Object Oriented Programming (CT-260) Lab 03

Introduction to Constructor and Destructor

Objectives

The objective of this lab is to familiarize students with constructor and destructor. By the end of this lab, students will be able to understand the concepts of constructor, different types of constructor, and destructor.

Tools Required

DevC++ IDE / Visual Studio / Visual Code

Course Coordinator –
Course Instructor –
Lab Instructor –
Prepared By Department of Computer Science and Information Technology
NED University of Engineering and Technology

Constructor

Constructor is a member function that is automatically called when we create an object. Constructor is used to initialize the data members according to the desired value. A constructor has the same name as that of the class (It is case sensitive) and it does not have a return type, not even void.

```
class  Person {
  public:
    Person(){ // constructor
      // code
    }
};
```

Constructor can also be used to declare run time memory (dynamic memory for the data members). By providing many definitions for constructor, you are actually implementing 'function overloading'; i.e. functions of same name but different parameters (not return type) and function definition.

Types of Constructor

There are 3 types of constructor:

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

Default Constructor

- This type of constructor does not have any arguments inside its parenthesis. It is called when creating objects without passing any arguments.

Example 1: Default Constructor

```
     #include <iostream>
     using namespace std;
     class construct {
     public:
      int a, b;
      construct(){
      a = 10;
      b = 20;
      }
     };
     int main(){
      construct c;
      cout << "a: " << c.a << endl
      << "b: " << c.b;
      return 1;
     }
```

Example 2: Default Constructor

```
     #include <iostream>
     using namespace std;
     class  Wall {  // declare a class
       private:
         double length;
       public:
         // default constructor to initialize variable
```

```
        Wall() {
          length = 5.5;
          cout << "Creating a wall." << endl;
          cout << "Length = " << length << endl;
        }
    };
    int main() {
      Wall wall1;
      return 0;
    }
```

Parameterized Constructor

- This type of constructor accepts arguments inside its parenthesis.
- Is called when creating objects by passing relevant arguments.
- 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.

Example 1: Parameterized Constructor

```
#include <iostream>
using namespace std;
class Point {
private:
 int x, y;
public:
 Point(int x1, int y1){
 x = x1;
 y = y1;
 }
 int getX(){
 return x;
 }
 int getY(){
 return y;
 }
};
int main(){
 Point p1(10, 15);
 cout << "p1.x = " << p1.getX() << ", p1.y = " <<p1.getY();
 return 0;
}
```

Example 2: Parameterized Constructor

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

// declare a class
class Wall {
  private:
    double length;
    double height;
  public:
    // parameterized constructor to initialize variables
    Wall(double len, double hgt) {
      length = len;
      height = hgt;
    }
    double calculateArea() {
      return length * height;
    }
};

int main() {
  // create object and initialize data members
  Wall wall1(10.5, 8.6);
  Wall wall2(8.5, 6.3);
  cout <<"Area of Wall 1: "<< wall1.calculateArea()<<endl;
  cout << "Area of Wall 2: " << wall2.calculateArea();
  return 0;
}
```

Copy Constructor

- A copy constructor is a member function which initializes an object using another object of the same class. The copy constructor in C++ is used to copy data of one object to another.

Example 1: Copy Constructor

```
#include<iostream>
#include<conio.h>
using namespace std;
class Example {
 int a, b;
public:
 Example(int x, int y) {
 a = x;
 b = y;
 cout << "\nIm Constructor";
 }
 Example(const Example& obj) {
 a = obj.a;
 b = obj.b;
 cout << "\nIm Copy Constructor";
 }
 void Display() {
 cout << "\nValues :" << a << "\t" << b;
 }
};
int main() {
 Example Object(10, 20);
 Example Object2(Object);
 Example Object3 = Object;
 Object.Display();
 Object2.Display();
 Object3.Display();
 return 0;
```

