Name of Teacher: Miss Radhika M. Patil

Class: B.Sc. Computer Science (Entire)- II **Semester : 3**

Course Title: Object Oriented Programming using C++

Unit 3: Constructor, Destructor, Operator Overloading

1. Constructor:

- Constructor is a special member function of a class .
- It is special because its name is **same as that of the class name.**
- Main task of constructor is to initialize the object of that class.
- There is no need to invoke the constructor explicitly, it is called/invoked automatically when the object of its associated class is created.

• Characteristics of Constructor:

1. The name of the constructor should be same as that of the class name.
2. Constructors are used to initialize the objects of that class.
3. Constructor of class is invoked automatically when object of class is created.
4. Constructor is called only once when object is created.
5. Constructors can also be overloaded.
6. Constructor does not return any value, not even void so they have no return type.
7. Constructors can not be **virtual.**
8. They can have default argument.

Program:

```
class Student
{
        int roll_no;
        char sname[20];
    public:
        Student()      // Constructor definition
        {
            roll_no=9129;
            strcpy (sname, “Abc”);
         }
        void display()
        {
            cout<<“\n Roll No.:”<<roll_no;
            cout<<“\n Name:”<<sname;
        }
};
```

```
void main()
 {
        Student s;
        s.display();
        getch();
 }
```

Assignment: Write an OOP to perform sum of two integer numbers using constructor.

Solve this assignment and attach the screenshot of program in pdf file along with the notes.

- **Types of Constructor:**

Constructors are of three types:

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

1. Default Constructor

- Constructor which **doesn't take any parameter** is called default constructor.
- If the programmer doesn't specify default constructor in program then the compiler provide default constructor.
- In C++, we can overload default compiler generated constructor.
- In both the cases (i.e. user created default constructor or constructor generated by compiler) default constructor is always **parameter less** i.e. it has no parameters.
- In default constructor data member of all objects are initialized to same value.

Syntax

```
class_name()
{
        //statements;
}
```

Program:

```
class Integer
{
        int a,b;
    public:
        Integer()
        {
            a=10;
            b=20;
        }

        void display()
        {
            cout<<"\n a= "<<a;
            cout<<"\n b= "<<b;
        }
};
void main()
{
        Integer i1, i2;
        cout<<" Object 1:"<<endl;
        i1.display();
         cout<<" Object 2:"<<endl;
        i2.display();
        getch();
}
```

Output:

```
Object 1:
a=10
b=20
Object 2:
a=10
b=20
```

2. Parameterized Constructor:

- In default constructor data member of all objects are initialized to same value.
- However, it may be necessary to initialize various data members of different objects with different values when they are extracted.
- It can be achieved by passing argument the to constructor function when objects are created.
- Constructor that **<u>can take an argument/parameter</u>** is called parameterized constructor.

Syntax:

```
class_name (parameter_list)
{
      //statements;
}
```

Example:

```
class Integer
{
        int a, b;
    public:
        Integer(int x, int y)
        {
            a=x;
            b=y;
        }
};
```

When constructor is parameterized, we must pass initial values as arguments to constructor function when object is created.

This can be done in two ways:

1. By calling constructor function implicitly
2. By calling constructor function explicitly.

1. By calling constructor function implicitly

e.g.

```
Integer i1= Integer( 10,20);
```

This statement creates an object i1 of class Integer and passes the values 10 and 20 to constructor.

2. By calling constructor function explicitly

e.g.

```
Integer i (30,50);
```

This method sometimes called as short hand method.

Program:

```
class Integer
{
        int a,b;
    public:
        Integer(int x, int y)
        {
            a=x;
            b=y;
        }

        void show()
        {
            cout<<"\n a="<<a;
            cout<<"\n b="<<b;
        }
};

void main()
{
        Integer i1 (10,20);
        Integer i2=Integer (30,40);
        cout<<"\n Object 1:"<<endl;
        i1.show();
        cout<<"\n Object 2:"<<endl;
        i2.show();
    getch();
}
```

Output:

```
Object 1:
a=10
b=20
Object 2:
a=30
b=40
```

3. Copy Constructor:

1. A copy constructor is a member function which initializes an object using another object of same class.
2. Copy constructor is used to copy object passed to it as an argument to constructor function.
3. Copy constructor is also a parameterized constructor that has reference of object as parameter.
4. Copy constructor if not defined in class then compiler itself defines it once.
5. If class has some pointer variable and some dynamic memory allocation then it is most common to have a copy constructor.

