Unit –2: Function ,Structure and Working with Object

Function

2.1 Introduction to Function

- A function is a group of statements that together perform a task.
- There are two types of function:
  1) Standard Library Functions: Predefined in C++
  2) User-defined Function: Created by users

User-defined Function

Three aspects of User-defined function:
1) Function prototype/ Function declaration
2) Function definition
3) Function call

1) Function prototype/ Function declaration

- A function declaration tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
- **Syntax:**

```
return_type    function_name( parameter list );
```

- **Example:**

```
int    max(int n1, int n2);
```

2) Function definition

- A function definition provides the actual body of the function.
- **Syntax:**

```
return_type function_name( parameter list )
{
  body of the function
}
```

- **Example:**

```
int max(int n1, int n2)
{
  int result;
```

```
    if (n1 > n2)
      result = n1;
    else
      result = n2;

    return result;
  }
```

3) Function call

- To use a function, you will have to call or invoke that function.
- When a program calls a function, program control is transferred to the called function.
- Example:

```
void main( )
{
        int a=10,b=20,c;
        c=max(a,b);          //Function call
        cout<< " Maximum = "<<c;
}
```

2.2 call by value and call by reference

call by value:

- In call by value, a copy of the variable is passed to the function.
- It passes the copies, so even if there is any change in the values inside the function, it will not reflect that change in the actual value

```
#include<iostream.h>
void exch(int a, int b);                    //function prototype
void main()
{
   int x,y;
   cout<<"Enter two numbers:"<<endl;
   cin>>x>>y;
   exch(x,y);
   getch( );
}
void exch(int a, int b)
{
   int temp;
   temp=a;
   a=b;
   b=temp;
```

```
    cout<<"After exchange:"<<endl;
    cout<<a<<" "<<b;
}
```

Enter two numbers:
3 6
After exchange:
6 3

Call By Reference:

- The c++ provides easy and effective solution using the reference variables.
- When function is called, it creates a reference for each argument and using references the actual arguments are accessed.
- This method of passing the arguments or parameters to the function is called call by reference.
- The call by reference function is written as

```
int exch (int& a, int& b)
{
    int temp;
    temp= a;
    a =b;
    b=temp;
 }
```

- Note that the function prototype should also mention the type of arguments as reference when argument are to be passed as reference. The prototype of above function is as follows.

```
void exch(int&,int&);              //function prototype with reference arguments
```

- When this function is called as

```
exch(x,y);                  //function call
```

It passes the arguments as
int &a=x; and
int &b=y;

- Which makes a alias of x and b alias of y and hence while function body exchanges a and b, it is same as exchanging x and y.

- **Example:**

```
#include<iostream.h>
void exch(int&, int&);                  //function prototype
```

```
void main( )
{
    int x,y;
    cout<<"Enter two numbers:"<<endl;
    cin>>x>>y;
    exch(x,y);
    cout<<"After exchange:"<<endl;
    cout<<x<<" "<<y;
}
void exch(int& a,int& b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}
```

```
Enter two numbers:
3  6
After exchange:
6  3
```

2.3 Returning values from function

- A return statement ends the execution of a function, and returns control to the calling function.
- A return statement can return a value to the calling function.
- Example:

```
#include<iostream.h>
int sum(int a, int b);                    //function prototype
void main()
{
        int x,y,s;
        cout<<"Enter two numbers:"<<endl;
        cin>>x>>y;
        s=sum(x,y);
        cout<<"Sum= :"<< s;
}
int sum(int a, int b)
{
        int c;
        c=a+b;
        return c;
}
```

2.4 Overloaded function:-different number of arguments, different kinds of arguments

- A program needs to compute the area of circle and area of rectangle.
- We can do is write two functions with names circle_area() and rect_area() and call circle_area whenever we want to compute area of circle and rect_area() whenever we want to compute area of rectangle.
- Programmer needs to remember separate names for each function and see the prototype each time whenever function call is required.
- As C does not allow same names to multiple functions , there is no way except to give separate names.
- **C++ solves this problem by allowing multiple function to have same names. This is known as function overloading.**
- For above example of area, we can write two functions with name area, but with different implementation.
- Complier differentiates it by number and type of the arguments. The area functions for above example is defined as

```
float area(float r);              //area of circle
float area(float l, float b);     //area of rectangle
```

