Module - 32

Constructors and Destructor

Table of Contents

Learning outcome –

After studying this module, you will be able to:

1. Study about Constructors
2. Learn about Different types of Constructors
3. Understand about Multiple Constructors in a class
4. Get familiar with Dynamic Initialization of Objects
5. Study about constructing two dimensional arrays
6. Learn about Dynamic Constructors

1. Introduction

C++ programming language offers a special member function called constructor which allows an object to initialize itself when it is created. It also provides one more special member function called destructor that is called when an object of the class goes out of scope, or whenever the objects are no longer required or when the memory space used by it is de-allocated with the help of delete operator.

2. Constructors

A constructor is a special member function whose main task is to initialize the objects of its class. It is considered as a special member function because it has the same name as that of the class name itself. The constructor is called upon whenever an object of its related class is created. A constructor constructs the values of data members of the class. C++ constructor is called only once in a lifetime of object when object is created. C++ constructors can be overloaded. C++ constructors do not return any value so constructor has no return type not even void data type.

Syntax:

```
class class_name
{
private:
//….
public:
class_name ();    //constructor
};
```

The constructor functions have some special characteristics, they are:

a) They are supposed to be declared only in the public section.

b) They are call upon automatically when the objects are created.

c) They do not have any return type, i.e., not even void, therefore, they cannot return any values.

d) They cannot be inherited, even though a derived class can call a base class constructor.

e) They can have default arguments as that of other C++ functions.

f) They cannot be virtual.

g) They cannot refer to their addresses.

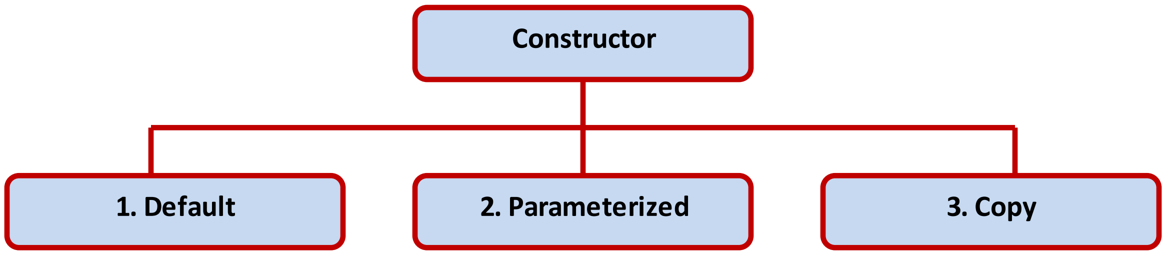
h) They make implicit calls to the new and delete operators when there is a requirement for memory allocation.

i) An object with a constructor cannot be used as a member of union.

Note: When a constructor is declared for a class, then it is must to initialize the objects.

3. Different types of Constructors

The three different types of constructors that are supported by C++ programming language are: Default Constructor, Parameterized Constructor and Copy Constructor.



3.1 Default Constructor

A constructor that accepts no parameters is called as default constructor. If no constructors are available for a class, then the compiler implicitly creates a default parameter less constructor without a constructor initializer and a null body. The initialization of the class objects becomes compulsory, when a constructor is declared for a class.

Syntax:

```
class_name ()
{
        Constructor Definition
};
```

Example:

The default constructor for class emp is emp :: emp(). If no such constructor is defined, then the compiler supplies a default constructor. The statement emp e1; invokes the default constructor of the compiler to create an object by name e1 of the class emp.