Syntax:

```
class_name (class_name &obj)
{
      //statements;
 }
```

Here, obj is reference to an object that is being used to initialize another object.

Example:

```
class Code
{
        int id;
    public:
        Code()
        {
            id=10;
        }
        Code (int x)
        {
            id=x;
        }
        Code (Code &A)     //copy constructor
         {
             id=A.id;
         }
         void show()
         {
            cout<<"\n id="<<id;
         }
};
void main()
{
        Code A;
        Code B(100);
        Code C(B);
        cout<<"\n Object A:";
        A.show();
        cout<<"\n Object B:";
        B.show();
        cout<<"\n Object C:";
        C.show();
   getch();
}
```

Output:

```
Object A:
id=10
Object B:
id=100
Object C:
id=100
```

- **Destructor:**

- Destructor is exactly opposite to the constructor.
- Like constructor, destructor is also a member function whose name same as that of the class name but. **preceded by a tilde (~) operator.**
- Destructor is called automatically when object is released or destroyed from memory.

- **Rules for writing a destructor function:**

1. Destructor must have a same name as the class name preceded with a tilde operator.
2. It never takes any argument nor return any value .
3. It can not be declared as static, volatile or constant.
4. It takes no arguments and therefore can’t be overloaded.
5. It should have public access in class declaration.

Syntax:

Declaration:

```
~ class_name();            //destructor declaration
```

Definition:

```
class_name ::~ class_name()            //destructor definition outside the class
  {
        //statements;
  }
```

Program:

```
class A
{
        public:
                A()
                {
                        cout<<"\n Constructor called for";
                }

                ~A()
                {
                        cout<<"\n Destructor called.";
                }
};

void main()
{
        A obj1;
        cout<<"\n Object 1:";
        int x=1;
        if( x= =1)
        {
                A obj2;
                cout<<"\n Object 2:";
        }
  getch();
}
```

Output:

Constructor called for Object 1:
Constructor called for Object 2:
Destructor called
Destructor called

Program 2:

```
int count=0;
class Alpha
{
        public:
                Alpha()
                {
                        count ++;
                        cout<<"\n No. of object created:"<<count;
                }

                ~Alpha()
                {
                        cout<<"\n No. of object destroyed:"<<count;
                        count - -;
                }
};
void main()
{
        cout<<"\n Enter into main:";
        Alpha A1, A2, A3, A4;
        cout<<"\n Object 1:";
        {
                cout<<"\n Enter into Block1:"<<endl;
                Alpha A5;
        }

        {
                cout<<"\n Enter into Block2:"<<endl;
                Alpha A6;

        }
    cout<<"\n Re-enter into main:";
  getch();
}
```

Output:

Enter into main:

No.of object created:1

No.of object created:2

No.of object created:3

No.of object created:4

Enter into Block 1:

No.of object created:5

No.of object destroyed:5

Enter into Block 2:

No.of object created:5

No.of object destroyed:5

Re-enter into main

- **Operator overloading:**

- Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined data type.
- Operator overloading is used to overload or redefines most of the operators available in C++.
- It is used to perform the operation on the user-defined data type.
- For example, C++ provides the ability to add the variables of the user-defined data type that is applied to the built-in data types.
- The advantage of Operators overloading is to perform different operations on the same operand.
- When operator is overloaded , its original meaning should not be lost. For instant operator + which has been overloaded to add two class variables (objects) can still be used to add two integers.
- To define an additional task to operator, we must specify what it means in relation to the class in which operator is applied.
- This can be done with **special function** called **operator function** which describes the task.

- **Operator that cannot be overloaded are as follows:**

1. Scope operator (::)
2. sizeof
3. member selector(.)
4. member pointer selector(*)
5. ternary operator(?:)

General Syntax:

return_type operator op (argument_list);

Where,

return_type: is the type of value returned by specified operation.

operator is keyword.

op is operator being overloaded.

'operator op' is the function name.

operator function must be either member function of friend function.

