Unit-3
Classes and Objects

Class:

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

The variables inside class definition are called as data members and the functions are called member functions and the access of these data members depends on the access specifiers

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 posses this behavior and characteristics.

Similarly, class is just a blue print, 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.

Class in C++ are similar to structures in C, the only difference being, class defaults to private access control, where as structure defaults to public.

Syntax:

```
class class_name
{
    Access specifier: Data members;
       Member functions(){}
};
```

Example

```
class employee
{
    public:
       int empid;
       string empname;
       float salary;
};
```

Class definition starts with the keyword **class** followed by the class name and ends with a **semicolon (;).**
The primary purpose of a class is to hold data/information.

Member Function of a Class

Member function of a class is a function that must be declared inside the class.
The member functions can be defined as

1. Inside Member Function
2. Outside Member Function

1. Inside Member Function:

Inside member function class can be declared in public or private section.

Example : Program to demonstrate member function
Accessing private member of a class using member function

```
#include <iostream.h>
class employee
{
    private:              //Private section starts
        int empid;
        float esalary;
    public:             //Public section starts
        void display()     //Member function
        {
            empid = 1;
            esalary = 10000;
            cout<<"Employee Id is : "<<empid<<endl;
            cout<<"Employee Salary is : "<<esalary<<endl;
        }
};

int main()
{
     employee e;      //Object Declaration
     e.display();       //Calling to member function
     return 0;
}
```

Output:
Employee Id is : 1
Employee Salary is : 10000

In the above program, the member function **display()** is defined inside the class in public section.
In **int main()** function, object **'e'** is declared. An object has permission to access the public members of the class.
The object **'e'** invokes the public member function **display().**

The public member function can access the private members of the same class. The **display()** function initializes the private member variables and displays the contents on the console.

2. Outside Member Function

If a function is small and inside the class, it is considered as an **Inline function.**
If a function is large, it should be defined outside the class.
When the member function of a class is defined outside the class they are called as **Externally defined member function.**

Scope Resolution (::) operator is used to define the member function of the class outside the scope and prototype declaration of function must be declared inside the class.

Syntax:

```
Return_type Class_name :: Function_name(argument_list)
{
    //Statements;
}
```

Example : Define member function class outside the scope/class

```
#include <iostream>
class employee
{
    private:                //Private section starts
        int empid;
        float esalary;
    public:                 //Public section starts
        void display(void);     //Prototype Declaration
};  //End of class
void employee :: display()     //Function Definition Outside the Class
{
    empid = 1;
    esalary = 10000;
    cout<<"Employee Id is : "<<empid<<endl;
    cout<<"Employee Salary is : "<<esalary<<endl;
}
int main()
{
    employee e;      //Object Declaration
    e.display();       //Calling to public member function
    return 0;
}
```

Output:

```
Employee Id is : 1
```

Employee Salary is : 10000

In the above program, the prototype declaration of **display()** function is declared inside the class terminated by class definition.
The function body of **display()** function is defined inside the class.
The function declaration of **display()** function is, **void employee :: display()** where, **void** is a return type and **employee** is a class name. The **scope resolution (::) operator** separates the class name and function name, followed by the body of function which is defined.

Access Modifiers in C++

Access modifiers define the access control rules.
It is used to set boundaries for availability of members of class.
Following are the three access modifiers in C++:

1. public
2. private
3. Protected

Access modifiers in the program, are followed by a colon. You can use either one, two or all 3 modifiers in the same class to set different boundaries for different class members. They change the boundary for all the declarations that follow them.

Access Modifiers	Description
public	**It is accessible from anywhere outside the class but within a program.**
private	**It cannot be accessed or viewed from outside the class.**
protected	**It is similar to a private member but it can be accessed in child classes which are called derived classes.**

Public Access Modifier :

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 access modifier
   public:
   int x;          // Data Member Declaration
   void display();   // Member Function decaration
}
```

Private Access Modifier:

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

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

Protected Access Modifier

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

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

Object:

Class is mere a blueprint or a template. No storage is assigned when we define a class. Objects are instances of class, which holds the data variables declared in class and the member functions work on these class objects.
Each object has different data variables. Objects are initialised using special class functions called Constructors and whenever the object is out of its scope, another special class member function called Destructor is called, to release the memory reserved by the object.

Syntax:

class_name variable_name;

Example

Consider the above Employee class. Object for Employee class can be defined as:
Employee e;
Here, object **e** of **Employee** class is defined.
It is exactly the same sort of declaration that we do for the variables of different data types.

Example : Following example demonstrates the working of Objects and Class in C++

