Lab Manual

CS214L – Object Oriented Programming Lab
Lab No: **05**
Topic: **Constructor and Destructor**

Class: **BSGM**
Semester: **II**
Session: **Spring, 2022**
Instructor: **Ms. Saira Qamar**

2

Lab Date: **March 1st, 2022**
Lab Time: **11:40hrs – 2:30hrs**

Air University Islamabad
FACULTY OF COMPUTING & ARTIFICIAL INTELLIGENCE
Department of Creative Technologies

Instructions

Submission: Use proper naming convention for your submission file. Name the submission file as Lab_NO_DEGREE_ROLLNUM (e.g. Lab_01_BSGM_00000). Submit the file on Google Classroom within the deadline. Failure to submit according to the above format would result in deduction of 10% marks. Submissions on the email will not be accepted.

Plagiarism: Plagiarism cases will be dealt with strictly. If found plagiarized, both the involved parties will be awarded zero marks in the assignment, all of the remaining assignments, or even an F grade in the course. Copying from the internet is the easiest way to get caught!

Deadline: The deadlines to submit the assignment are hard. Late submission with marks deduction will be accepted according to the course policy shared by the instructor. Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.

Comments: Comment your code properly. Bonus marks (maximum 10%) will be awarded to well comment code. Write your name and roll number (as a block comment) at the beginning of the solution to each problem.

Tip: For timely completion of the assignment, start as early as possible. Furthermore, work smartly - as some of the problems can be solved using smarter logic.

1. Note: Follow the given instructions to the letter, failing to do so will result in a zero.

Objectives

In this lab, you will learn:

- About Constructor and its Types
- How shallow and deep copy works
- About Destructor and its usage

Concepts

1. Constructor:

A constructor is a **member function** that has the **same name** as the **class**. It is automatically **called** when the object is **created** in memory, or **instantiated**. They are very useful for initializing member variables or performing other **setup operations**.

To **illustrate** how constructors work, look at this **Demo** class **declaration**:

```
classDemo
{
public:
	Demo(); // Constructor
};
Demo::Demo()
{
	cout<<"Welcome to the constructor!\n";
}
```

The class Demo only has **one member**: a **function** also named **Demo**. This function is the **constructor**. When an i**nstance** of this **class** is **defined**, the function **Demo** is automatically **called.**

Notice that the constructor's **functionheader** looks **different** than that of a **regularmemberfunction**. There is **noreturntype—notevenvoid**. This is because constructors are not executed by explicit function calls and cannot return a value

1.1 Purpose of Constructor:

To **understand pointers** we have to **understand how variables are stored**. Variables are stored in **memory cells** inside the **computer's memory**. The computer's memory is made up of consecutive memory cells, a byte long, each with a **unique** address.

A constructor's **purpose** is to **initialize** an object's **attributes**. Because the constructor executes as soon as the **object** is **created**, it can initialize the object's **datamembers** to

validvalues before those members are used by other code. It is a good practice to always write a constructor for every class.

1.2 The Default Constructor:

A **default** constructor is a constructor that takes **noarguments** as shown in previous example.
If you write a class with **noconstructor** whatsoever, when the class is **compiled,C++** will automatically write a **defaultconstructor** that does **nothing**. If class has no constructor; so when the class was compiled, C++ will **generate** the **default** constructor:

Demo::Demo() { }

1.3 Constructor Overloading:

When two or more functions share the same name, the function is said to be **overloaded**.

A class's constructor may be **overloaded**. One constructor might take an **integer argument**, for example, while another constructor takes a **double**. There could even be a third constructor taking **two integers**. As long as each constructor takes a **different list of parameters**, the compiler can tell them **apart**.

Let's look at an example of how you can create **overloaded constructors**. The **InventoryItem** class holds the following **data** about an **item** that is **stored** in **inventory**:

- Item's description (a string object)
- Item's cost (a double)
- Number of units in inventory (an int)

1.3.1 Code Example:

```
       classInventoryItem
{
private:
       string description; // The item description
       double cost; // The item cost
       int units; // Number of units on hand
public:
       // Constructor #1 Default
       InventoryItem()
       {
              // Initialize description, cost, and units.
              description ="";
              cost = 0.0;
              units = 0;
       }
       // Constructor #2 Parameterized
       InventoryItem(stringdesc)
       {
              // Assign the value to description.
              description =desc;
              // Initialize cost and units.
              cost = 0.0;
```

```
                units = 0;
        }
        // Constructor #3 Parameterized
        InventoryItem(stringdesc, doublec, intu)
        { // Assign values to description, cost, and units.
                description =desc;
                cost = c;
                units = u;
        }
}
int main()
{
        InventoryItem item1;

        // Create an InventoryItem object and call
        // constructor #2.
        InventoryItem item2("Pliers");
        // Create an InventoryItem object and call
        // constructor #3.
        InventoryItem item3("Wrench", 8.75, 20);
        return 0;
}
```

1.4 Shallow Copy and Deep Copy:

Depending upon the resources like **dynamic memory** held by the **object**, either we need to perform **Shallow Copy** or **Deep Copy** in order to **create** a **replica** of the **object**. In general, if the **variables** of an object have been **dynamicallyallocated** then it is required to do a **DeepCopy** in order to **create** a **copy** of the object.