- **Process of Overloading operator involves following steps:**

1. Create class that defines data type that is to be used in overloading operation.
2. Declare operator function 'operator op' in public part of class. It may be either member function or friend function.
3. Define operator function to implement required operation.

- **Rules for Overloading operator**

1. Only existing operators can be overloaded.
2. New operators can't be created and overloaded.
3. Overloaded operator must have at least one operand of user defined data type (i.e. object).
4. We can't change the meaning of an operator i.e. we can't redefine + operator to subtract one value from other.
5. Overloaded operators follow the syntax rules of original operator.
6. When **unary operators** are overloaded through a **member function** take **no explicit arguments**, but, if they are overloaded by using a **friend function**, then it will take **one argument**.
7. When **binary operators** are overloaded through a **member function** takes **one explicit argument**, and if they are overloaded through a **friend** function then it will take **two explicit arguments**.
8. In case of binary operator overloading using member function, the left hand operand must be an object of relevant class.

1. Unary Operator Overloading:

- Let us consider unary operator , when used as unary it takes just one operand.
- This operator changes sign of an operand when applied to basic data item.
- It can be applied to an object in same way as it is applied to integer or float variable.
- The unary operator when applied to an object then it should **change the sign of each of it's data members.**

- **Unary Operator Overloading Using Member Function:**

When **unary operators** are overloaded through a **member function** it will take **no explicit arguments.**

Program:

```
class UnaryDemo
 {
            int x,y,z;
      public:
            UnaryDemo (int, int, int);
            void display();
            void operator –();
 };
```

```
UnaryDemo:: UnaryDemo (int a, int b, int c)
{
        x=a;
        y=b;
        z=c;
}

void UnaryDemo :: display()
{
      cout<<"\n x="<<x;
      cout<<"\n y="<<y;
      cout<<"\n z="<<z;
}

 void  UnaryDemo :: operator –()
{
       x=-x;
       y=-y;
       z=-z;
}
```

```
void  main()
{
        UnaryDemo  d(10,20,30);
        cout<<"\n Before Overloading :";
        d.display();
         -d;                  //operator overloading
         cout<<"\n After Overloading:";
         d.display();
       getch();
}
```

Output:

```
Before Overloading:
 x=10
 y=20
 z=30
After Overloading:
 x= -10
 y= -20
 z= -30
```

- **Unary Operator Overloading Using Friend Function:**

When **unary operators** are overloaded using a **friend function**, then it will take **one argument**.

Program:

```
class UnaryDemo
 {
            int x,y,z;
      public:
            UnaryDemo (int, int, int);
            void display();
            friend void operator –(UnaryDemo);
 };
```

```
UnaryDemo:: UnaryDemo (int a, int b, int c)
{
        x=a;
        y=b;
        z=c;
}

void UnaryDemo :: display()
{
      cout<<"\n x="<<x;
      cout<<"\n y="<<y;
      cout<<"\n z="<<z;
}

 void operator –(UnaryDemo d)
{
       x=-d.x;
       y=-d.y;
       z=-d.z;
}
```

```
void  main()
{
          UnaryDemo  d1(10,20,30);
          cout<<"\n Before Overloading :";
          d1.display();
           -d1;                  //operator overloading
           cout<<"\n After Overloading:";
           d1.display();
         getch();
}
```

Output:

Before Overloading:
x=10
y=20
z=30
After Overloading:
x= -10
y= -20
z= -30

Assignment:

1. Write an Object Oriented Program (OOP) to increment value of variable using unary ++ operator overloading.
2. Write an OOP to decrement the value of variable using unary -- operator overloading.
3. Write an OOP to overload logical not operator.

Solve these assignments and attach the screenshot of program in pdf file along with the notes.

- **Binary Operator Overloading:**

1. A same mechanism can be used to overload binary operator.
2. Binary operator requires two operands.
3. As a rule in overloading a binary operator, the left hand operand (object) is used to invoke operator function and right hand operand is used as an argument passed to operator function.