- **Example:**

```
#include<iostream.h>
using namespace std;
const float pie=3.14;
float area(float r)  //function to compute area of circle
{
        return(pie*r*r);
}
float area(float 1,float b)   //function to compute area  of rectangle
{
        return(l*b);
}
void main()
{
        float  radius;
        cout<<"Enter radius:";
        cin>>radius;
        cout<<"Area of circle="<<area(radius)<<endl;
        float length, breadth;
        cout<<"Enter length and breadth:";
        cin>>length>>breadth;
        cout<<"Area of rectangle="<<area(length, breadth)<<endl;
}
```

Output:

```
Enter radius:3.4
```

```
Area of circle:36.298404
Enter length and breadth: 3    4
Area of rectangle:12.000000
```

2.5 Inline function

- Functions provide the flexibility to use the same code many places in a program.
- This reduces the size of a program as well as reduces memory requirements at run time.
- This advantage becomes overhead if the function needs time to call (passing arguments and transferring control) and return (returning value and control back).
- C++ goes one step ahead and provides **inline function** which are capable of acting as both macros as well as functions.
- The **syntax** for defining inline function is as follows.

```
inline return_type function_name(arguements)
{
        //body of the function
}
```

- The word inline is reserved word and preceding function header by it makes function inline.
- The word inline requests the compiler to make the function inline means expand every call to this function by its body i.e. definition.

- **Example:**

```
#include<iostream.h>
using namespace std;
const float pie=3.14;
inline float area(float r)
{
        return (pie*r*r);
}
void main()
{
        float radius;
        cout<<"Enter radius:";
        cin>>radius;
        cout<<"Area="<<area(radius)<<endl;
}
```

2.6 Default arguments

- Function defined with three arguments can not be called with one or two arguments in C, but should be called with always three arguments.
- C++ allows calling of function with less number of arguments then listed in its header.
- C++ makes it possible by allowing default arguments i.e. arguments having default values.

- Whenever function is called and value for such argument is not passed, compiler uses the default value.

- **Example:**

```
#include<iostream>
using namespace std;
void set_point(int x,int y=0); //default arguments
void main()
{
        int p,q;
        cout<<"Enter x coordinate:";
        cin>>p;
        set_point(p);
        cout<<"Enter x and y coordinates:"
        cin>>p>>q;
        set_point(p,q);
}
        //function to set the point
void set_point(int x,int y)
{
        cout<<"("<<x<<","<<y<<")"<<endl;
}
```

Output:

```
Enter x coordinate:5
(5,0)
Enter x and y coordinates:6 8
(6,8)
```

Structure

2.7 A simple Structure

- Structure is a collection of variables of different data types under a single name.
- The struct keyword defines a structure type followed by an identifier (name of the structure).
- When a structure is created, no memory is allocated.

- **Example:**

```
struct person
{
    char name[50];
    int age;
    float salary;
};
```

Here a structure name is person
It has three members: name, age and salary.

2.8 Defining structure

- Structure can be declared before or after the main ().

- **Syntax:**

```
struct     Structure_Name
{
        datatype data_member1;
        data_type data_member2;
        - - - - - - - - - - - - - -
        data_type member N;
} struct_var1,struct_var2,… ;
```

Where ,
struct - keyword.
structure_Name - the name of structure
data_member1,data_member2 ,.. - the variables of different data types.
struct_var1, struct_var2,… - the variables of structure name type.

- **Example:**

```
struct student
 {
        int rollno;
        char name [50];
         int marks;
}s1,s2;
```

2.9 Defining structure variable

- Structure variables can be declared by two way
  1) with structure declaration
  2) as a separate declaration inside or outside main( ).
- Data member can be accessed using structure variable.

<u>with structure declaration</u>

- Example:

```
struct student
 {
```

```
        int rollno;
        char name[50];
        int marks;
    }s1,s2;
```

Here, s1 and s2 are structure variable.

as a separate declaration inside or outside main( )

- **Example:**

```
struct student
 {
        int rollno;
        char name [50];
        int marks;
};
void main( )
{
        struct student s1,s2;
}
```

2.10 Accessing structure members

- To access any member of a structure, we use the member access operator (.).
- The member access operator is coded as a dot between the structure variable name and the structure member that we want to access.
- **Syntax:**

  structure_variable.member_name

- **Example:**

```
struct student
 {
        int rollno;
        char name [50];
        int marks;
};
void main( )
{
        struct student s1,s2;
        s1.rollno=1;                    // accessing member rollno
        s1.name="ABC";                  // accessing member name
        s1.marks=60;                    // accessing member marks
}
```

2.11 Initializing structure member

- Initialization is the assignment of an initial value for a member of structure.