1.4.1. Shallow Copy:

In shallow copy, an **object** is **created** by simply **copying** the **data** of all **variables** of the original object.

This works well if none of the variables of the object are **defined** in the **heapsection** of memory. If some variables are **dynamicallyallocatedmemory** from heap section, then **copied** object variable will also **reference** then **same** memory location.

This will create **ambiguity** and **run-time errors** dangling pointer. Since **both** objects will **reference** to the **samememorylocation**, then change made by **one** will **reflect** those change in **anotherobject** as well. Since we wanted to create a replica of the object, this purpose will not be filled by Shallow copy.

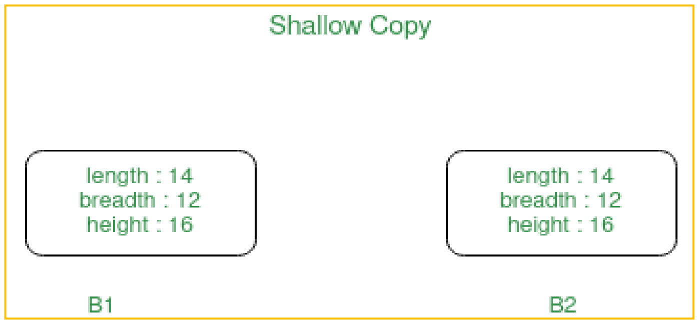


Figure 1: Shallow Copy

Shallow Copy of object if some variables are defined in heap memory, then:

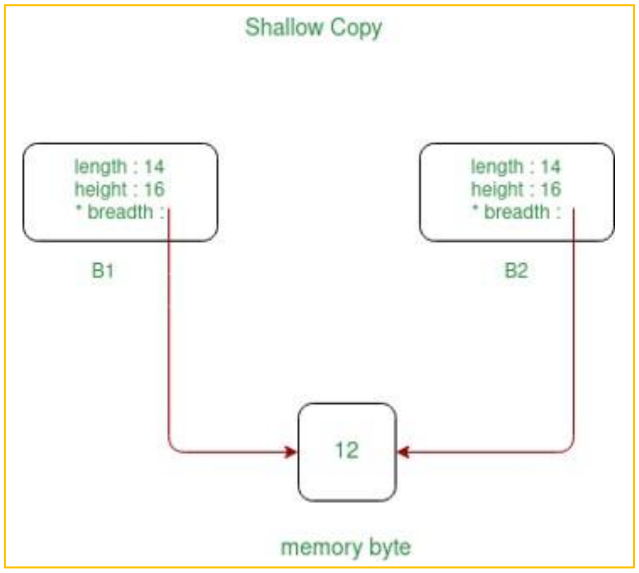


Figure 2: Shallow Copyif some variables are defined in heap memory

1.4.2. <u>Deep Copy:</u>

In Deep copy, an object is **created** by copying data of **allvariables** and it also **allocatessimilarmemoryresources** with the **samevalue** to the object. In order to perform Deep copy, we need to **explicitlydefine** the copy constructor and **assigndynamicmemory** as well if required. Also, it is required to **dynamicallyallocatememory** to the variables in the other **constructors**, as well.

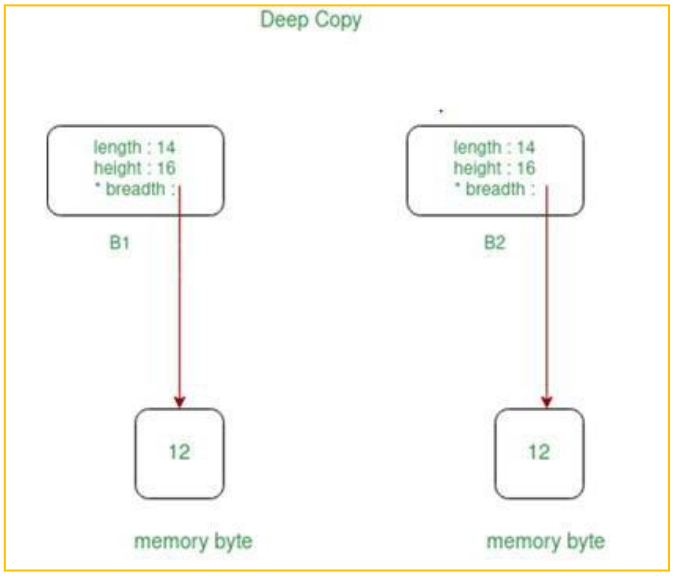


Figure 3: Deep Copy

2. Destructor

Destructor is member**function** with the **samename** as the **class**, preceded by a **tilde** character (~). For example, the destructor for the Demo class would be named **~Demo**.