Program to implement the usage of default constructor

```
# include <iostream.h>
using namespace std;
class plot
{
        float plot_size;
        public :
        plot()
        {
                plot_size = 2400.50;
        }
        void disp_plot()
        {
                cout<<"The size of the plot is"<<plot_size<<endl;
        }
};
int main()
{
        plot p1;
        p1.disp_plot();
        return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
The size of the plot is 2400.50
```

Program to illustrate the usage of default constructor

```
#include<iostream>
using namespace std;
class marks
{
        public:
          int maths;
          int science;

          marks()
          {
                    maths=0;
                    science=0;
          }

         void display()
         {
                    cout << "Maths :  " << maths <<endl;
                    cout << "Science :" << science << endl;
         }
};

int main()
{
  marks m1;
  m1.display();
  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Maths :  0
Science : 0
```

Program to add sum of two numbers and display the result using default constructor

```
#include<iostream>
using namespace std;
class sum
{
    int a,b;
public:
    sum()
    {
        a=10;
        b=50;
        cout<< "Sum = "<<a+b;
    }
};
int main()
{
    sum t;
}
```

When the above code is compiled and executed, it produces the following result:

```
Sum = 60
```

3.2 Parameterized Constructors

In practice, it may be compulsory to initialize a variety of data elements of different objects with different values when they are created. C++ permits to accomplish this purpose by passing arguments to the constructor function when the objects are created. The constructor that can take arguments or parameters is called as parameterized constructor or argument constructor. When some arguments are passed to the constructor, then it will automatically pass the arguments to the constructor and the values will be retrieved by the relevant data members of the class.

Syntax:

```
class_name(Argument_List)
{
-----
-----
}
```

Example:

```
class integer
{
int m, n;
public:
integer(int x, int y); //parameterized constructor
};
integer :: integer(int x, int y)
{
m=x; n=y;
}
```

When an object is declared using the parameterized constructor, it is must to pass the initial values as arguments to the constructor function. It can be done in two ways, they are:

By calling the constructor explicitly

By calling the constructor implicitly

The explicit declaration can be as follows:
integer i1 = integer (0, 100);

The implicit declaration is called as the shorthand method and is used frequently to implement and will be as follows:
integer i1(0,100);

It is also possible to define constructor functions as inline functions.

Example:

```
class integer
{
int m,n;
public:
integer(int x, int y) //inline constructor
{
m=x; n=y;
}
};
```

Note: The parameters of a constructor can be of any type except that of the class to which it belongs.

Program to implement the usage of parameterized constructor

```
#include <iostream.h>
using namespace std;
class smple
{
        int a;
        public:

        sample(int j)
        {
        a = j;
       }

int geta()
       {
        return a;
        }
};
int main()
{
        Sample s1 = 99;
        cout << s1.geta();
        return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
99
```

Program to implement the usage of Parameterized Constructor to print the subject marks

```
#include<iostream>
using namespace std;
class Marks
{
   public:
   int maths;
   int science;

   Marks(int mark1,int mark2)
  {
      maths = mark1;
      science = mark2;
   }

  void display ()
  {
      cout << "Maths :  " << maths <<endl;
      cout << "Science :" << science << endl;
  }
};

int main()
{
  Marks m(90,85);
  m.display();
  return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Maths :  90
Science : 85
```

Program to illustrate the usage of parameterized constructor

```
# include <iostream.h>
# include <string.h>
using namespace std;
class course
{
int coursecode;
char coursename[30];
public :
course(int, char *);
void dispdata()
{
cout<<"Course code "<<coursecode<<endl;
cout<<"Course Name "<<coursename<<endl;
}
};
course :: course(int code, char *s)
{
coursecode = code;
strcpy(coursename, s);
}
int main()
{
course c(500, "C++");
c.dispdata();
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Course code 500
Course Name C++
```

Program to calculate area and volume using parameterized construtor

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

class av
{
   int a,b,l;
public:
   av()
   {
      l=10;
      b=20;
      a=l*b;
      cout<< "Area = "<<a<<endl;
   }
   av(int s)
   {
      int v;
      v=s*s*s;
      cout<< "Volume = "<<v<<endl;
```

```
    }
    ~av()
    {
        cout<< "Destructor invoked"<<endl;
    }
};
int main()
{
    av t;
    int s;
    s=10;
    av t1(s);
}
```

When the above code is compiled and executed, it produces the following result:

```
Area = 200
Volume = 1000
```

Program to calculate power of a number using parameterized constructor

