Assignment 3

Submission Date- 22 /10/2024

Title – Study of Constructor and Operator Overloading in C++

Objective- To illustrate the working of constructor and operator overloading in C++ Programming

Problem Statement-

Implement a class Complex which represents the Complex Number data type. Implement the following operations:
1. Constructor (including a default constructor which creates the complex number (0+0i).
2. Overloaded operator + to add two complex numbers.
3. Overloaded operator * to multiply two complex numbers.
4. Overloaded << and >> to print and read Complex Numbers.

Software & Hardware requirements- any Text editor and Terminal in Linux/ Turbo C++ Compiler installed on PC.

Theory-

Complex number –

A complex number is an element of a number system that extends the real numbers with a specific element denoted i, called the imaginary unit and satisfying the equation $i^2=-1$; every complex number can be expressed in the form a+bi, where a and b are real numbers. For the complex number a+bi, where a is called the real part, and b is called the imaginary part.

Constructor-

Constructor in C++ is a special method that is invoked automatically at the time an object of a class is created. It is used to initialize the data members of new objects

generally. The constructor in C++ has the same name as the class or structure. It constructs the values i.e. provides data for the object which is why it is known as a constructor.

Syntax of Constructors in C++

```
<class-name> ()
{
...
}
```

Eg.

```
class MyClass
{     // The class
  public:          // Access specifier
   MyClass()
 {     // Constructor
      cout << "Hello World!";
    }
};

int main()
{
 MyClass myObj;     // Create an object of MyClass (this will call the constructor)
  return 0;
}
```

Types of Constructor-

Types of Constructors in C++

Constructors can be classified based on in which situations they are being used. There are 4 types of constructors in C++:

1. **Default Constructor**: No parameters. They are used to create an object with default values.
2. **Parameterized Constructor**: Takes parameters. Used to create an object with specific initial values.

3. **Copy Constructor**: Takes a reference to another object of the same class. Used to create a copy of an object.
4. **Move Constructor**: Takes an rvalue reference to another object. Transfers resources from a temporary object.

1. Default Constructor

A default constructor is a constructor that doesn't take any argument. It has no parameters. It is also called a zero-argument constructor.

Syntax of Default Constructor

```
className()
 {
   // body_of_constructor
}
```

The compiler automatically creates an implicit default constructor if the programmer does not define one.

2. Parameterized Constructor

Parameterized constructors make it possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

Syntax of Parameterized Constructor

```
className (parameters...)
{
    // body
}
```

If we want to initialize the data members, we can also use the initializer list as shown:

```
MyClass::MyClass(int val) : memberVar(val) {};
```

3. Copy Constructor

A copy constructor is a member function that initializes an object using another object of the same class.

Syntax of Copy Constructor

Copy constructor takes a reference to an object of the same class as an argument.

```
ClassName (ClassName &obj)
{
  // body_containing_logic
}
```

Just like the default constructor, the C++ compiler also provides an implicit copy constructor if the explicit copy constructor definition is not present.

Here, it is to be noted that, unlike the default constructor where the presence of any type of explicit constructor results in the deletion of the implicit default constructor, the implicit copy constructor will always be created by the compiler if there is no explicit copy constructor or explicit move constructor is present.

4. Move Constructor

The move constructor is a recent addition to the family of constructors in C++. It is like a copy constructor that constructs the object from the already existing objects. but instead of copying the object in the new memory, it makes use of move semantics to transfer the ownership of the already created object to the new object without creating extra copies.

It can be seen as stealing the resources from other objects.

Syntax of Move Constructor

```
className (className&& obj)
{
    // body of the constructor
}
```

The move constructor takes the rvalue reference of the object of the same class and transfers the ownership of this object to the newly created object.

Like a copy constructor, the compiler will create a move constructor for each class that does not have any explicit move constructor.

Operator Overloading-

C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. Operator overloading is a compile-time polymorphism.

For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Numbers, Fractional Numbers, Big integers, etc.

Example:

```
int a;
float b,sum;
sum = a + b;
```

Here, variables "a" and "b" are of types "int" and "float", which are built-in data types. Hence the addition operator '+' can easily add the contents of "a" and "b". This is because the addition operator "+" is predefined to add variables of built-in data type only.

Code

```
#include<iostream>
using namespace std;
class complex
{
float x;
float y;
public:
complex()
{
x=0;
y=0;
}
complex operator+(complex);
complex operator*(complex);
friend istream &operator >>(istream &input,complex &t)
{
cout<<"Enter the real part";
input>>t.x;
cout<<"Enter the imaginary part";
input>>t.y;
}
friend ostream &operator <<(ostream &output,complex &t)
{
output<<t.x<<"+"<<t.y<<"i\n";
}
};
complex complex::operator+(complex c)
{
complex temp;
temp.x=x+c.x;
temp.y=y+c.y;
return(temp);
}
complex complex::operator*(complex c)
{
complex temp2;
temp2.x=(x*c.x)-(y*c.y);
```

```
temp2.y=(y*c.x)+(x*c.y);
return (temp2);
}
int main()
{
complex c1,c2,c3,c4;
cout<<"Default constructor value=\n";
cout<<c1;
cout<<"\n Enter the 1st number\n";
cin>>c1;
cout<<"\n Enter the 2nd number\n";
cin>>c2;
c3=c1+c2;
c4=c1*c2;
cout<<"\n The first number is ";
cout<<c1;
cout<<"\n The second number is ";
cout<<c2;
cout<<"\n The addition is ";
cout<<c3;
cout<<"\n The multiplication is ";
cout<<c4;
return 0;
}
/*
```

Out Put :~$ ./a.out

```
Default constructor value= 0+0i
Enter the 1st number
Enter the real part2
Enter the imaginary part4
Enter the 2nd number
Enter the real part4
Enter the imaginary part8
The first number is 2+4i
The second number is 4+8i
The addition is 6+12i
The multiplication is -24+32i
student@student-OptiPlex-3010:~$
*/
```

Conclusion-

Understanding concept of constructor and operator overloading and their usage is essential for effective object initialization and management.