Week 5: Lecture 9

OOP C++

The main purpose of C++ programming is to add object orientation to the C programming language and classes are the central feature of C++ that supports object-oriented programming and are often called user-defined types.

A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package. The data and functions within a class are called members of the class.

C++ Class Definitions

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we defined the Box data type using the keyword **class** as follows –

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

The keyword **public** determines the access attributes of the members of the class that follows it. A public member can be accessed from outside the class anywhere within the scope of the class object. You can also specify the members of a class as **private** or **protected** which we will discuss in a sub-section.

Define C++ Objects

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box –

```
Box Box1;               // Declare Box1 of type Box
Box Box2;               // Declare Box2 of type Box
```

Both of the objects Box1 and Box2 will have their own copy of data members.

Constructors

A constructor is a special type of member function that is called automatically when an object is created.

In C++, a constructor has the same name as that of the class and it does not have a return type. For example,

```
class People

{

public:

People()

{ //body}

};
```

Here, the function People() is a constructor of the class People. Notice that the constructor –

- Has the same name as the class.
- Does not have a return type.
- Is public.

Parameterized Constructor

In C++, a constructor with parameters is known as a parameterized constructor. This is the preferred method to initialize member data.

```
class Rectangle

{

int length;

int height;

public:

Rectangle(int l, int h)

{

length = l;

height = h;

}
```

Copy Constructor

A copy constructor is a member function that initializes an object using another object of the same class. A copy constructor has the following general function prototype:

ClassName (const ClassName &old_obj);

When is copy constructor called?
In C++, a Copy Constructor may be called in the following cases:
1. When an object of the class is returned by value.
2. When an object of the class is passed (to a function) by value as an argument.
3. When an object is constructed based on another object of the same class.
4. When the compiler generates a temporary object.

Destructor

Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.

How are destructors different from a normal member function?

Destructors have same name as the class preceded by a tilde (~)
Destructors don’t take any argument and don’t return anything.

Can there be more than one destructor in a class?

No, there can only one destructor in a class with classname preceded by ~, no parameters and no return type.

Example:

```
class Rectangle

{

int length;

int height;

public:

Rectangle() //constructor

{

//body

}

~ Rectangle() //destructor

{
}
```

Passing objects as arguments

The objects of a class can be passed as arguments to member functions as well as nonmember functions either by value or by reference. When an object is passed by value, a copy of the actual object is created inside the function. This copy is destroyed when the function terminates.

```
class SUM
{
private:
   int value;
public:
   SUM()
   {
      value = 0;
   }
   SUM(int S)
   {
      value = S;
   }
void add(SUM s)
   {
     value = value+s.value;
   }
   void show()
   {
      cout<<"Value is: "<<value<<endl;
   }
};
```

Returning object from function

A function can also return objects either by value or by reference. When an object is returned by value from a function, a temporary object is created within the function, which holds the return value. This value is further assigned to another object in the calling function.

Syntax:

```
Class_name function_name(parameter_list)
{
//body of the function
}
```

Difference between Structure and Union

Struct	Union
The struct keyword is used to define a structure.	The union keyword is used to define union.
When the variables are declared in a structure, the compiler allocates memory to each variables member. The size of a structure is equal or greater to the sum of the sizes of each data member.	When the variable is declared in the union, the compiler allocates memory to the largest size variable member. The size of a union is equal to the size of its largest data member size.
Each variable member occupied a unique memory space.	Variables members share the memory space of the largest size variable.
Changing the value of a member will not affect other variables members.	Changing the value of one member will also affect other variables members.
Each variable member will be assessed at a time.	Only one variable member will be assessed at a time.
We can initialize multiple variables of a	In union, only the first data member can be

structure at a time.	initialized.
All variable members store some value at any point in the program.	Exactly only one data member stores a value at any particular instance in the program.
The structure allows initializing multiple variable members at once.	Union allows initializing only one variable member at once.
It is used to store different data type values.	It is used for storing one at a time from different data type values.
It allows accessing and retrieving any data member at a time.	It allows accessing and retrieving any one data member at a time

Storage Classes

Storage Classes are used to describe the features of a variable/function. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.