```
#include <iostream.h>
using namespace std;
class power
{
double b;
int e;
double val;
public:
power(double base, int exp);
double get_power()
{
return val;
}
};

power::power(double base, int exp)
{
b = base;
e = exp;
val = 1;
if(exp==0)
return;
for( ; exp>0; exp--)
val = val * b;
}

int main()
{
power x(4.0, 2), y(2.5, 1), z(5.7, 0);
cout << x.get_power() << " ";
cout << y.get_power() << " ";
cout << z.get_power() << "\n";
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
16 2.15 1
```

Program to calculate cube of a number using parameterized constructor

```
#include <iostream.h>
using namespace std;
class cube
{
int x, y, z;
public:
cube(int i=0, int j=0, int k=0)
{
x=i;
y=j;
z=k;
}
int volume()
{
return x*y*z;
}
};

int main()
{
cube a(2,3,4), b;
cout << a.volume() << endl;
cout << b.volume();
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
24 0
```

3. 3 Copy Constructor

The copy constructor is a type of constructor which is used to create a copy of an already existing object of a class type. A copy constructor takes a reference to an object of the same as itself as an argument. All member values of one object can be assigned to the other object using copy constructor. For copying the object values, it is must that both the objects must belong to same class. The copy constructor can accept a reference to its own class as a parameter. The process of initializing through a copy constructor is known as copy initialization. The compiler provides a default Copy Constructor to all the classes. It is must to use & (Ampersand) or Address Operator when an object is used or passed to a constructor function. It is usually of the form X (X&), where X is the class name.

Syntax:

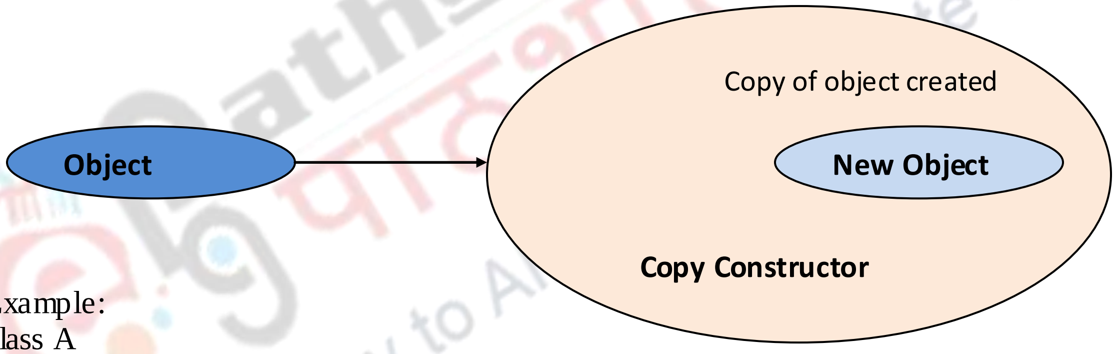
```
class_name :: class_name (class_name &ptr)
```

The Copy Constructor can also be defined using a const keyword.

Syntax:

```
class_name :: class_name (const class_name & ptr)
```

Copy constructor can be represented in the figure as follows:



Example:

```
class A
{
………
public:
A(A &);
};
```

The situations where copy constructor is called:

- ❖ When a new object is initialized to an object of the same class.
- ❖ When an object is passed to a function by value.
- ❖ When an object is returned from a function by value.
- ❖ When the compiler generates a temporary object.

The copy constructor is used to:

- ➢ Initialize one object from another of the same type.
- ➢ Copy an object to pass it as an argument to a function.
- ➢ Copy an object to return it from a function.

Program to illustrate the use of Copy Constructor

```
#include<iostream>
using namespace std;
class Example
{
    int a,b;
    public:
    Example(int x,int y)
    {
    cout<<"\nIn Constructor";
    a=x;
    b=y;
    }

   Example(Example &e)
  {
     a = e.a
     b=e.b;
   }
    void Display()
    {
    cout<<"\nValues :"<<a<<"\t"<<b;
    }
};

int main()
{
        Example Object(10,20);
        Example Object2=Object;
        Object.Display();
        Object2.Display();
        return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
In Constructor
Values :10      20
Values :10      20
```

Program to find the factorial of a number using Copy Constructor

