

Course Code: CL-1004	Course : Object Oriented Programming Lab
Instructor(s) :	Anam Qureshi

Lab # 04

### Outline

- Introduction to Constructor
- Types of constructor
- Destructor

## INTRODUCTION TO Constructor

### Constructor and its Types

- **Constructor** is the special type of member function in C++ classes, which are automatically invoked when an object is being created. It is special because its name is same as the class name.
- **To initialize data member of class:** In the constructor member function (which will be declared by the programmer) we can initialize the default vales to the data members and they can be used further for processing.
- **To allocate memory for data member:** Constructor can also be used to declare run time memory (dynamic memory for the data members).
- Constructor has the same name as the class name. It is case sensitive.
- Constructor does not have return type.
- We can overload constructor, it means we can create more than one constructor of class.
- It must be public type.

### Types of Constructors

- **Default Constructors:** Default constructor is the constructor which doesn't take any argument. It has no parameters.
- **Parameterized Constructors:** It is 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.
- **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.

### Destructors

- 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.
- The 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.

## Data Members & its Types

- **Data Members:** The variables which are declared in any class by using any fundamental data types (like int, char, float etc) or derived data type (like class, structure, pointer etc.) are known as Data Members.

### Types of Data Members

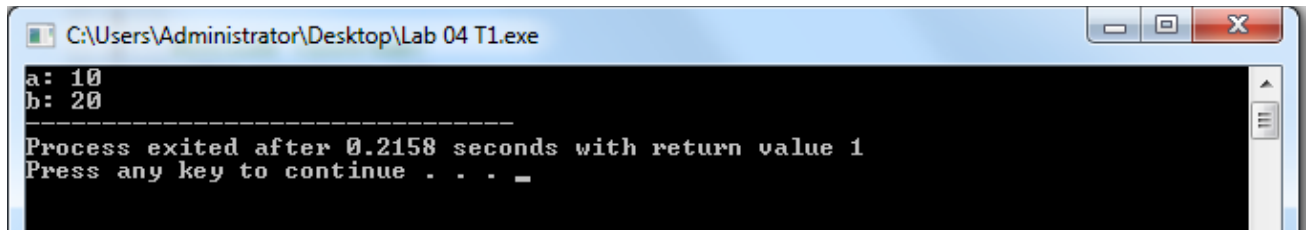
- **Private members:** The members which are declared in private section of the class (using private access modifier) are known as private members. Private members can also be accessible within the same class in which they are declared.
- **Public members:** The members which are declared in public section of the class (using public access modifier) are known as public members. Public members can access within the class and outside of the class by using the object name of the class in which they are declared.

### Sample C++ Code:

#### **Code#1 (Default Constructors)**

```
#include <iostream>
using namespace std;
class construct
{
public:
    int a, b;
    construct()
    {
        a = 10;
        b = 20;
    }
};

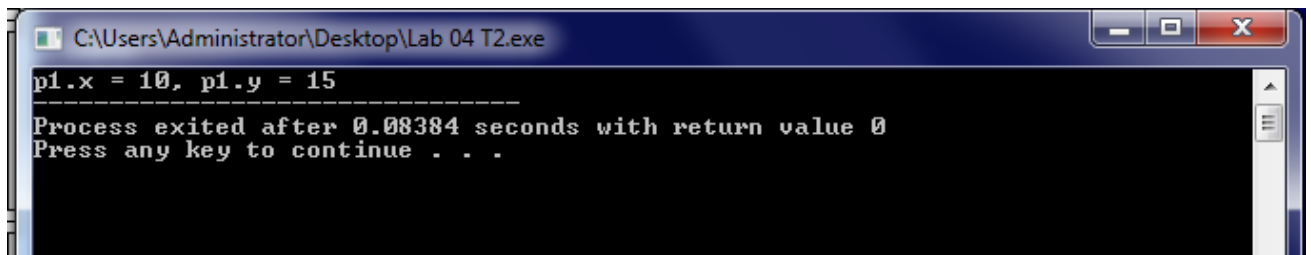
int main()
{
    construct c;
    cout << "a: " << c.a << endl
        << "b: " << c.b;
    return 1;
}
```



```
C:\Users\Administrator\Desktop\Lab 04 T1.exe
a: 10
b: 20
-----
Process exited after 0.2158 seconds with return value 1
Press any key to continue . . . _
```