There are following storage classes, which can be used in a C++ Program

- automatic
- register
- static
- extern
- mutable

Automatic Storage Class

It is the default storage class for all local variables. The auto keyword is applied to all local variables automatically.

```
int main()
{
   int month;
   auto int m = 7;
   cout<<m<<endl;
}
```

auto can only be used within functions

Register storage class

```
int main()
{
   cout << "Demonstrating register class"<<endl;
   register char b = 'T';
   cout<<b<<endl;;
   return 0;
}
```

Static storage class

```
void func()
{
   static int i = 0;
   int j = 0;
   i++;
   j++;
   cout<<"i = "<<i<<" "<<"j = "<<j<<endl;
}
int main()
{
   func();
   func();
   func();
}
```

Static class data

```
class Counter
{
private:
    static int Count;
public:
    void setCount()
    {
        Count++;
    }
    int getCount()
    {
        return Count;
    }
};
int Counter::Count = 0;
int main()
{
    Counter c1,c2,c3;
    c1.setCount();
    cout<<"Count is: "<<c1.getCount()<<endl;
    c2.setCount();
    cout<<"Count is: "<<c2.getCount()<<endl;
    c3.setCount();
    cout<<"Count is: "<<c3.getCount()<<endl;
    return 0;
}
```

Storage Class	Keyword	Lifetime	Visibility	Initial Value
Automatic	auto	Function block	Local	Garbage
Register	register	Function block	Local	Garbage
Static	static	Whole program	Local	Zero

Extern	extern	Whole program	Global	Zero
Mutable	mutable	Class	Local	Garbage

What is a Function?

A function is a unit of code that is often defined by its role within a greater code structure. Specifically, a function contains a unit of code that works on various inputs, many of which are variables, and produces concrete results involving changes to variable values or actual operations based on the inputs.

Component	Purpose	Example
Declaration (Prototype)	Specifies function name, argument types, and return value. Alerts compiler that a function is coming up later.	void func();
Call	Causes the function to be executed	func();
Definition	The function itself. Contains the lines of code that constitute the function.	void func() { //lines of code }

Pass by Value	**Pass by Reference**
Mechanism of copying the function parameter value to another variable	Mechanism of passing the actual parameters to the function.
Changes made inside the function are not reflected in the original value.	Changes made inside the function are reflected in the original value.
Makes a copy of the actual parameter.	Address of the actual parameter passes to the function.
Function gets a copy of the actual parameter.	Function accesses the original variable's content.
Requires more memory	Requires less memory
Requires more time as it involves copying values.	Requires a less amount of time as there is no copying.

Inline Function

When the program executes the function call instruction the CPU stores the memory address of the instruction following the function call, copies the arguments of the function on the stack and finally transfers control to the specified function.

The CPU then executes the function code, stores the function return value in a predefined memory location/register and returns control to the calling function. This can become overhead if the execution time of function is less than the switching time from the caller function to called function (callee).

```
inline return-type function-name(parameters)
{  // function code }
```

It is also possible to define the inline function inside the class. In fact, all the functions defined inside the class are implicitly inline. Thus, all the restrictions of inline functions are also applied here. If you need to explicitly declare inline function in the class then just declare the function inside the class and define it outside the class using inline keyword.

```
class S
{
public:
   inline int square(int s)  // redundant use of inline
   {
        // function body
    }
};
```

Arrays of objects

An array is a collection of similar data items stored at contiguous memory locations. Elements can be accessed randomly using indices of an array.

```
class Student
{
private:
    int id;
    string name;
public:
    void getData()
    {
        cout<<"Enter Id: ";
        cin>>id;
        cout<<"Enter name: ";
        cin>>name;
    }
void putData()
    {
        cout<<"Id: "<<id<<endl;
        cout<<"Name: "<<name<<endl;
    }
};
int main()
{
    Student st[5];
    int i;
    for(i = 0; i<5; i++)
    {
        st[i].getData();
    }
    for(i = 0;i<5;i++)
    {
        st[i].putData();
    }
    return 0;
}
```

Strings in c++

The two headers are completely different. cstring is inherited from C and provides functions for working with C-style strings (arrays of char terminated by '\0'). string was born in C++ and defines the std::string class along with its non-member functions.

```
int main()
{
    string s1("Man");
    string s2 = "Beast";
    string s3;
    s3 = s1;
    cout<<"s3 = "<<s3<<endl;
    s3= "Neither "+s1+" nor ";
    s3+=s2;
    cout<<"s3 = "<<s3<<endl;
s1.swap(s2);
    cout<<s1<<" nor "<<s2<<endl;
    return 0;
}
```

Code Explanation:

Here the first 3 lines of code show three ways to define string objects. The first two initialize strings, and the second creates an empty string variable. The next line shows simple assignment with the = operator.

The string class uses a number of overloaded operators. The overloaded + operator concatenates one string object with another. You can also use the += operator to append a string to the end of an existing string. The statement is:

```
s3 += s2;
```

The example also introduces our first string class member function: swap(), which exchanges the values of two string objects. It’s called for one object with the other as an argument.