```
#include<iostream.h>
using namespace std;
class copy
{
              int var,fact;
              public:
```

```
            copy(int temp)
            {
             var = temp;
            }

             double calculate()
            {
                    fact=1;
                    for(int i=1;i<=var;i++)
                    {
                    fact = fact * i;
                    }
                    return fact;
            }
};
int main()
{
    int n;
   cout<<"\n\tEnter the Number : ";
   cin>>n;
   copy obj(n);
   copy cpy=obj;
   cout<<"\n\t"<<n<<" Factorial is:"<<obj.calculate();
   cout<<"\n\t"<<n<<" Factorial is:"<<cpy.calculate();
   return 0;
}
```

When the above program is compiled and executed, it produces the following result:

```
Enter the Number: 5
Factorial is: 120
Factorial is: 120
```

Program to display marks using copy constructor

```
#include<iostream>
using namespace std;
class marks
{
public:
   int maths;
   int science;
   marks()
  {
     maths=0;
     science=0;
   }
```

```
    marks(const marks &obj)
   {
      maths=obj.maths;
      science=obj.science;
    }
    void display()
   {
      cout<<"Maths :   " << maths
      cout<<"Science : " << science;
    }
};

int main()
{
    marks m1;

    marks m2(const marks &m1);

    m2.display();

    return 0;
}
```

Output

```
Maths :  0
Science : 0
```

Program to print the contents of point using copy constructor

```
#include<iostream>
using namespace std;
class Point
{
private:
    int x, y;
public:
     Point(int x1, int y1)
     {
     x = x1;
     y = y1;
     }

    Point(const Point &p2)
    {
    x = p2.x;
    y = p2.y;
    }
```

```
   int getX()
  {
       return x;
  }
   int getY()
   {
       return y;
   }
};

int main()
{
   Point p1(10, 15); // Normal constructor
   Point p2 = p1; // Copy constructor
   cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
   cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();
    return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
p1.x = 10, p1.y = 15
p2.x = 10, p2.y = 15
```

Note: A reference variable can be used as an argument to the copy constructor, but it is not possible to pass the argument by value to a copy constructor.

4. Multiple Constructors in a class (Overloaded Constructors)

The constructors can also be overloaded like other member functions. The constructor of a class may also be overloaded so that even with different number and types of initial values, an object may still be initialized like any other function. When more than one type of constructor is defined in the class, then it is said to be Overloaded Constructors. Any number of Constructors cane exists in a class, but, it is compulsory that they should differ in the parameter list. Depending upon the number of parameters, the compiler executes appropriate constructor. The overloading of constructors is used to increase the flexibility of a class by having more number of constructors for a single class.

Program to implement the usage of multiple constructors in a class

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

class Example
{
   int a,b;
   public:
```

```
   Example()
  {
   a=50;
   b=100;
   cout<<"\n Default Constructor";
   }


   Example(int x,int y)
  {
   a=x;
   b=y;
   cout<<"\n Parameterized Constructor";
   }

   void Display()
  {
   cout<<"\nValues :"<<a<<"\t"<<b;
   }
};

int main()
{
     Example Object(10,20);
     Example Object2;
     Object.Display();
     Object2.Display();
     return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Parameterized Constructor
Default Constructor
Values :10      20
Values :50      100
```

Program to display the rollno and name using multiple constructors in a class

```
#include<iostream>
using namespace std;
class Student
{
 int rollno;
 char name[50];
 public:
 Student(int x)
 {
```

```
 rollno=x;
 name="";
}
Student(int x, char str[])
{
 rollno=x ;
 strcpy(name,str);
}

  void display()
  {
       cout<<"rollno is "<<rollno<<" name is "<<name<<endl;
}
};

int main()
{
 Student S1(10);
 Student S2(11,"AAA");
 S1.display();
 S2.display();
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
10
11 AAA
```

