University of Management and Technology

Knowledge Unit of Science and Technology

Laboratory Manual

[CC1022L]: [Object Oriented Programming]

[Semester Spring-2023]

Class: [BSSE, BSAI, BSCS, BSIT]

Lab [3]: [Constructors & Destructors]

Instructor: [Ms. Fatima Wajahat]

University of Management and Technology

Constructors & Destructors:

Objectives:

- What is constructor?
- Why we use constructors?
- Difference between constructors and methods
- Types of constructors
  1. Default constructor
  2. Parametrized constructor
     - 2.1. Constructor overloading
  3. Copy constructor
     - 3.1. Shallow copy constructors
     - 3.2. Deep copy constructors
- What are destructors?
- Why we use destructors?

- **What are constructors?**

Constructors:

A constructor is a special "member function" in C++ that has the same name as the class it belongs to. A constructor in C++ is a special method that is automatically called when an object of a class is created. The constructor has the same name as the class, it is always public, and it does not have any return value.

Constructors definition inside and outside the class:

Inside the class:

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

class room
{
    int length;
    int width;
    public:
    room ()
    {
        //definition
    }
};

int main ()
{
    room r;
}
```

Outside the class:

The scope resolution operator is used as an identifier to define the constructors outside the class:

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

class room
{
    int length;
    int width;
    public:
        room();
};

room::room()
{
    //definition
}

int main ()
{
    room r;
}
```

Example:

```
class MyClass {       // The class
  public:             // Access specifier
    MyClass() {       // Constructor
      cout << "Hello World!";
    }
};

int main() {
  MyClass myObj;     // Create an object of MyClass (this will call the constructor)
  return 0;
}
```

- **Why we use constructor?**

Use of constructors:

Constructors are used to initialize some useful values for an object's data members. The constructor is used to initialize values and is automatically called by the compiler. A constructor enables you to provide any custom initialization that must be done before any other methods can be called.

- **Difference between constructors and methods:**

- A constructor helps in initializing an object whereas a method is a grouping of instructions that returns a value upon its execution.
- Constructors does not have a return type whereas functions have a return type.
- The name of the constructor and class will always be the same whereas for the method, we can use any name.
- A Constructor helps in initializing an object that doesn't exist whereas A Method performs functions on pre-constructed or already developed objects.
- Constructors can't be inherited by subclasses whereas methods can be inherited by subclasses.

4. Types of Constructors:

There are 3 types of constructors:

1. Default Constructor
2. Parametrized Constructor
3. Copy Constructor

1. Default Constructor

A constructor with no parameters is known as a **default constructor**. In the example given below, Wall () is a default constructor.

Code:

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

// declare a class
class  Wall {
  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;
}
```

Output:

```
Creating a wall.
Length = 5.5

--------------------------------
Process exited after 0.08236 seconds with return value 0
Press any key to continue . . .
```

2. Parametrized Constructor:

In C++, a constructor with parameters is known as a parameterized constructor.

Code:

```cpp
#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;
}
```

Output:

```
Area of Wall 1: 90.3
Area of Wall 2: 53.55
--------------------------------
Process exited after 0.08477 seconds with return value 0
Press any key to continue . . .
```

2.1. Constructor Overloading:

Overloaded constructors have the same name (name of the class) but the different number of arguments. Depending upon the number and type of arguments passed, the corresponding constructor is called.

Code:

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

class Wall {
  private:
    double length;
    double height;
    double width;

  public:
    Wall(double len, double hgt) {
      length = len;
      height = hgt;
    }
    Wall (double len, double hgt, double wdt)
    {
        length=len;
        height=hgt;
        width=wdt;
    }
    double calculateArea() {
      return length * height;
    }
    double calculateArea1() {
      return length * height * width;
    }
};

int main() {
  Wall wall1(10.5, 8.6);
  Wall wall2(8.5, 6.3, 4.5);

  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall 2: " << wall2.calculateArea1()<<endl;

  return 0;}
```

Output:

```
Area of Wall 1: 90.3
Area of Wall 2: 240.975

--------------------------------
Process exited after 0.07257 seconds with return value 0
Press any key to continue . . .
```

3. Copy Constructor:

A copy constructor is a member function that initializes an object using another object of the same class. In simple terms, a constructor which creates an object by initializing it with an object of the same class, which has been created previously is known as a copy constructor.

