in sequence. Example: Tree, Graph
Homogeneous	Inhomogeneous data structures, all the elements are of the same type. Example: Array
NonHomogeneous	In the Non-Homogeneous data structure, the elements may or may not be of the same type. Example: Structures
Static	Static data structures are those whose sizes and structures associated memory locations are fixed, at compile time. Example: Array
Dynamic	Dynamic structures are those which expands or shrinks depending on the program need and its execution. Also, their associated memory locations changes. Example: Linked List created using pointers

If you have any queries please visit- https://studywithakash.in/

Gmail – studywithakash311@gmail.com

+918871317984

THANK YOU