Program to illustrate the usage of multiple constructors in a class

```
#include <iostream.h>
using namespace std;
class point
{
public:
int x;
int y;
point()
{
x = y = 0;
}

point(int a)
{
x = y = a;
}
point(int a, int b)
{
x = a; y = b;
}
};

int main()
{
point p1;
point p2(4);
point p3(8, 12);
cout << "Point p1's x,y value:: " <<p1.x << " , "<< p1.y << "\n";
cout << "p2's x,y value:: "<<p2.x << " ,"<< p2.y << "\n";
cout << "p3's x,y value:; "<<p3.x << " , "<< p3.y << "\n";
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Point p1's x,y value:: 0 , 0
p2's x,y value:: 4 ,4
p3's x,y value:; 8 , 12
```

Program to calculate interest using multiple constructors in a class

```
#include<iostream.h>
#include<stdlib.h>
using namespace std;
class INTEREST
{
        float principal,rate,totalamount;
        int time;

        public:
                INTEREST();
                INTEREST(float p, int t, float r);
                INTEREST(float p, int t);
                INTEREST(float p, float r);
                void calculateamount();
                void display();
};
INTEREST::INTEREST()
{
        principal = 0.0;
      time = 0;
      rate=0.0;
}
INTEREST::INTEREST(float p, int t, float r)
{
        principal = p;
        time = t;
        rate = r;
}
INTEREST::INTEREST(float p, int t)
{
        principal = p;
        time = t;
        rate = 0.08;
}
INTEREST::INTEREST(float p, float r)
{
        principal = p;
        time = 2;
        rate = r;
}
void INTEREST::calculateamount()
{
        totalamount = principal + (principal*time*rate)/100;
}
void INTEREST::display()
{
        cout<<"Principal Amount: Rs."<<principal<<"\n";
        cout<<"Period of investment: "<<time<<" years\n";
        cout<<"Rate of interest: "<<rate<<"\n";
        cout<<"Total Amount: Rs."<<totalamount<<"\n";
}
int main()
{

        INTEREST d1;
        INTEREST d2(2000, 2, 0.07f);
        INTEREST d3(4000, 1);
        INTEREST d4(3000, 0.12f);
        d1.calculateamount();
        d2.calculateamount();
        d3.calculateamount();
        d4.calculateamount();
        cout<<"Object 1\n";
        d1.display();
        cout<<"\nObject 2\n";
        d2.display();
        cout<<"\nObject 3\n";
        d3.display();
        cout<<"\nObject 4\n";
        d4.display();
```

```
	return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Object 1
Principal Amount: Rs.0
Period of investment: 0 years
Rate of interest: 0
Total Amount: Rs.0

Object 2
Principal Amount: Rs.2000
Period of investment: 2 years
Rate of interest: 0.07
Total Amount: Rs.2002.8

Object 3
Principal Amount: Rs.4000
Period of investment: 1 years
Rate of interest: 0.08
Total Amount: Rs.4003.2

Object 4
Principal Amount: Rs.3000
Period of investment: 2 years
Rate of interest: 0.12
Total Amount: Rs.3007.2
```

5. Dynamic Initialization of Objects

The objects of the class can be initialized dynamically, i.e., the primary value of the object may be provided during run time. The main advantage of dynamic initialization is that it is possible to provide initialization formats by using overloaded constructors, which tries to give the flexibility so that different format of data can be used at runtime based on the situation.

6. Dynamic Constructors

The constructor that is used to allocate the memory to the objects at the run time is called as dynamic constructor. The memory allocation to objects is allocated with the help of 'new' operator. It is possible to dynamically initialize the objects with the help of dynamic constructor.

Program to implement the usage of dynamic constructor

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

class Account
{
```

```
private:

int account_no;
int balance;

public :

Account(int a,int b)
{
account_no=a;
balance=b;
}

void display()
{
cout<< "\nAccount number is : "<< account_no;
cout<< "\nBalance is : " << balance;
}
};

int main()
{
int ac_no,bal;

cout<< "Enter account no : ";
ac_no=101;

cout<< "\nEnter balance : ";
bal=12000;
Account *acc=new Account(ac_no,bal); //dynamic constructor
acc->display();

return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Account number is : 101
Balance is : 12000
```

Program to illustrate the usage of dynamic constructor