It has two types:

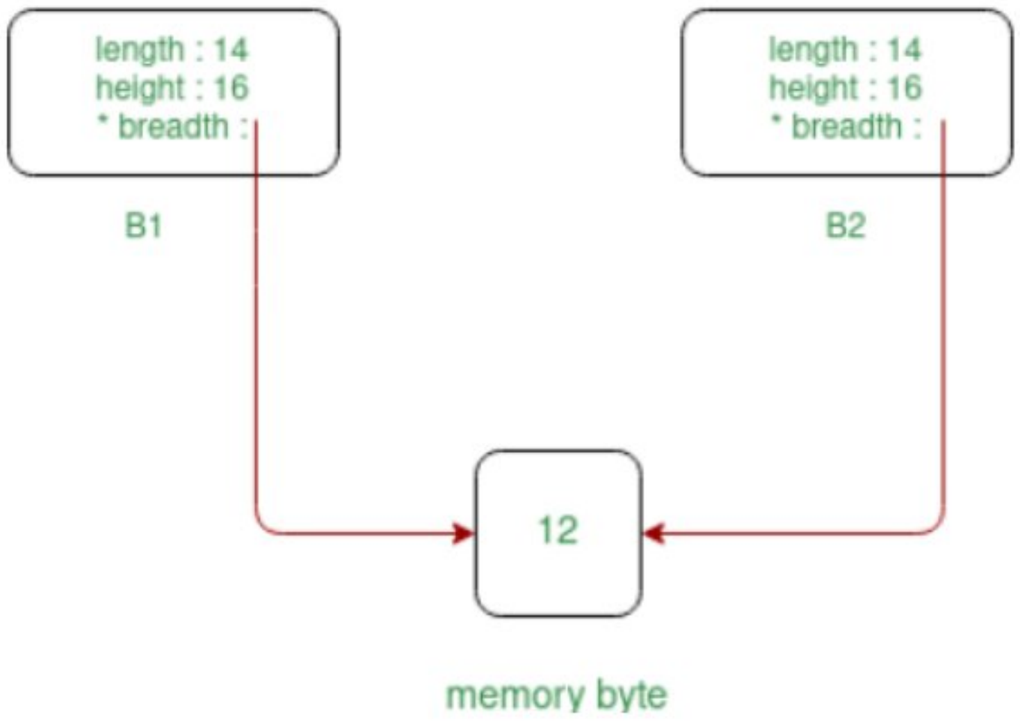
1. Shallow copy constructors
2. Deep copy constructors

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.

Shallow Copy Constructors:

n 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 heap section of memory. If some variables are dynamically allocated memory from heap section, then the copied object variable will also reference the same memory location.

If the objects variables are declared in heap section of memory then the following problem occur.



Example: Simple Copy

Code:

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

class Wall {
  private:
    double length;
    double height;
    double width;
  public:
    Wall(double len, double hgt) {
      length = len;
      height = hgt;
    }
    Wall (Wall &wall1)
    {
        length=wall1.length;
        height=wall1.height;
    }
    double calculateArea() {
      return length * height;
    }
};
int main() {
  Wall wall1(10.5, 8.6),wall2(wall1);
 // Wall wall2=wall1; // another method is by using assignment operator

  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall 2: " << wall2.calculateArea()<<endl;
  return 0;
  }
```

Output:

```
Area of Wall 1: 90.3
Area of Wall 2: 90.3

--------------------------------
Process exited after 0.07244 seconds with return value 0
Press any key to continue . . .
```

Example: Copy with updating

Code:

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

class Wall {
  private:
    double length;
    double height;
    double width;

  public:
    Wall(double len, double hgt) {
      length = len;
      height = hgt;
    }
    Wall (Wall &wall1) //copy with updation
    {
        length=wall1.length+1;//+1 is to update the value of length
        height=wall1.height+1;//+1 is to update the value of height
    }
    double calculateArea() {
      return length * height;
    }
};

int main() {
  Wall wall1(10.5, 8.6),wall2(wall1);
 // Wall wall2=wall1; // another method is by using assignment operator


  cout << "Area of Wall 1: " << wall1.calculateArea() << endl;
  cout << "Area of Wall 2: " << wall2.calculateArea()<<endl;

  return 0;}
```

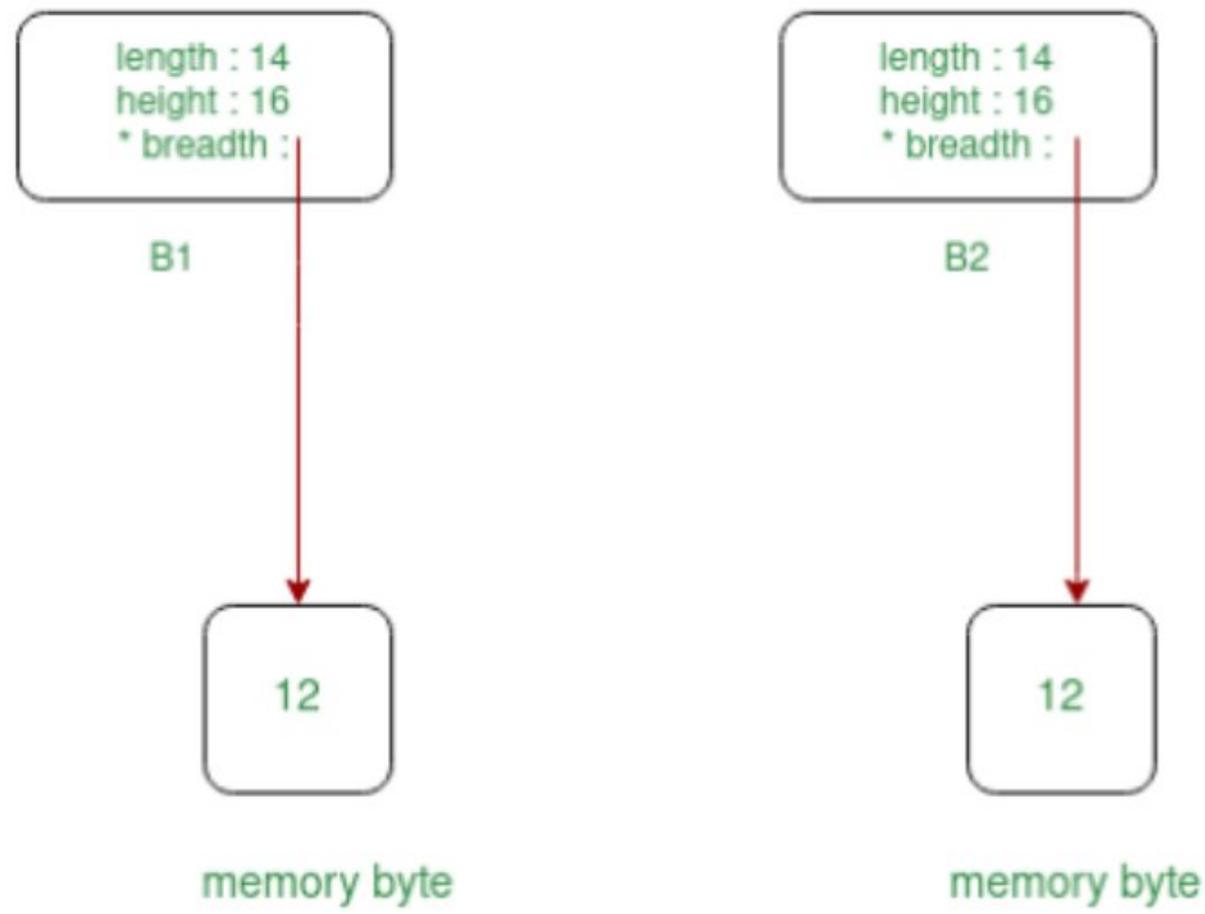
Output:

```
Area of Wall 1: 90.3
Area of Wall 2: 110.4

--------------------------------
Process exited after 0.06818 seconds with return value 0
Press any key to continue . . .
```