1. Binary Operator Overloading using Member Function:

Program:

```
class Temp
{
                int a, sum;
      public:
              Temp (int x)
               {
                        a=x;
               }

            void operator + (Temp);
};
```

```
void Temp:: operator + (Temp t)
{
        sum = a+t.a;
        cout<<“\n a of object 1:”<<a;
        cout<<“\n a of object 2:”<<t.a;
        cout<<“\n Sum=”<<sum;
}

void main()
{
        Temp t1 (10);
        Temp t2 (20);
         t1+t2;          //overloading + operator
          getch();
}
```

Output:

```
a of object 1:10
a of object 2: 20
Sum=30
```

- **Consider an operator function**

```
void Temp:: operator + (Temp t)
{
        sum = a+t.a;
        cout<<"\n a of object 1:"<<a;
        cout<<"\n a of object 2:"<<t.a;
        cout<<"\n Sum="<<sum;
}
```

1. It's a member function of class Temp. Function is expected to add two object values.
2. Consider the statement that invokes the operator function .

   t1+t2;
3. We know that a member function is invoked only by object of same class and dot operator.
4. Here, object **t1** takes responsibility of **invoking operator function** and object **t2** plays role of **an argument** i.e. it is passed to operator function as an argument.
5. Statement **t1+t2** is equivalent to

   t1. operator +(t2);
6. In +() function data member of object t1 i.e. a are accessed directly and data members of object t2 that is passed as argument are accessed using dot operator.

   Here, t.a refers to object t2 and a refers to object t1.

Assignment:

1. Write an OOP to overload binary – operator using member function..
2. Write an OOP to overload binary * operator using member function.
3. Write an OOP to overload binary / operator using member function.

Solve these assignments and attach the screenshot of program in pdf file along with the notes.

2. Binary Operator Overloading using friend Function:

Program:

```
class Temp
 {
                int a, sum;
       public:
               Temp (int x)
                {
                        a=x;
                }

            friend void operator + (Temp, Temp);
 };
```

```
void  operator + (Temp t1, Temp t2)
{
        sum = t1.a+t2.a;
        cout<<"\n a of object 1:"<<t1.a;
        cout<<"\n a of object 2:"<<t2.a;
        cout<<"\n Sum="<<sum;
}

 void main()
{
        Temp t1 (10);
        Temp t2 (20);
         t1+t2;          //overloading + operator
          getch();
}
```

Assignment:

1. Write an OOP to overload binary – operator using friend function..
2. Write an OOP to overload binary * operator using friend function.
3. Write an OOP to overload binary / operator using friend function.

Solve these assignments and attach the screenshot of program in pdf file along with the notes.

Program:
Write an OOP to perform addition of two complex numbers using binary operator overloading.

```
class Complex
{
            int x,y;
    public:
             Complex(int a, int b)
              {
                    x=a;
                    y=b;
              }
           void display();
           friend void operator +(Complex, Complex);

};

 void operator +(Complex c1, Complex c2)
{
     Complex t;
    t.x= c1.x+ c2.x;
    t.y= c1.y+ c2.y;
   cout<<"\n Addition of two complex numbers="<<t.x<<"+"<<t.y<<"i";
}
```

```
 void Complex::display()
{
      cout<<x<<"+"<<y<<"i";
}


void main()
{
       Complex c1(4,5);
       Complex c2(7,6);
       cout<<"\n First Complex no.:";
       c1.display();
       cout<<"\n Second Complex no.:";
       c2.display();
       Complex c3;
       c1+c2;   //overloading + operator
       getch();
}
```

- **Manipulation of Strings using operator overloading:**

1. There are no operators for manipulating the strings.
2. Main drawback of string manipulation is that whenever string is to be copied, programmer must first determine its length and then allocate required amount of memory .
3. The strings object may be treated like any other built-in data type.

Program:

Write an OOP to reverse a string using unary operator overloading

```
class Rev
{
            char s[20];
    public:
             Rev(char n[20])
             {
                   strcpy (s,n);
             }
        void display();
        void operator –();
};
```

```
void Rev::display()
{
    cout<<"\n String is:"<<s;
}

 void Rev::operator –()
{
       strrev (s);
}
```

```
void main()
{

      char n[20];
      cout<<"\n Enter String:";
      cin>>n;
      Rev r (n);
      cout<<"\n Before Overloading:";
      r. display();
      -r;
      cout<<"\n After Overloading:";
      r. display();
  getch();
}
```

Assignment:

1. Write an OOP to concatenate two strings using binary operator overloading

Solve these assignments and attach the screenshot of program in pdf file along with the notes.

THANK YOU…