## Code#2 (Parameterized Constructors)

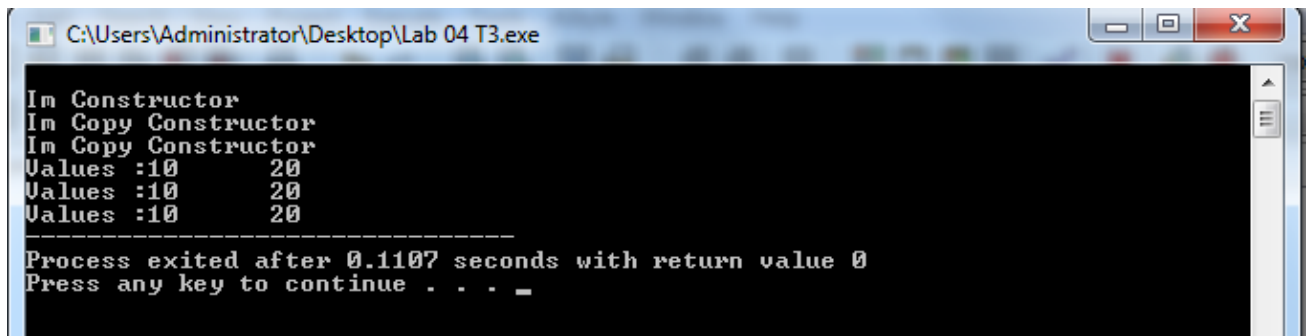
```
#include <iostream>
using namespace std;
class Point
{
private:
    int x, y;
public:
    Point(int x1, int y1)
    {
        x = x1;
        y = y1;
    }
    int getX()
    {
        return x;
    }
    int getY()
    {
        return y;
    }
};
int main()
{
    Point p1(10, 15);
    cout << "p1.x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



```
C:\Users\Administrator\Desktop\Lab 04 T2.exe
p1.x = 10, p1.y = 15
-----
Process exited after 0.00384 seconds with return value 0
Press any key to continue . . .
```

### Code#3 (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;
}
```



```
C:\Users\Administrator\Desktop\Lab 04 T3.exe

Im Constructor
Im Copy Constructor
Im Copy Constructor
Values :10      20
Values :10      20
Values :10      20
-----
Process exited after 0.1107 seconds with return value 0
Press any key to continue . . . _
```

### Copy Constructor

- creating a copy of an object means to create an exact replica of the object having the same literal value, data type, and resources.

### Syntax

- *// Copy Constructor*  
*Test Obj1(Obj);*  
*or*  
*Test Obj1 = Obj;*
- *// Default assignment operator*  
*Test Obj2;*  
*Obj2 = Obj1;*

### Initializing one object with another

```
class A{
    int val;
    public:
    A(int val1) {
        val = val1; }
    A(){ }
    void setVal(int val1) { val = val1; }
    void showVal() { cout << "Value: " << val << endl; }
};

int main()
{
    A a1( 10 );
    a1.showVal( );
    A a2 = a1;
    a2.showVal( );
    a2.setVal( 20 );
    a1.showVal( );
    a2.showVal( );
}
```