Deep Copy Constructor:

In Deep copy, an object is created by copying data of all variables, and it also allocates similar memory resources with the same value to the object. In order to perform Deep copy, we need to explicitly define the copy constructor and assign dynamic memory as well, if required. Also, it is required to dynamically allocate memory to the variables in the other constructors, as well.



Code:

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

class calculate
{
    int a;
    int b;
    int *p;
    public:
    calculate()
    {
        a=30;
        b=90;
        p=&b;
    }
    calculate (calculate &cal) //copy with updation and assigned new momory location
    {
        a=cal.a+1;
        p=new int; //new keyword is used to assigned new memory location and the type of the data stored in memory is of int type
        *p=*(cal.p)+1;
    }
    void display ()
    {
        cout<<"The sum of the two numbers is "<<a+*p<<endl;
    }
};
int main ()
{
    calculate cal,cal1(cal);
    cal.display();
    cal1.display();
    return 0;
}
```

Output:

```
The sum of the two numbers is 120
The sum of the two numbers is 122

--------------------------------

Process exited after 0.07476 seconds with return value 0
Press any key to continue . . .
```

- **What are destructors?**

  A destructor is a member function that is invoked automatically when the object goes out of scope or is explicitly destroyed by a call to delete. A destructor has the same name as the class, preceded by a tilde (~).

- **Why we use destructors:**

  Destructors are usually used to deallocate memory and do other cleanup for a class object and its class members when the object is destroyed.

- **Example:**

Code:

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

// declare a class
class  Wall {
  private:
    double length;

  public:
    // default constructor to initialize variable
    Wall() {
      length = 5.5;
      cout << "Creating a wall." << endl;
      cout << "Length = " << length << endl;
    }
    // destructor to deallocate memory
    ~ Wall()
    {
        cout<<"The destructor is called"<<endl;
    }
};

int main() {
  Wall wall1;
  return 0;
}
```

Output:

```
Creating a wall.
Length = 5.5
The destructor is called

--------------------------------
Process exited after 0.06687 seconds with return value 0
Press any key to continue . . .
```

Lab Tasks:

LAB Task 1:

Create a class named 'Programming'. While creating an object of the class, if nothing is passed to it, then the message "I love programming languages" should be printed. If some String is passed to it, then in place of "programming languages" the name of that String variable should be printed. **For example**, while creating object if we pass "Java", then "I love Java" should be printed.

LAB Task 2:

Write a program to print the name of student by creating a Student class. If no name is passed while creating an object of Student class, then the name should be "Unknown", otherwise the name should be equal to the string value passed while creating object of Student class.

LAB Task 3:

Create a class named 'Rectangle' with two data members- length and breadth and a method to calculate the area which is length*breadth. The class has three constructors which are:

1 - having no parameter - values of both length and breadth are assigned zero.

2 - having two numbers as parameters - the two numbers are assigned as length and breadth respectively.

3 - having one number as parameter - both length and breadth are assigned that number.

4 – also create the destructor to deallocate memory.

Now, create objects of the 'Rectangle' class having none, one and two parameters and print their areas.

Lab Task 4:

Create a class of triangle having two constructors one is default or parametrized (it depends on you) and the other one is shallow copy constructor you have to update the value of base and height in copy constructor and print the area using both constructors the formula used to calculate area is 1/2 (base*height). And also create a deep copy constructor by taking height as a pointer. And repeat the same updation with it.