```
     }
```

Example 2: Copy Constructor

```
#include <iostream>
using namespace std;
class Wall {  // declare a class
     private:
         double *length;
         double *height;
     public:
    // initialize variables with parameterized constructor
     Wall(){
        length=NULL;
        height=NULL;
     }
     Wall(double len, double hgt) {
           length = new double;
           height = new double;
     if (length!=NULL && height !=NULL){
           *length=len;
           *height=hgt;
     }
     else
         exit(1);
    }
// copy constructor - Deep Copy with a Wall object as parameter
// copies data of the obj parameter
     Wall(Wall &obj) {
         length=new double;
         height= new double;
         *length = *(obj.length);
         *height = *(obj.height);
      }
     set_values(double len, double hei){
          *length=len;
          *height=hei;
     }
     double calculateArea() {
           return (*length) * (*height);
        }
         ~Wall(){
          delete height;
          delete length;
          }
     };
int main() {
   // create an object of Wall class
     Wall wall1(10.5, 8.6);
       // copy contents of wall1 to wall2
     Wall wall2 = wall1;
       // print areas of wall1 and wall2
     cout << "Area of Wall 1: "<<wall1.calculateArea()<< endl;
     cout << "Area of Wall 2: " << wall2.calculateArea()<<endl;
     wall2.set_values(2.3,1.5);
     cout<<"after changing values for wall 2"<<endl;
     cout <<"Area of Wall 1: " << wall1.calculateArea()<< endl;
     cout << "Area of Wall 2: " << wall2.calculateArea();
     return 0;
}
```

Destructor

A destructor is a special member function that works just opposite to constructor, unlike constructors that are used for initializing an object, destructors destroy (or delete) the object. Destructor function is automatically invoked when the objects are destroyed. It cannot be declared static or const. A destructor has the same name as the constructor but is preceded by a tilde (~).

Destructor does not have arguments. It has no return type not even void. An object of a class with a Destructor cannot become a member of the union. A destructor should be declared in the public section of the class. The programmer cannot access the address of destructor.

```
class  Person {
  public:
    Person(){ // constructor
      // code
    }
    ~Person(){ //destructor
      // code
    }
};
```

In a class definition, if both *constructor* and *destructor* are not provided, the compiler will automatically provide default *constructor* for you. Hence during the instantiation of the object, the data member will be initialized to any value.

Example:

```
#include <iostream>
using namespace std;
class HelloWorld{
public:
 HelloWorld(){
 cout<<"Constructor is called"<<endl;
 }
 ~HelloWorld(){
 cout<<"Destructor is called"<<endl;
 }
 void display(){
 cout<<"Hello World!"<<endl;
 }
};
int main(){
 HelloWorld obj;
 obj.display();
 return 0;
}
```

Exercise

1. Write a C++ program to copy the value of one object to another object using copy constructor. For example you can define a class for complex number and create its object for performing this task. Remember that you must allocate memory dynamically to data members.
2. In a virtual battle arena game called "Epic Clash," players control powerful characters to engage in intense battles against each other. Each character has distinct abilities and attributes, including health, attack power, and defense. Your task is to implement the Character class to encapsulate these attributes, provide getter and setter methods for them, and offer different constructors to create characters with various starting conditions.
   Encapsulation: Ensure that the character's attributes (health, attackPower, and defense) are private to the Character class, accessible only through appropriate getter and setter methods, allocated memory dynamically in heap.
   Constructors: Implement three constructors: A default constructor that initializes a character with standard starting values for health, attack power, and defense. A parameterized constructor that allows specifying custom values for health, attack power, and defense. A copy constructor that creates a new Character object by copying the data from an existing Character object.
3. Create a class tollbooth. The two data items are a type int to hold the total number of cars and a type double to hold the total amount of money collected. A constructor initializes both these to 0. When a car passes the toll, a member function called payingCar( ) increments the car total and adds 0.50 to the cash total. Another member function displays the two totals. DESIGN and IMPLEMENT this case. Make assumptions (if required) and include it in the description before designing the solution.
4. Some of the characteristics of a book are the title, author(s), publisher, ISBN, price, and year of publication. Design a class **bookType** that defines the book as an ADT.
   - Each object of the class **bookType** can hold the following information about a book: title, up to four authors, publisher, ISBN, price, and number of copies in stock. To keep track of the number of authors, add another member variable.
   - Include the member functions to perform the various operations on objects of type **bookType**.
     For example, the usual operations that can be performed on the title are to show the title, set the title, and check whether a title is the same as the actual title of the book. Similarly, the typical operations that can be performed on the number of copies in stock are to show the number of copies in stock, set the number of copies in stock, update the number of copies in stock, and return the number of copies in stock. Add similar operations for the publisher, ISBN, book price, and authors. Add the appropriate constructors and a destructor (if one is needed).
   - Write the definitions of the member functions of the class **bookType**.
   - Write a program that uses the class **bookType** and tests various operations on the objects of the class **bookType**. Declare an array of 100 components of type **bookType**. Some of the operations that you should perform are to search for a book by its title, search by ISBN, and update the number of copies of a book.