### OUTPUT:

Val: 10 a1.val

Val: 10 a2.val

Val: 10 a1.val

Val: 20 a2.val

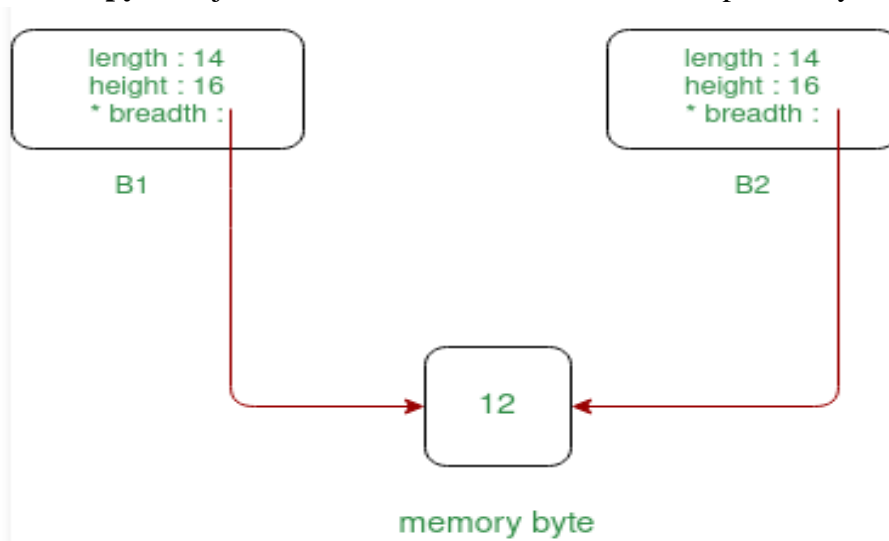
- If we don't define our own copy constructor, the compiler creates a default copy constructor for each class
- The default copy constructor performs member-wise copy between objects
- Default copy constructor works fine unless an object has pointers or any runtime allocation

### User define copy constructor

ClassName (const ClassName &ob);

### **Shallow Copy**

- Default constructor always perform a *shallow copy*
- In shallow copy, an object is created by simply copying the data of all variables of the original object.
- **Shallow Copy** of object if some variables are defined in heap memory, then:



Example:

```
#include <iostream>
using namespace std;
class box {
    int length;
    int* breadth;
    int height;
public:
    box() {
        breadth = new int; }
    void set_dimension(int len, int brea, int heig) {
        length = len;
        *breadth = brea;
        height = heig; }
    void show_data() {
        cout << " Length = " << length
            << "\n Breadth = " << *breadth
            << "\n Height = " << height
            << endl; }
};
// Driver Code
int main()
{
    // Object of class first
    box first;
    // Set the dimensions
    first.set_dimension(12, 14, 16);
    // Display the dimensions
    first.show_data();
    box second = first;
    // Display the dimensions
    second.show_data();
    second.set_dimension(1, 1, 6);
    first.show_data();
    second.show_data();
    return 0;
}
```

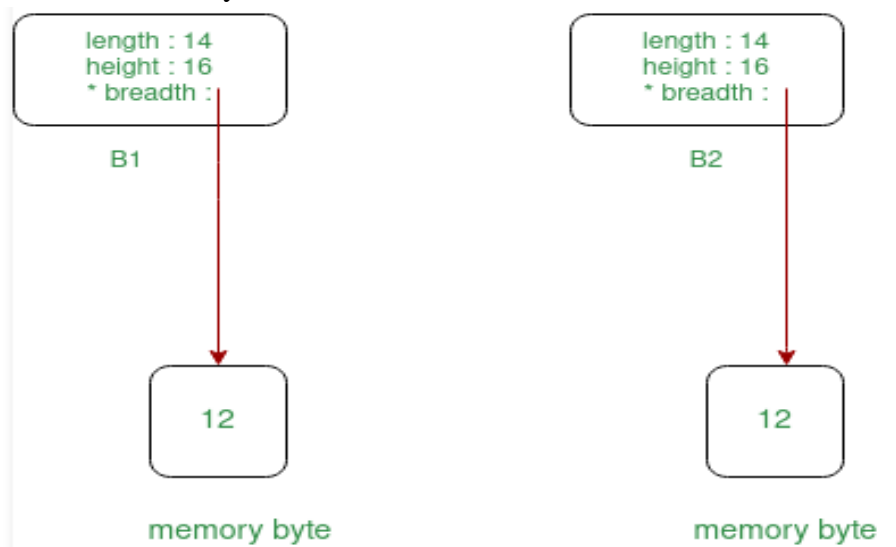
C:\Users\Nida\Documents\Untitled1.exe

```
Length = 12
Breadth = 14
Height = 16
Length = 12
Breadth = 14
Height = 16
Length = 12
Breadth = 1
Height = 16
Length = 1
Breadth = 1
Height = 6

-----
Process exited after 6.282 seconds with return value 0
Press any key to continue . . .
```