Destructors are **automaticallycalled** when an object is **destroyed**. In the same way that constructors **set things up** when an object is created, destructors perform **shutdown procedures** when the object **goes out of existence**. For example, a common **use** of destructors is to **free memory** that was **dynamically** allocated by the class object.

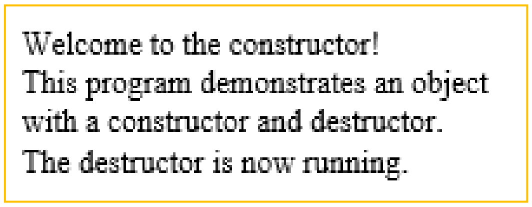
2.1 Code Example:

```
classDemo
{
public:
        Demo(); // Constructor
        ~Demo(); // Destructor
};
Demo::Demo()
{
        cout<<"Welcome to the constructor!\n";
}
Demo::~Demo()
{
        cout<<"The destructor is now running.\n";
}
//*********************************************
// Function main. *
```

```
//*********************************************
int main()
{
        DemodemoObject; // Define a demo object;

        cout<<"This program demonstrates an object\n";
        cout<<"with a constructor and destructor.\n";
        return 0;
}
```

Output:



```
Welcome to the constructor!
This program demonstrates an object
with a constructor and destructor.
The destructor is now running.
```

2.2 Code Example: Demo

```
#include<iostream>
#include<string>
#include<iomanip>
usingnamespacestd;

classInventoryItem
{
private:
        string description; // The item description
        double * cost; // The item cost
        int units; // Number of units on hand
public:
        // Constructor #1 Default
        InventoryItem()
        {
                // Initialize description, cost, and units.
                description ="";
                cost = newdouble(0);
                units = 0;
        }

        // Constructor #2 Parameterized
        InventoryItem(stringdesc)
        {
                // Assign the value to description.
                description =desc;
                // Initialize cost and units.
                cost = newdouble(0);
                units = 0;
        }

        // Constructor #3 Parameterized
        InventoryItem(stringdesc, doublec, intu)
        { // Assign values to description, cost, and units.
                description =desc;
                cost = newdouble(0);
                *cost = c;
                units = u;
        }

// Constructor #3 Copy
        // Comment and Uncomment this to see Output (Shallow vs Deep Copy)
        InventoryItem(constInventoryItem&inventoryItemObj)
        { // Assign values to description, cost, and units.
                description =inventoryItemObj.getDescription();
                cost = newdouble(0);
                *cost = inventoryItemObj.getCost();
                units = inventoryItemObj.getUnits();
        }
        // Mutator functions
        voidsetDescription(stringd)
        {
                description =d;
        }
        voidsetCost(doublec)
        {
                *cost = c;
        }
        voidsetUnits(intu)
        {
                units = u;
        }
        // Accessor functions
        stringgetDescription() const
        {
                return description;
        }
        doublegetCost() const
        {
                return *cost;
```

```
		}
		intgetUnits() const
		{
			return units;
		}
		~InventoryItem()
		{
			delete cost;
		}
};
```

```
int main()
{
        // Create an InventoryItem object and call
                // the default constructor.
        InventoryItem item1;
        item1.setDescription("Hammer"); // Set the description
        item1.setCost(6.95); // Set the cost
        item1.setUnits(12); // Set the units

        // Create an InventoryItem object and call
        // constructor #2.
        InventoryItem item2("Pliers");

        // Create an InventoryItem object and call
        // constructor #3.
        InventoryItem item3("Wrench", 8.75, 20);
        InventoryItem item4 = item3;
        cout<<"The following items are in inventory:\n";
        cout<<setprecision(2) << fixed <<showpoint;

        // Display the data for item 1.
        cout<<"Description: "<< item1.getDescription() <<endl;
        cout<<"Cost: $"<< item1.getCost() <<endl;
        cout<<"Units on Hand: "<< item1.getUnits() <<endl<<endl;

        // Display the data for item 2.
        cout<<"Description: "<< item2.getDescription() <<endl;
        cout<<"Cost: $"<< item2.getCost() <<endl;
        cout<<"Units on Hand: "<< item2.getUnits() <<endl<<endl;

        // Display the data for item 3.
        cout<<"Description: "<< item3.getDescription() <<endl;
        cout<<"Cost: $"<< item3.getCost() <<endl;
        cout<<"Units on Hand: "<< item3.getUnits() <<endl;

// Display the data for item 4.
        cout<<"Description: "<< item4.getDescription() <<endl;
        cout<<"Cost: $"<< item4.getCost() <<endl;
        cout<<"Units on Hand: "<< item4.getUnits() <<endl;
        cout<<endl<<"Shallow vs Deep Copy- Comment Copy Constructor to
see difference in output"<<endl;
        item3.setCost(20);
        cout<<"Item 3 Cost: $"<< item3.getCost() <<endl;
        cout<<"Item 4 Cost: $"<< item4.getCost() <<endl;
        return 0;
}
```

