National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department Of Electrical Engineering and Computer Sciences

Instructor: Mehreen Tahir **Date:** 13th November 2023

Lab Engineer: Mehwish Kiran **Time:** 10:00am – 12:50pm

CS 212: Object Oriented Programming

Lab 07: Inheritance

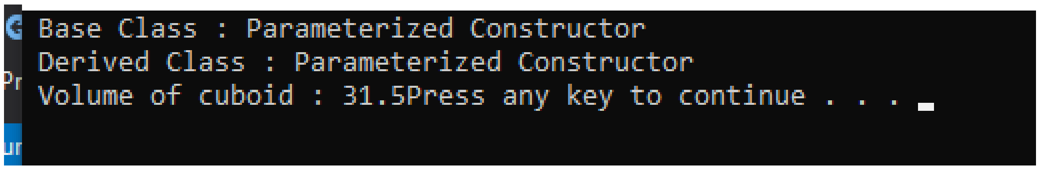
Information	Description
Name:	Irfa Farooq
CMS ID:	412564
Class:	BEE-14
Section:	D
Tenure:	Fall 2023

Task 1: Understand and execute the given example.

Code:

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

class Cuboid {
private:
    double length;

protected:
    double width;

public:
    double height;

    Cuboid() : length(0), width(0), height(0)
    {

        cout << " Base Class : Default Constructor " << endl;
    }

    Cuboid(double a, double b, double c) : length(a), width(b), height(c)
    {
        cout << " Base Class : Parameterized Constructor " << endl;
    }

    void setData(double a, double b, double c)
    {
        length = a;
        width = b;
        height = c;
    }
    double getLength() const {
        return length;
    }

    void displayData(double a, double b, double c)
    {
        cout << "length = " << length << endl;
        cout << "width = " << width << endl;
        cout << "height = " << height << endl;   }
    ~Cuboid()
    {
        cout << " Base Class : Destructor " << endl;
    }
};
class myCuboid : public Cuboid {
public:
    myCuboid()
    {
        cout << "  Derived Class: Default Constructor " << endl;
    }

    myCuboid(double a, double b, double c) : Cuboid(a, b, c)
    {
        cout << " Derived Class : Parameterized Constructor " << endl;
    }

    void calculateVolume()
    {
        cout << " Volume of cuboid : " << getLength() * width * height;
    }

    ~myCuboid()
    {
        cout << " Derived Class : Destructor " << endl;
    }
};
int main()
{
    myCuboid object(2.5, 3.0, 4.2);
    object.calculateVolume();
    system("pause");
    return 0;
}
```

Output Screenshots



```
Base Class : Parameterized Constructor
Derived Class : Parameterized Constructor
Volume of cuboid : 31.5Press any key to continue . . .
```

Task 2

Verify the *three cases* of *Base Class Access*:

Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

a. Try to access the three data members of base class in derived class (in some function) for each of three cases, and if access is not possible (in any case), try to find a solution. Note: do not change the access of data members: length, width and height in base class.

b. Then try to access the data members of base class in main (you can display them in main using cout)

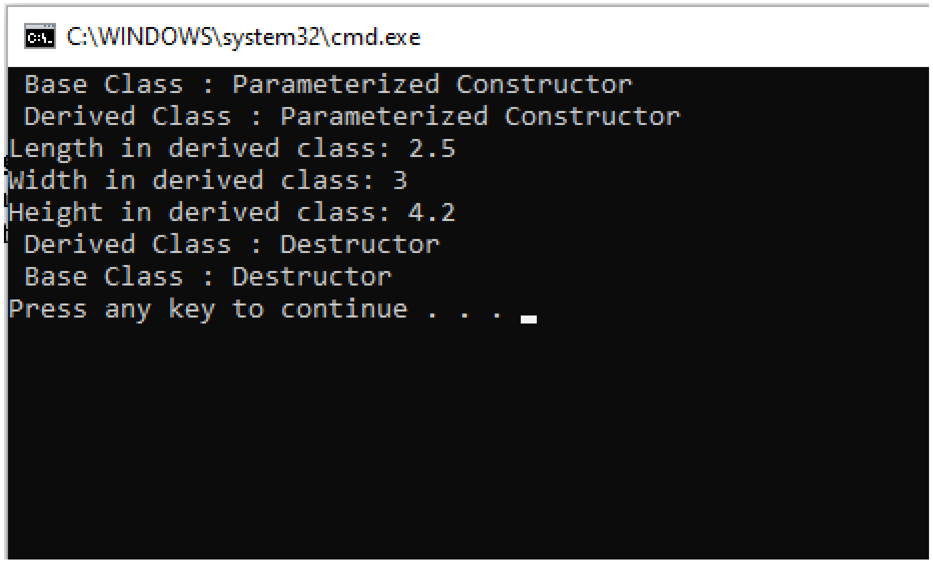
Part a:

c. **Using Public , Private and Protected visibility mode:**

Code:

```
#include <iostream>

using namespace std;

class Cuboid {
private:
    double length;

protected:
    double width;

public:
    double height;

    Cuboid() : length(0), width(0), height(0) {
        cout << " Base Class : Default Constructor " << endl;
    }
    Cuboid(double a, double b, double c) : length(a), width(b), height(c) {
        cout << " Base Class : Parameterized Constructor " << endl;
    }


    void setData(double a, double b, double c) {
        length = a;
        width = b;
        height = c;
    }


    double getLength() const {
        return length;
    }

    void displayData() {
        cout << "length = " << length << endl;
        cout << "width = " << width << endl;
        cout << "height = " << height << endl;
    }

    ~Cuboid() {
        cout << " Base Class : Destructor " << endl;
    }
};

class myCuboid : public Cuboid {
public:
    myCuboid() {
        cout << "  Derived Class: Default Constructor " << endl;
    }

    myCuboid(double a, double b, double c) : Cuboid(a, b, c) {
        cout << " Derived Class : Parameterized Constructor " << endl;
    }

    // Accessing base class members in the derived class
    void accessBaseData() {
        // Case 1: Accessing private member in derived class (not allowed)
        // Solution: Use the getter function in the base class
        double len = getLength();
        cout << "Length in derived class: " << len << endl;

        // Case 2: Accessing protected member in derived class
        double wid = width;
        cout << "Width in derived class: " << wid << endl;

        // Case 3: Accessing public member in derived class
        double hei = height;
        cout << "Height in derived class: " << hei << endl;
    }

    ~myCuboid() {
        cout << " Derived Class : Destructor " << endl;
    }
};

int main() {
    myCuboid object(2.5, 3.0, 4.2);
    object.accessBaseData();

    return 0;
}
```

Output Screenshots



```
C:\WINDOWS\system32\cmd.exe
 Base Class : Parameterized Constructor
 Derived Class : Parameterized Constructor
Length in derived class: 2.5
Width in derived class: 3
Height in derived class: 4.2
 Derived Class : Destructor
 Base Class : Destructor
Press any key to continue . . .
```

Part b:

Using public visibility mode:

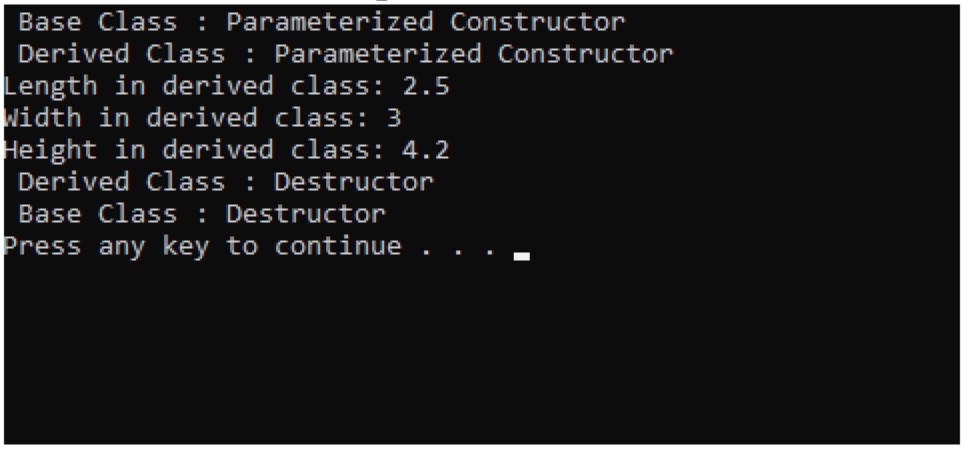
Code:

```
#include <iostream>

using namespace std;

class Cuboid {
private:
    double length;

protected:
    double width;

public:
    double height;

    Cuboid() : length(0), width(0), height(0) {
        cout << " Base Class : Default Constructor " << endl;
    }