- **Syntax:**

```
structure_variable.member_name = value;
```

- **Example:**

```
struct student
 {
        int rollno;
        char name [50];
         int marks;
};
void main( )
{
         struct student s1,s2;
         s1.rollno=1;
         s1.name=”ABC”;
         s1.marks=60;
}
```

Class and Object

2.12 Introduction to class and object

❖ Class

- Class is a collection of objects.
- In class we can bind the **data** (variable) and its associated **functions** together.
- It is called data encapsulation.
- Class allows the data and function to be hidden if necessary.
- When we create a class it treat as a built in data type.
- So, we can create the variable of that data-type, which is called as objects.
- A class specification has generally two parts:
  - Class declaration
  - Class function definitions
- The class declaration describes the type and scope of its members.
- The class function definition describes how the class functions are implemented.
- To declare the class we have to use the **class** keyword.

❖ Objects:

- Object is a basic run-time entity in object oriented system.
- Object is a one type of class variable.
- Object can be declared as same as the variable declaration.

2.13 Declaration of class and object

Declaration of class

- The class declaration describes the type and scope of its members.
- To declare the class we have to use the **class** keyword.

Syntax of class declaration:

```
class class-name
{
        private:
                variable declarations;
                function declarations;
        public:
                variable declarations;
                function declarations;
};
```

- The variable which is declared inside the class declaration that is known as **data members.**
- The functions which is declared inside the class declarations that is known as **member functions.**
- The data member and member functions are also known as the class members.
- **Private** and **public** specifies the scope (where it is used) of the class members.
- By default, the scope of the class members is private.
- The class members which are declared as private can be accessed only from within the same class.
- The class members which are declared as public can be accessed from outside the class also.
- Only the member function of the class can access the private data member and the private member functions of the same class.

- **Example of class declaration:**

```
class student
{
        int no;
        char name[20];
public:
        void getdata( );
        void putdata( );
};
```

Here, no and name are the data members which has the private scope.
And getdata( ), putdata( ) are the member functions which has the public scope.

Declaration of object:

- Objects can be declared by two way
    1) with class declaration

2) as a separate declaration inside or outside main( ).

1) with class declaration

- Objects can be created when a class is defined by placing their names immediately after the closing brace, same as structures.
- **Example**:

```
class student
{
        . . . . .
        . . . . .
}s1,s2,s3;
```

Where s1,s2 and s3 are the object name.

2)as a separate declaration inside or outside main( ).

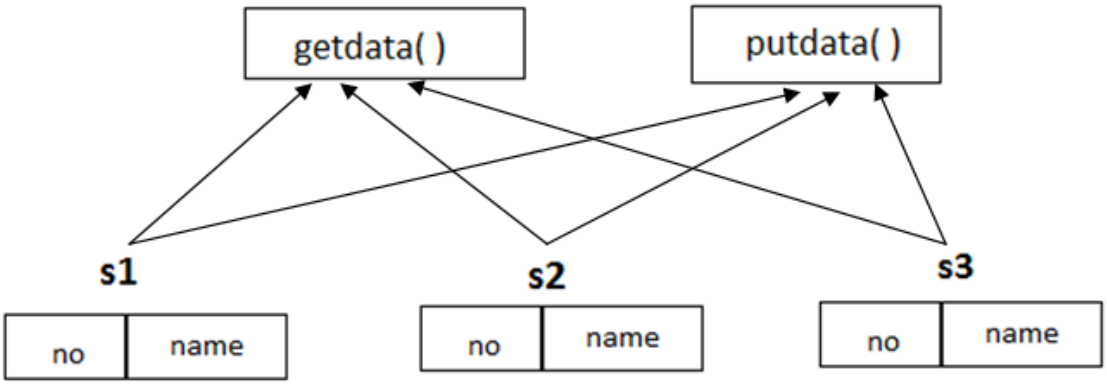
- Objects can be created after declaration of class.
- It can be declared inside main( ) function.
- **Syntax:**

```
class-name object-name;
```

- **Example :**

```
student   s1,s2,s3;
```

where, student is the class name and s1,s2 and s3 are the object name.