### Deep Copy

- Deep copy is only possible with user-defined copy constructors
- In user-defined copy constructors, we make sure that pointers (or references) of copied object point to new memory locations






Example:

```
#include <iostream>
using namespace std;
class box {
    int length;
    int* breadth;
    int height;
public:
    box() {
        breadth = new int; }
    void set_dimension(int len, int brea, int heig) {
        length = len;
        *breadth = brea;
        height = heig; }
    void show_data() {
        cout << " Length = " << length
            << "\n Breadth = " << *breadth
            << "\n Height = " << height
            << endl; }
    box(box& sample)
    {
        length = sample.length;
        breadth = new int;
        *breadth = *(sample.breadth);
        height = sample.height;
    }
    // Destructors
    ~box()
    {
        delete breadth;
    }
};
```

```
// Driver Code
int main()
{
    // Object of class first
    box first;
    // Set the dimensions
    first.set_dimension(12, 14, 16);
    // Display the dimensions
    first.show_data();
    box second = first;
    // Display the dimensions
    second.show_data();
    second.set_dimension(1, 1, 6);
    first.show_data();
    second.show_data();
    return 0;
}
```

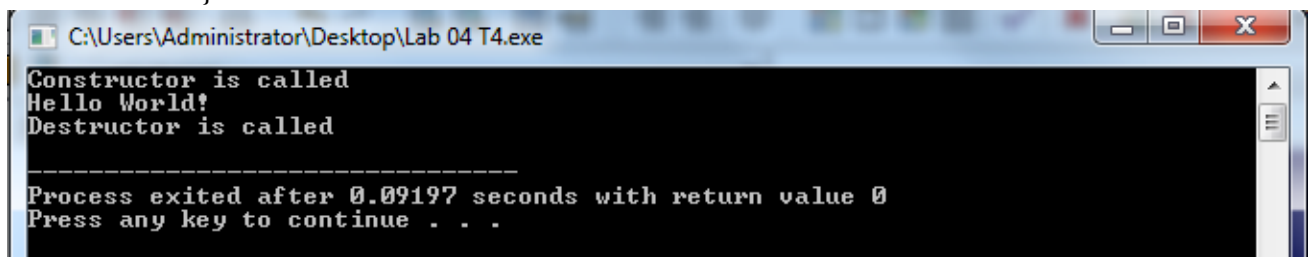
 C:\Users\Nida\Documents\Untitled1.exe

```
Length = 12
Breadth = 14
Height = 16
Length = 12
Breadth = 14
Height = 16

-----
Process exited after 0.02923 seconds with return value 0
Press any key to continue . . .
```