Lab Tasks

1. Create a class Named Student which can save student information containing Roll Number, First Name, Last Name, Student class, Marks (can be in points) and grade. Set default values for class members in default constructor. Initialize class and fill with user provided values. At last pass this class instance to a function named PrintData, which prints incoming information contained within student instance. Student Information to Store: Roll Number, First Name, Last Name, Student Class, Total Marks and Grade.
2. Overload default constructor for the class created in exercise 1.
   - Overload constructor with one integer argument which sets roll number value.
   - Overload constructor with three arguments for roll number first and last name.
   - Initialize student instance with default constructor, and print values on screen using PrintData function. (Created in exercise 1)
   - Initialize student instance with overloaded constructor, and print values on screen using PrintData function. (Created in exercise 1)
3. Suppose you have a Bank Account with an initial amount of $50 and you have to add some more amount to it. Create a class 'AddAmount' with a data member named 'amount' with an initial value of $50. Now make two constructors of this class as follows:
   - Without any parameter - no amount will be added to the Bank Account.
   - Having a parameter which is the amount that will be added to the Bank Account
   - Create an object of the 'AddAmount' class and display the final amount in the Bank Account.
4. Write a class declaration named Circle with a private member variable named radius. Write set and get functions to access the radius variable, and a function named getArea that returns the area of the circle. The area is calculated as 3.14159 * radius * radius 44. Add a default constructor to the Circle class, the constructor should initialize the radius member to 0.
5. Add an overloaded constructor to the Circle class in question 4. The constructor should accept an argument and assign its value to the radius member variable.
6. Write a class named Car that has the following member variables:
   - **yearModel**. An int that holds the car’s year model.
   - **make** . A string that holds the make of the car.
   - **speed** . An int that holds the car’s current speed.

   In addition, the class should have the following constructor and other member functions.
   - **Constructor**: The constructor should accept the car’s year model and make as arguments. These values should be assigned to the object’s yearModel and make member variables. The constructor should also assign 0 to the speed member variables.
   - **Accessor**: Appropriate accessor functions to get the values stored in an object’s yearModel, make, and speed member variables.

- **Accelerate**: The accelerate function should add 5 to the speed member variable each time it is called.
- **Brake:** The brake function should subtract 5 from the speed member variable each time it is called.

Demonstrate the class in a program that creates a Car object, and then calls the accelerate function five times. After each call to the accelerate function, get the current speed of the car and display it. Then, call the brake function five times. After each call to the brake function, get the current speed of the car and display it.

7. Write a class named Employee that has the following member variables

- **name**. A string that holds the employee's name.
- **idNumber**. An int variable that holds the employee's ID number.
- **department**. A string that holds the name of the department where the employee works.
- **position**. A string that holds the employee's job title.

The class should have the following constructors:

- A constructor that accepts the following values as arguments and assigns them to the appropriate member variables: employee's name, employee's ID number, department, and position.
- A constructor that accepts the following values as arguments and assigns them to the appropriate member variables: employee's name and ID number. The department and position fields should be assigned an empty string ("").
- A default constructor that assigns empty strings ("") to the name, department, and position member variables, and 0 to the ID Number member variable.
- Write the destructor, count it "Hey look I am in destructor".

Write appropriate setter functions that store values in these member variables and getter functions that return the values in these member variables. Once you have written the class, write a separate program that creates three Employee objects to hold the following data.

Name	ID Number	Department		Position
Susan Meyers	47899	Accounting		Vice President
Mark Jones	39119	IT		Programmer
Joy Rogers	81774	Manufacturing		Engineer

The program should store this data in the three objects and then display the data for each employee on the screen.