```
#include <iostream.h>
class dyncons
{
 int * p;
 public:
 dyncons()
 {
  p=new int;
  *p=10;
 }
 dyncons(int v)
 {
  p=new int;
  *p=v;
 }
 int dis()
 { return(*p);
 }
};

int main()
{
dyncons o, o1(9);
cout<<"The value of object o's p is:";
cout<<o.dis();
cout<<"\nThe value of object 01's p is:"<<o1.dis();
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
The value of object o's p is:10
The value of object o1's p is:9
```

7. Destructor

The job of memory de-allocation from object is done by special member function of the class called as destructor. The most common use of destructor is to de-allocate and release memory that was allocated for the object by the constructor. It is a member function whose name is the same as the class name but is preceded by tilde (~) symbol. A class cannot have more than one destructor. It takes no arguments and no return types can be specified for it, not even void. It is called automatically by the compiler when an object is destroyed or when the object goes out of scope. Destructors are parameter less functions.

Syntax:

```
        class class_name
        {
        private:
        //….
        public:
        ~class_name ();    //destructor
        };
```

Characteristics of Destructor

- It never takes any argument nor does it return any value.
- Only one destructor can be defined for a class.
- It cannot be overloaded.
- It is must that it should be defined in the public section of the class.

Program to implement the concept of constructor and destructor

```
#include<iostream>
using namespace std;
class Marks
{
        public:
          int maths;
          int science;

          //constructor
          Marks()
          {
            cout << "Inside Constructor"<<endl;
            cout << "C++ Object created"<<endl;
          }

          //Destructor
          ~Marks()
           {
            cout << "Inside Destructor"<<endl;
            cout << "C++ Object destructed"<<endl;
          }
};

int main( )
{
        Marks m1;
         Marks m2;
         return 0;
}
```

When the above program is compiled and executed, it produces the following result:

```
Inside Constructor
C++ Object created
Inside Constructor
C++ Object created

Inside Destructor
C++ Object destructed
```

```
Inside Destructor
C++ Object destructed
```

Program to find sum of two numbers using constructor and destructor

```
#include <iostream>
using namespace std;
class myclass
{
int a,b;
public:
myclass();
~myclass();
void show();
};

myclass::myclass()
{
cout << "In constructor\n";
a = 30;
b = 20;
}

myclass::~myclass()
{
cout << "Destructing...\n";
}

void myclass::show()
{
cout << "A ="  << a << endl << "B =" << b << endl;
cout << "Sum is " << a+b << endl;
}
int main()
{
myclass ob;
ob.show();
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
In constructor
A =30
B =20
Sum is 50
Destructing...
```

Program to perform addition, subtraction and multiplication of two complex numbers using constructor

```
#include<iostream>
using namespace std;
class complex
{
private:
  float rp1, rp2;
  float ip1, ip2;
public:
    complex (float, float, float, float);
  void add ();
  void sub ();
  void mul ();
};
complex::complex (float x, float y, float z, float w)
{
  rp1 = x;
  rp2 = z;
  ip1 = y;
  ip2 = w;
}
void complex::add ()
{
  float x, y;
  x = (rp1) + (rp2);
  y = (ip1) + (ip2);
    if(y>0.0)
    {
cout<<"\nResult of Addition is "<<x<<" + "<<y<<" i";
    }
    else
cout<<"\nResult of Addition is "<<x<<"  "<<y<<" i";;
}

void complex::sub ()
{
  float x, y;
  x = (rp1) - (rp2);
  y = (ip1) - (ip2);
    if(y>0.0)
    {
cout<<"\nResult of Subtraction is "<<x<<" + "<<y<<" i";
    }
    else
```