2.14 Access Specifier-Private, public and protected

- Access specifiers define how the members (variable or function) of a class can be accessed.
- In C++, there are three access specifiers:
  - public - members are accessible from outside the class
  - private - members cannot be accessed (or viewed) from outside the class
  - protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes.
- By default, all members of a class are private if you don't specify an access specifier:

- **Example:**

```
class student
{
        int no;                 //private specifier
        char name[20];
    public:

        void getdata( )         // public specifier
        {
            cout<< "Enter Numnber and Name :";
            cin>>no>>name;
        }
        void putdata( )         // public specifier
        {
            cout<< "Number : "<<no<<" \nName :"<<name;
        }

};
void main( )
{
    student s1;
    s1.getdata( );
    s1.putdata( );
}
```

2.15 Defining member function inside

- Member function definitions can be defined in two places:
  - Outside the class definition
  - Inside the class definition
- **Inside the class definition:**
  - Whenever we replace the function declaration by the actual function definition inside the class then it is called the member function definition inside the class.
  - **Example**:

```
class student
{
        int no;
        char name[20];
    public:
        void getdata( );
        void putdata( )   // definition inside class
        {
            cout<< "Number : "<<no<<" Name :"<<name;
        }
};
```

2.16 Defining member function outside of the class using scope resolution operator

- Member function definitions can be defined in two places:
    1. Outside the class definition
    2. Inside the class definition

Outside the class definition

- Member function can be declared outside the class, if it is only declared inside class.
- Definition of member function are like normal function
- Whenever the definition of class member function outside the class, the member name must be qualified by the class name using the scope resolution operator.
- **Syntax:**

```
return_type     class_name::function_name(Arguments)
{
        Statements;
}
```

- **Example**:

```
class student
{
                int no;
                char name[20];
        public:
                void getdata( );
                void putdata( )   // definition inside class
                {
                        cout<< "Number : "<<no<<" Name :"<<name;
                }
};
void student::getdata( )
{
                cout<<"Enter Number: ";
                cin>>no;
                cout<<"Enter Name: ";
                cin>>name;
}
```

2.17 private member function

- A function declared inside the private access specifier of the class, is known as a private member function.
- Normally we define data members as private and function member as public.
- But in some cases we require to declare function as private.

- That private member function is allowed to access within class only.
- Here in below example, swap function is used from inside the class only so define that function as private.

- **Example:**

```
class array
{
  private:
        int a[5];
        void swap(int i,int j)          //Private function
        {
              int temp;
              temp=a[i];
              a[i]=a[j];
              a[j]=temp;
        }
  Public:
        void getarray( )
        {
              for(int i=0,i<5;i++)
                    cin>>a[i];
        }
        void sort( )
        {
              for(int i=0,i<4;i++)
              {
                    for(int j=i+1;j<5;j++)
                    {
                          if(a[i]>a[j])
                                swap(i,j);
                    }
              }
        }
        void putarray( )
        {
              for(int i=0,i<5;i++)
                    cout<<a[i];
        }
};
  void main( )
  {
        array a1;
        a1.getarray( );
        a1.sort( );
        a1.putarray( );
  }
```

2.18 outside member function as inline

- C++ provides inline functions to reduce the function call overhead.
- We can define a member function outside the class definition and still make it inline by just using the qualifier inline in the header line of the function definition.
- Example:

```
class item
{
        .....
        .....
    public:
            void getdata(int a,float b);
};
inline void item :: getdata(int a,float b)
{
         number=a;
        cost=b;
}
```

2.19 Static member and member function

Static Data member

- A data member of a class can be qualified as static.
- The properties of a static member variable are similar to that of a static variable.
- **Characteristic of a static data members:**
  - It is initialized to zero when the first object of its class is created. No other initialization is permitted.
  - Only one copy of that member is created for the entire class and it is shared by all the objects of that class.
- It is visible only within the class, but its lifetime is the entire program.
- Static variables are normally used to maintain values common to the entire class.
- The definition of the static data member is written outside the class.

- **Syntax of definition of static data member:**

data-type class-name:: variable-name= value;

where the value is optional.

- **Example:**

int item :: count;

where, item is the class-name and count is the static data member.

- **Example of static data member:**

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

```
#include<conio.h>
class item
{
        int code;
        static int count;
public:

        ---------
        ---------
};
int item::count;
void main()
{

        ---------
        ---------

}
```

Static Member Functions

- The member function of the class which is declared with the static keyword, it is called as static member functions.

- **Static member has the following characteristics:**
  - A static member function can have access to only other static members declared in the same class. Static members can be either data member or member function.
  - A static member function can be called using the class name as follows:
    class-name :: function-name;

- **Example:**

```
#include<iostream.h>
#include<conio.h>
class test
{
        int code;
        static int count;
        public:
                void setcode( );
                void showcode( );
                static void showcount( )
                {
                        cout<<"Count : "<<count<<endl;
```

```
        }
};
int test::count;
void main( )
{
        --------
        --------
        test::showcount( );
        --------
        --------
        test::showcount( );
        --------
        --------
}
```

2.20 array of object

- An object of class represents a single record in memory, if we want more than one record of class type, we have to create an array of class or object.
- An array is a collection of similar type, therefore an array can be a collection of class type.
- An array of objects is declared in the same way as an array of any built-in data type
- **Syntax:**

```
class_name  array_name [size] ;
```