```
#include <iostream.h>
class Employee
{
    private:
        int empid;
        float salary;
    public:
        int emp_details()
        {
            empid=100;
            salary=10000.0;
        }
        int show()
        {
            cout<<"Employee Id : "<<empid<<endl;

           cout<<"Employee Salary : "<<salary<<endl;
        }
};

int main()
{
   Employee e;
   e.emp_details();
   e.show();
   return 0;
}
```

Output:
Employee Id : 100

Employee Salary : 10000

In the above program, there are two data members **empid and salary.** Two member functions **emp_details() & show()** are defined under **Employee** class.

Object **e** of the **Employee** class is declared. Function emp_details() for the object e is executed using code **e.emp_details()** which includes details of the employee. Then, function **show()** for the object **e** is executed which displays details of the employee and returns it to the calling function.

Function

Function is a block of code that performs some operation.
It is a self-contained block of statements that perform a task.
Function is used to break down the complex program into the smaller chunks.
It is useful for encapsulating common operations in a single reusable block that clearly describes what the function does.
It defines input parameters that enable callers to pass arguments into the function and returns a value as output.

Syntax:

```
return_type function_name (argument_list)
{
    //Statements;
}
```

Following are the parts of a function:

return_type	It is a data type that function returns. It is not necessary that function will always return a value.
function_name	Function name is the actual name of the function.
argument_list / parameters	It allows passing arguments to the function from the location where it is called from. The argument list is separated by comma.
Function body	It contains a collection of statements that define what the function does.

Example : Program demonstrating the Function

```
#include <iostream.h>
int addition(int no1, int no2);                    //Function declaration
int main()
{
    int num1=10, num2=20, result;                  //Local variable
    result = addition(num1,num2);       //Calling function. num1 & num2 are Actual Parameters
```

```
    cout<<"Addition is : "<<result<<endl;
    return 0;
}
int addition(int no1, int no2)     //Function definition. no1 & no2 are Formal Parameters
{
    int disp;
    disp=no1+no2;
    return disp;
}
```

Output:
Addition is : 30

Unit-3
Friend Function

Friend functions are special functions of C++ and considered to be a loophole in the Object Oriented Programming concepts.
Friend function is defined by its keyword **friend.**
Private and Protected data of class can be accessed using friend function.
It is defined inside the body of class either in private or public section.
When we define friend function, the entire class and all of its members are friends.

Syntax:

```
class class_name
{
    friend return_type function_name(arguments);
}
```

Example : Demonstrates the working structure of Friend function

```
#include <iostream.h>
class Employee
{
    private:
       int empid;
    public:
       Employee(): empid(0){ }
       friend int display(Employee);  //friend function
};
int display(Employee e)          //function definition
{
     e.empid = 100;            //accessing private data from non-member function
     return e.empid;
}
int main()
{
```

```
    Employee e;
    cout<<"Employee Id: "<<display(e);
    return 0;
}
```

Output:
Employee Id: 100

In the above program, friend function **display()** is declared inside Employee class. So, the private data can be accessed from this function.

Friend Class

It is possible to make an entire class a friend of another class. When we create a friend class then all the member functions of the friend class also become the friend of the other class. This requires the condition that the friend becoming class must be first declared or defined.

The basic **Syntax** to make class friend is :

```
friend class class-name;
```

In Above syntax, friend and class is the keyword, and class_name could be any class that already exists, if the class doesn't exist, then we must use the prototype of that class at the top of Friend class.

Example:

```
#include<iostream.h>
class A;  //class A prototype:
class B  //Class B begin:
{
   friend class A; //Friend Class Declaration:
   private:
      int Y; //Private member, need to access this member:

   public:
      B(){ Y=0;} //constructor:
};

class A //Class A definition:
{
   private:
      int X; //Private member:

   public:
    A(){X=10;}     //constructor:
```

```
    void add(B obj) //function with Class B and its Object as an parameter:
    {
       obj.Y=25; //Class B private member Access:
       cout<<obj.Y<< " + "<<X<<" = "<<obj.Y+X<<endl;
    }
};


main()
{
   A Obj1; //class A object:
   B Obj2; //Class B object:

/*now we need to call Function using Class A object:
  and this function need class B object as an parameter.*/
   Obj1.add(Obj2);

   return 0;
}
```

In the Above Example, We started with the prototype of Class B. as we know that it's important to declared class prototype because Compiler compiles the code line by line from top to bottom. so when compiler will about to compile the code line *(friend class A;)* , it will generate an error *(i.e. "A" Was not declared)* , because compiler don't know what is This "A" & where is its definition, so when we will declared an Class prototype at the top, then compiler will know about class "A". so its important.

Inline Function

A function defined in the body of a class declaration is an inline function.
Inline function is a combination of macro and function.
It is a powerful concept in C++ programming language.
This function increases the execution time of a program.
Inline is a request to the compiler. It is an optimization technique used by the compiler.
The keyword inline is used before the function name to make function inline.

Syntax:

```
inline function_name()
{
    //Function body
}
```

Example : Demonstrating the Inline function execution

```
#include <iostream.h>
inline void display()
{
    cout<<"Welcome";
}
int main()
{
    display();  // Call it like a normal function
}
```

Output:
Welcome

Where does the Inline function not work?

- Inline function does not work if the functions are recursive.
- Inline function is not used when the function contains static variables.
- Inline function does not return any value even if the return statement is exists in the function.

Advantages of Inline Function

- Inline function does not require calling function overhead.
- It makes the program faster.
- Inline function increases locality of reference by utilizing instruction cache.
- It saves overhead of return call from a function.

Disadvantages of Inline Function

- Inline function increases function size so that it may not fit in the cache and causes lots of cache miss.
- If Inline function is used in the header files, it increases the header file size and makes it unreadable.
- Inline function is not useful for embedded system where large binary size is not preferred due to memory size constraints.

Function Overloading

C++ provides new feature that is function overloading. It can be considered as an example of polymorphism feature in C++.

If two or more functions have same name but different parameters, it is said to be **Function Overloading.**
It allows you to use the same function name for different functions in the same scope/class.
It is used to enhance the readability of the program.

There are two ways to overload a function:

1. Different number of arguments.
2. Different datatypes of argument.

1. Different number of arguments

In different number of arguments, two functions have same name but different number of parameters/arguments of the same datatype.

Example: Demonstrating function overloading with different number of arguments

```
#include<iostream.h>
int add(int a, int b)
{
    cout<<a+b<<endl;
}
int add(int a, int b, int c)
{
    cout<<a+b+c<<endl;
}
int main()
{
    add(10,20);
    add(10,20,30);
}
```

Output:
30
60

In the above example, the **add()** function is overloaded with two and three arguments.

2. Different datatypes of argument

In different datatypes of argument, you can define two or more functions with same name and same number of parameters but with the different datatype of parameters/arguments.

Example : Demonstrating Function Overloading with different datatypes of argument

```
#include<iostream.h>
int add(int a, int b)
{
```

```
    cout<<a+b<<endl;
}
double add(double a, double b)
{
    cout<<a+b<<endl;
}
int main()
{
    add(10,20);
    add(10.5,20.5);
}
```

Output:

30
31

Static Members in C++

Static is a keyword in C++ used to give special characteristics to an element. Static elements are allocated storage only once in a program lifetime in static storage area. And they have a scope till the program lifetime

It can be used with data members as well as the member functions.

Static Keyword can be used with following,

- ## Static Variable inside functions:

  Static variables when used inside function are initialized only once, and then they hold there value even through function calls.
  These static variables are stored on static storage area , not in stack.

  Example:

```
#include<iostream.h>
void counter()
{
   static int count=0;
   cout << count++;
}

void main()
{
   for(int i=0;i<5;i++)
   {
      counter();
   }
```

```
}
```

Output:

```
0 1 2 3 4
```

Let's see the same program's output **without using static** variable.

Example:

```
#include<iostream.h>
void counter()
{
   int count=0;
   cout << count++;
}

void  main()
{
   for(int i=0;i<5;i++)
   {
      counter();
   }
}
```

Output:

```
0 0 0 0 0
```

If we do not use static keyword, the variable count, is reinitialized everytime when counter() function is called, and gets destroyed each time when counter() functions ends.
But, if we make it static, once initialized count will have a scope till the end of main() function and it will carry its value through function calls too.
If you don't initialize a static variable, they are by default initialized to zero.

- ## Static Data Member in class

Static data members of class are those members which are shared by all the objects. Static data member has a single piece of storage, and is not available as separate copy with each object, like other non-static data members.
Static member variables (data members) are not initialied using constructor, because these are not dependent on object initialization.
Also, it must be initialized explicitly, always outside the class. If not initialized, Linker will give error.

Example:

```
#include<iostream.h>
class X
```

```
{
   public:
   static int i;
   X()
   {
      // construtor
   };
};
int X::i=1;
void main()
{
   X obj;
   cout << obj.i;   // prints value of i
}
```

Output:

```
1
```

Once the definition for static data member is made, user cannot redefine it. Though, arithmetic operations can be performed on it.

- ## Static Member Functions in class

A function is made static by using static keyword with function name. These functions work for the class as whole rather than for a particular object of a class. It can be called using the object and the direct member access . operator. But, its more typical to call a static member function by itself, using class name and scope resolution :: operator.

Example:

```
#include<iostream.h>
class X
{
   public:
   static void f()
   {
      // statement
   }
};

void  main()
{
   X::f();   // calling member function directly with class name
}
```

These functions cannot access ordinary data members and member functions, but only static data members and static member functions.