```
cout<<"\nResult of Subtraction is "<<x<<"  "<<y<<" i";;
}

void complex::mul ()
{
 float x, y;
 cout << rp1 << rp2 << ip1 << ip2 << endl;
 x = (rp1 * rp2) - (ip1 * ip2);
 y = (rp1 * ip2) + (rp2 * ip1);
   if(y>0.0)
   {
cout<<"\nResult of Multiplication is "<<x<<" + "<<y<<" i";
   }
   else
cout<<"\nResult of Multiplication is "<<x<<"  "<<y<<" i";;
}

int main ()
{
 float a, b, c, d;
 int ch;
 cout << "Enter first complex no." << endl;
 cin >> a >> b;
 cout << "Enter second complex no." << endl;
 cin >> c >> d;
 comp c1 (a, b, c, d);
 do
   {
    cout << "1.Add" << endl;
    cout << "2.Sub" << endl;
    cout << "3.Mul" << endl;
    cout << "4.Exit" << endl;
    cout << "Enter choice" << endl;
    cin >> ch;
    switch (ch)
          {
          case 1:
            {
              c1.add ();
              break;
            }
          case 2:
            {
              c1.sub ();
              break;
            }
          case 3:
            {
              c1.mul ();
              break;
            }
          case 4:
            {
              exit (0);
            }
          }
    }
  while (ch != 4);
}
```

When the above code is compiled and executed, it produces the following result:

```
Enter first complex no. 2 5
Enter second complex no -3 -4
Enter choice 2
Result of Subtraction is 5 + 9 i
```

Restrictions that apply to constructors, destructor

- Constructors and destructors do not have return types nor can they return values.
- References and pointers cannot be used on constructors and destructors, because their addresses cannot be taken.
- Constructors cannot be declared with the keyword virtual.
- Constructors and destructors cannot be declared static, const, or volatile.
- Unions cannot contain class objects that have constructors or destructors.

Differences between Constructor and Destructor

Constructor	**Destructor**
1) A constructor is a special member function whose main task is to initialize the objects of its class.	1) Destructor is a special member function whose main task is to destroy the objects that have been created by a constructor.
2) The name of the constructor is same as that of the class name.	2) The name of the constructor is same as that of the class name but it is prefixed with a tilde (~) sign.
3) Constructor is invoked when the object of its associated class is created.	3) It is invoked implicitly by the compiler upon exit of the program to clean up the storage that is no longer available.
4) Constructor can take arguments.	4) Destructor does not take any arguments.

5) Example:
class emp
{
int eno;
	5) Example:
	class emp
	{
	int eno;

```
        float salary;
        public:
        test() //constructor
        {
                eno=100;
                salary=10000;
        }
        ….. //member functions goes here
};
```

```
        float salary;
        public:
        test() //constructor
        {
                eno=100;
                salary=10000;
        }
       ~test()  //destructor
       {
        cout<<"employee object destroyed";
       }
       …..//member functions goes here

        };
```

8. Summary

- A constructor is a special member function whose main task is to initialize the objects of its class.
- C++ constructors do not return any value so constructor has no return type not even void data type.
- When a constructor is declared for a class, then it is must to initialize the objects.
- The three different types of constructors that are supported by C++ programming language are: Default Constructor, Parameterized Constructor and Copy Constructor.
- A constructor that accepts no parameters is called as default constructor.
- The constructor that can take arguments or parameters is called as parameterized constructor or argument constructor.
- The parameters of a constructor can be of any type except that of the class to which it belongs.
- A copy constructor takes a reference to an object of the same as itself as an argument.
- The process of initializing through a copy constructor is known as copy initialization.
- The Copy Constructor can also be defined using a const keyword.
- The constructor of a class may also be overloaded so that even with different number and types of initial values, an object may still be initialized like any other function.
- The overloading of constructors is used to increase the flexibility of a class by having more number of constructors for a single class.
- The main advantage of dynamic initialization is that it is possible to provide initialization formats by using overloaded constructors, which tries to give the flexibility so that different format of data can be used at runtime based on the situation.
- The constructor that is used to allocate the memory to the objects at the run time is called as dynamic constructor.
- The memory allocation to objects is allocated with the help of 'new' operator.

- The job of memory de-allocation from object is done by special member function of the class called as destructor.
- Destructor is a member function whose name is the same as the class name but is preceded by tilde (~) symbol.
- Only one destructor can be defined for a class.
- References and pointers cannot be used on constructors and destructors, because their addresses cannot be taken.