- **Example:**

```
class books
{
        char  tit1e[30];
        float  price ;
public:
        void getdata ()
        {
                cout<<"Title:";
                cin>>title;
                cout<<"Price:";
                cin>>price;
        }
        void putdata ()
        {
                cout<<"Title:"<<title<< "\n";
                cout<<"Price:"<<price<< "\n";
        }
} ;
```

```
void  main ()
{
        books b[3] ;
        for(int i=0;i<3;i++)
        {
                cout<<"Enter details o£ book "<<(i+1)<<"\n";
                b[i].getdata();
        }
        for(int i=0;i<3;i++)
        {
                cout<<"\nBook "<<(i+l)<<"\n";
                b[i].putdata() ;
        }
        getch( );
}
```

2.21 Object as a function argument

- 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.
- Whenever an object of a class is passed to a member function of the same class, its data members can be accessed inside the function using the object name and the dot operator.

- **Example:**

C++ program to add two complex numbers by passing objects to a function.

```
#include <iostream>
using namespace std;

class Complex
{
   private:
        int real;
        int imag;
   public:
         void readData()
        {
                cout << "Enter real and imaginary number respectively:"<<endl;
                cin >> real >> imag;
        }
        void addComplexNumbers(Complex comp1, Complex comp2)
        {
                real=comp1.real+comp2.real;
                imag=comp1.imag+comp2.imag;
        }
```

```
        void displaySum()
        {
                cout << "Sum = " << real<< "+" << imag << "i";
        }
};
void  main()
{
        Complex c1,c2,c3;
        c1.readData();
        c2.readData();
        c3.addComplexNumbers(c1, c2);          // Object c1,c2 as a function argument
        c3.displaySum();
        getch( );
}
```

2.22 friend function

- To make an outside function friendly to a class. We have to declare this function as a friend of the class.
- The function declaration should be preceded by the keyword friend.
- The functions that are declared with the keyword friend are known as friend function.
- The function is defined elsewhere in the program like a normal C++ function.

- **Characteristics of the Friend Function:**
  - It is not in the class to which it has been declared as friend.
  - Since it is not in the scope of the class, it cannot be called using the object of that class.
  - It can be invoked like a normal function without the help of the object.
  - It cannot access the member names directly and has to use an object name and membership operator with each member name.
  - It can be declared either in the public or private part of a class.
  - It has the objects as arguments.

- **Example:**

```
#include<iostream.h>
#include<conio.h>
class ABC;              //Forward declaration of class
class XYZ
{
        int a;
public:
        void setvalue(int x)
        {
                a=x;
        }
```

```
        friend void max(ABC,XYZ);
};
class ABC
{
        int b;
public:
        void setvalue(int y)
        {
                b=y;
        }
        friend void max(ABC,XYZ);
};
void max(ABC p,XYZ q)
{
        if(p.b>q.a)
        {
                cout<<"b is max";
        }
        else
        {
                cout<<"a is max";
        }
}
void main()
{
        ABC p;
        XYZ q;
        clrscr();
        p.setvalue(20);
        q.setvalue(10);
        max(p,q);
        getch();
}
```

Output of the Program:

```
b is max
```

2.23 returning object

- A function can 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.

- Example:

```
#include <iostream>
using namespace std;
class  Example
{
   private:
          int temp;
   public:
           void readData(int i)
          {
                  temp=i;
          }

          Example  copy( Example    obj)
          {
                  return obj;                    //Return object
          }
  };
  void main( )
  {
          Example e1,e2,e3;
          e1.readData(10);
          e2.readData(20);
          e3=e1.copy(e2);
  }
```

2.24 friend class

- A friend class can access both private and protected members of the class in which it has been declared as friend.
- We can also use a friend Class in C++ using the friend keyword.

- **Syntax:**

```
friend   class   ClassName;
```

- **Example:**

```
#include <iostream>
using namespace std;
class A
{
        int x =5;
```

```
        friend class B;          // friend class.
};
class B
{
public:
        void display(A &a)
        {
                cout<<" x is : "<<a.x;
        }
};

void main()
{
        A a;
        B b;
        b.display(a);
        return 0;
}
```

- **Output:**

```
X is: 5
```

- When a class is declared a friend class, all the member functions of the friend class become friend functions.
- Since Class B is a friend class, we can access all members of Class A from inside Class B.
- However, we cannot access members of Class B from inside Class A. It is because friend relation in C++ is only granted, not taken.