Object Oriented Programming in C++

Nepal College of Information Technology

Course Instructor : Er. Rabina Chaudhary

Chapter 3:Message, Instance and Initialization

Message Passing:

- A request for an object to perform its operation is called message
- A message for an object is a request for execution of a procedure and therefore will invoke a function in the object that generates the desired results
- Message passing involves specifying
    - name of the object
    - name of the function
    - information to be sent

Message Passing:

Syntax for message passing:

```
objectName . functionName (attributes/ information);
```

Example:

```
objectA . setData(20,30);
```

object   message   information

Initialization in C++:

```
class Test
        {
                int x,y;
                public:
                void setData()
                {
                        cout<<"Enter values of x and y";
                        cin>>x>>y;
                }
        };
void main()
{
        Test objOne;
        objOne.setData(); /*objOne.setData() invokes the member function setData() which assigns initial
value of private data members x and y of objOne. */
}
```

Initialization in C++:

```
 class Test
		{
			int x,y;
			public:
			void setData(int a,int b)
			{
				x=a;
				y=b;
			}
		};
 void main()
{
		Test objOne;
		objOne.setData(10,20); /*objOne.setData() invokes the member function setData(int,int) and
passes values to assign initial value of private data members x and y of objOne. */
}
```

Constructor:

- A constructor is a member function of a class which initializes object of the class
- Constructor is automatically called when object of the class is created
- Constructor are useful for initializing the values of data of object

Constructor has following characteristics:

1. Constructor name should be same as its class name
2. Constructor does not have any return type not even void
3. It is called automatically when an object is created
4. It should be declared in public section
5. It cannot be static

Constructor:

Syntax for declaration of constructor:

```
class ClassName
{
        ...// data members
        public:
        ClassName(); //constructor
        ... //other member functions
};
```

Constructor:

- Example:

```
 class Construct
{
            int x,y;
            public:
            Construct()
            {
              x=10;
               y=20;
            }
};

 void main()
{
            Construct  obj1 ;
            …
}
```

Types of Constructor:

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor

1. Default Constructor:

- A constructor without any argument is called default constructor
- It has no parameters
- If we do not specify a constructor explicitly, the compiler generates a default constructor implicitly

Default Constructor:Example

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

 class Construct
        {
                int x,y;
                public:
                Construct()
                {
                        x=0;
                        y=0;
                }
                void display()
                {
                cout<<“Value of x=“<<x<<endl<<“Value of y =“<<y;
                }
        };
```

```
void main()
 {
        Construct obj;
        obj.display();
        getch();
  }
```

2. Parameterized Constructor:

1. Constructor with arguments is parameterized constructor
2. Here, we pass arguments while defining object of the class which are assigned to data members of the class
3. It is used to initialize data members of different objects with different initial values when they are created

Parameterized Constructor: Example

```
#include<iostream.h>
#include<conio.h>
 class Construct
      {
      int x,y;
      public:
            Construct() //default constructor
            {
            x=0;
            y=0;
            }
            Construct(int a,int b) //parameterized  constructor
            {
                  x=a;
                  y=b;
            }
            void display()
            {
            cout<<"Value  of x="<<x<<endl;
          cout<<"Value   of y ="<<y<<endl;
            }
      };
```

```
void main()
  {
        Construct  obj1(100,200),  obj2(500,600)  ;
        obj1.display();
        obj2.display();
        getch();
  }
```

Practice:

Q1. Write a program to define a class Rational. Use constructor to initialize the values of data members of object (i.e num and denum) and display the rational number. Use appropriate member functions as required.

2.2 Constructor with default arguments

- Parameterized constructor with default values in arguments

Example:

```
class Construct
	{
	int a,b,c;
	public:
		Construct();
		Construct(int x,int y=20, int z=30);
		void display();
	};
```

```
Construct:: Construct()
        {
                a=0;
                b=0;
                c=0;
        }
Construct::Construct(int x,int y, int z)
        {
                a=x;
                b=y;
                c=z;
        }
 void Construct:: display()
        {
        cout<<endl<<"Value of a="<<a;
        cout<<endl<<"Value of b="<<b;
        cout<<endl<<"Value of c="<<c;
        }
```

```
 void main()
{
        Construct obj1(1,2,3);
        Construct obj2(120,89);
        obj1.display();
        obj2.display();

        getch();
}
```

Practice:

Q1. Define a class Bank. Calculate simple interest using default value of rate=1.5%. Use constructor to initialize the object and appropriate member functions to display and calculate the result.

Q2. Define a class Complex. Write a program to add two complex numbers. Use constructor to initialize the objects.

Q3. . Define a class Complex. Write a program to add two complex numbers using friend function. Use constructor to initialize the objects.

3. Copy Constructor:

- Copy constructor is a constructor which creates an object as a copy of existing object of same class
- Copy constructor is used to create an object by initializing it with an object of the same class, which has been already created
- It uses reference to an object of same type (class) as argument

Copy Constructor:

Syntax for declaration of copy constructor:

```
class Construct
{
        …
        public:
        …
        Construct(Construct &); //copy constructor
};
```

Copy Constructor:

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

 class Construct
                    {
                    int a,b,c;
                     char name[20];
                    public:
                                        Construct();                    //default constructor
                                        Construct(int x,int y, int z);  //parameterized constructor
                                        Construct(Construct &);         //Copy constructor
                                        void display();
                    };
Construct:: Construct()
                    {

                                        a=0;
                                        b=0;
                                        c=0;
                    }
```

```
Construct::Construct(int x,int y, int z)
                {
                                a=x;
                                b=y;
                                c=z;
                }
Construct::Construct(Construct &obj)
{
                a=obj.a;
                b=obj.b;
                c=obj.c;
}
 void Construct:: display()
                {
                cout<<endl<<"Value of a="<<a;
                cout<<endl<<"Value of b="<<b;
                cout<<endl<<"Value of c="<<c;
                }
 void main()
{
            Construct obj1(10,20,30);     //parameterized constructor called
            Construct obj2;               //Default constructor called

            cout<<"Object 1 :"<<endl;
            obj1.display();

            cout<<endl<<endl<<endl;
            cout<<"Object 2: "<<endl;
            obj2.display();

            cout<<endl<<endl<<endl;
            cout<<"Object 3 : "<<endl;
            Construct obj3(obj1);         //copy Constructor called
            obj3.display();

            getch();
}
```

Can Constructor be Overloaded?

- Yes, Constructor can be overloaded in same way as functions can be overloaded
- A class can have more than one constructor where each constructor take different set of parameters
- Constructor can be overloaded as class can have more than one constructor having same name but different signature
- When we define an object of a class, correct version of the constructor is called on the basis of passed arguments

Write an example program to illustrate constructor overloading.

Practice:

- Q1. Create a class called Mountain with data members name, height and location. Use constructor that initialize the members to the values passed to it as parameters. A function cmpHeight() to compare height of two objects and function displayInfo() to display information of mountain. In main create two objects of the class mountain and display the information of mountain with greatest height.

Destructor:

- A special member function of the class that is called when an object of the class is destroyed
- A constructor initializes the object, whereas a destructor destroys the object
- Destructor de allocates the memory allocated by object
- It is invoked automatically when the object goes out of the scope

Characteristics of destructor:

1. Destructor has same name as class and is preceded by a tilde(~)
2. Destructor doesn’t have any argument
3. Destructor doesn’t have any return type, not even void
4. It is called automatically whenever an instance of the class to which it belongs , goes out of scope
5. It is defined in public section of class

Destructor:

Syntax:

```
class className
{
        public:
        ~className ()
                {
                        destructor code
                }
                …
};
```

Example:

```
#include<iostream.h>
#include<conio.h>
class ABC
                {
                        public:
                                    ABC(){
                                    cout<<“This is a constructor”<<endl;
                                    }
                                    ~ABC(){
                                    cout<<“This is a destructor”<<endl;
                                    }
                };
 void main()
                {
                                    {
                                                    ABC obj1, obj2;
                                    }
                getch();
                }
```

Destructor:

Objects are destroyed in reverse direction of creation

Example program:

```
#include<iostream.h>
#include<conio.h>
class ABC
{
                int id;
                public:
                ABC()                   //default constructor
                  {
                                id=0;
                  }
                ABC(int a)              //parameterized constructor
                  {
                                id=a;
                                cout<<endl<<"Constructor called for object : "<<id<<endl;
                }
                ~ABC()                  //destructor
                {
                                cout<<endl<<"Destructor called for object :"<<id<<endl;
                }
};
```

Program continued

```
void main()
          {

                      {
                               cout<<"**************Block  1*****************"<<endl;
                               ABC obj1(1),obj2(2);
                                         {
                                                   cout<<endl<<endl<<"***************Block  2***************"<<endl;
                                                   ABC obj3(3),obj4(4);
                                                   cout<<endl<<endl<<endl<<endl;
                                                   cout<<endl<<endl<<"*********End  of block 2**********"<<endl;
                                          }
                               cout<<endl<<endl<<"***********End  of block 1***********"<<endl;
                     }
                     getch();
          }
```

Can destructor be overloaded?

- Destructor cannot be overloaded
- There can be only one destructor in a class
- Destructor doesn’t take any argument nor has a return type
- So multiple destructors with different signatures are not possible in a class
- Hence, destructor overloading is not possible

What will be the output of this program?

```
 class Car
{
                    static int count;
                    public:
                    Car()
                    {
                                        count++;
                                        cout<<“There are “<<count<<“ number of
cars”<<endl;
                    }
                    ~Car()
                    {
                                        count--;
                    cout<<“There are “<<count<<“ number of cars”<<endl;
                    }
};
```

Program continued

```
 void main()
{
 cout<<“We are in main”<<endl;
Car c1,c2,c3,c4;
{
                 cout<<“We are in block1”<<endl;
                 Car c5;
}
{
                 cout<<“We are in block2”<<endl;
                 Car c6;
}
 cout<<“We have re-entered main”<<endl;
}
```

Practical Questions:

Q1. Imagine a tollbooth at a bridge. Carspassing by the booth are expected to pay five-rupee toll. Mostly they do, but sometimes a car go by without paying. The tollbooth keeps track of the number of cars that have gone by and of the total amount of money collected.

Model this tollbooth with class (say TollBooth). The two data members to hold number of cars and amount of money collected. The constructor may initialize both these to 0. A member function (say payingCar() increments the car total and adds 5 to the cash total, while another member function (say noPayCar()) increments the car total but adds nothing to the cash total. Finally member function display() displays the two totals.

Hint: allow user to press a particular key to indicate a paying car and another key to indicate non paying car.

Q2. Define a class to represent a bank account. Include the following members:

Data Members:

i) Name of the account holder

ii) Account number

iii) Balance Amount is the account

Member functions:

i) Open an account

ii) Deposit and withdraw money

iii) Display account information

Write a program to test this class for 10 costumers. Make use of all three types of constructors (whenever appropriate).