### Example (Destructors)

```
#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;
}
```

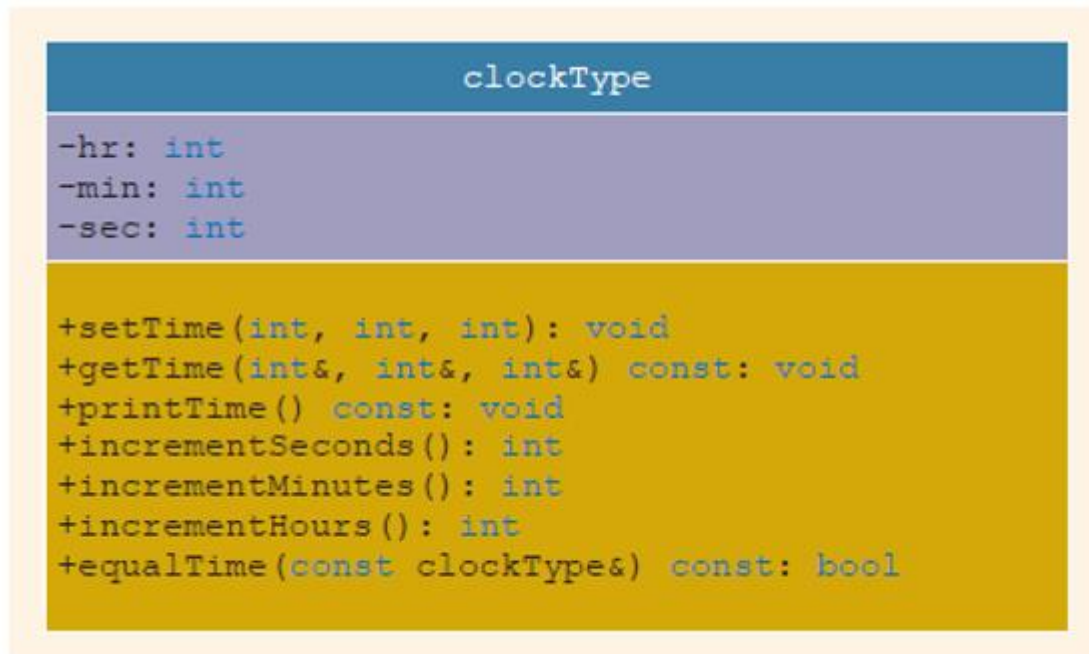


```
C:\Users\Administrator\Desktop\Lab 04 T4.exe
Constructor is called
Hello World!
Destructor is called

-----
Process exited after 0.09197 seconds with return value 0
Press any key to continue . . .
```

## UML FOR CLASS DIAGRAMS

A class and its members can be described graphically using a notation known as the Unified Modeling Language (UML) notation. For example, Figure 1 shows the UML class diagram of the class `clockType`.



### Exercise Lab 04

#### Question # 01:

Design and implement a class **dayType** that implements the day of the week in a program. The class **dayType** should store the day, such as **Sun for Sunday**. The program should be able to perform the following operations on an object of type **dayType**:

- Set the day.
- Print the day.
- Return the day.
- Return the next day.
- Return the previous day.
- Calculate and return the day by adding certain days to the current day. For example, if the current day is Monday and we add 4 days, the day to be returned is Friday. Similarly, if today is Tuesday and we add 13 days, the day to be returned is Monday.
- Add default, parametrized and copy constructors.

#### Question # 02:

Define the class **personType** so that, in addition to what the existing class does, you can:

- Set the first name only.
- Set the last name only.
- Store and set the middle name.
- Check whether a given first name is the same as the first name of this person.
- Check whether a given last name is the same as the last name of this person. Write the definitions of the member functions to implement the operations for this class. Also, write a program to test various operations on this class.

#### Question # 03:

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.

#### **Question # 04:**

In this exercise, you will design a class **memberType**.

- Each object of **memberType** can hold the name of a person, member ID, number of books bought, and amount spent.
- Include the member functions to perform the various operations on the objects of **memberType** — for example, modify, set, and show a person's name. Similarly, update, modify, and show the number of books bought and the amount spent.
- Add the appropriate constructors.
- Write the definitions of the member functions of **memberType**.
- Write a program to test various operations of your class **memberType**.

#### **Question # 05:**

Using the classes designed in Lab Task 03 and 04, write a program to simulate a bookstore. The bookstore has two types of customers: those who are members of the bookstore and those who buy books from the bookstore only occasionally.

- Each member has to pay a \$10 yearly membership fee and receives a 5% discount on each book purchased.
- For each member, the bookstore keeps track of the number of books purchased and the total amount spent.
- For every eleventh book that a member buys, the bookstore takes the average of the total amount of the last 10 books purchased, applies this amount as a discount, and then resets the total amount spent to 0.

Write a program that can process up to 1000 book titles and 500 members. Your program should contain a menu that gives the user different choices to effectively run the program; in other words, your program should be user driven.