    Cuboid(double a, double b, double c) : length(a), width(b), height(c) {
        cout << " Base Class : Parameterized Constructor " << endl;
    }

    void setData(double a, double b, double c) {
        length = a;
        width = b;
        height = c;
    }

    double getLength() const {
        return length;
    }

    double getWidth() const {
        return width;
    }
```

```cpp
    void displayData() {
        cout << "length = " << length << endl;
        cout << "width = " << width << endl;
        cout << "height = " << height << endl;
    }

    ~Cuboid() {
        cout << " Base Class : Destructor " << endl;
    }
};

class myCuboid : public Cuboid {
public:
    myCuboid() {
        cout << "  Derived Class: Default Constructor " << endl;
    }

    myCuboid(double a, double b, double c) : Cuboid(a, b, c) {
        cout << " Derived Class : Parameterized Constructor " << endl;
    }

    // Accessing base class members in the derived class
    void accessBaseData() {
        // Case 1: Accessing private member in derived class (not allowed)
        // Solution: Use the getter function in the base class
        double len = getLength();
        cout << "Length in derived class: " << len << endl;

        // Case 2: Accessing protected member in derived class
        double wid = width;
        cout << "Width in derived class: " << wid << endl;

        // Case 3: Accessing public member in derived class
        double hei = height;
        cout << "Height in derived class: " << hei << endl;
    }

    ~myCuboid() {
        cout << " Derived Class : Destructor " << endl;
    }
};
```

```
int main() {
    myCuboid object(2.5, 3.0, 4.2);

    // Access private member in main using the getter function
    cout << "Length in main: " << object.getLength() << endl;

    // Access protected member in main using the accessor function
    cout << "Width in main: " << object.getWidth() << endl;

    // Access public member in main
    cout << "Height in main: " << object.height << endl;

    return 0;
}
```

Output Screenshots



```
 Base Class : Parameterized Constructor
 Derived Class : Parameterized Constructor
Length in derived class: 2.5
Width in derived class: 3
Height in derived class: 4.2
 Derived Class : Destructor
 Base Class : Destructor
Press any key to continue . . .
```

Using protected visibility mode:

Code:

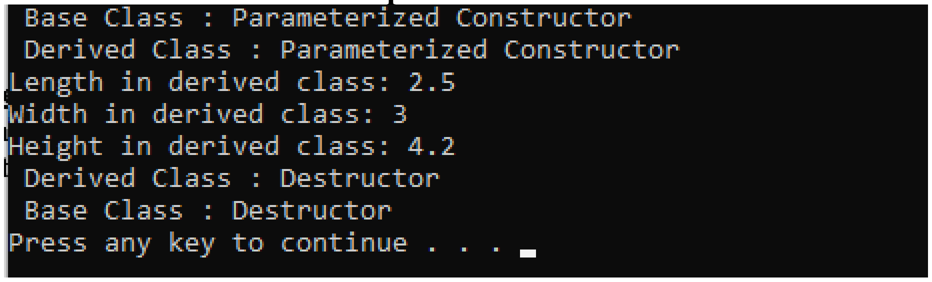
```
#include<iostream>
using namespace std;
class Cuboid {
private:
    double length;

protected:
    double width;
public:
    double height;

    Cuboid() : length(0), width(0), height(0)
    {

        cout << " Base Class : Default Constructor " << endl;
    }

    Cuboid(double a, double b, double c) : length(a), width(b), height(c)
    {
        cout << " Base Class : Parameterized Constructor " << endl;
    }

    void setData(double a, double b, double c)
    {
        length = a;
        width = b;
        height = c;
    }

    void displayData(double a, double b, double c)
    {
        cout << "length = " << length << endl;
        cout << "width = " << width << endl;
        cout << "height = " << height << endl;
    }
    double getlength(){
        return  length;
    }
    double getwidth(){
        return  width;
    }
    ~Cuboid()
    {
        cout << " Base Class : Destructor " << endl;
    }
};
class myCuboid : protected Cuboid {
public:
    myCuboid()
    {
        cout << "  Derived Class: Default Constructor " << endl;
    }

    myCuboid(double a, double b, double c) : Cuboid(a, b, c)
    {
        cout << " Derived Class : Parameterized Constructor " << endl;
    }
    double get_length(){
        return getlength();
    }
    double get_width(){
        return  getwidth();
    }
    double getheight(){
        return  height;
    }
    void calculateVolume()
    {
        cout << " Volume of cuboid : " << getlength()* width * height;
    }

    ~myCuboid()
    {
        cout << " Derived Class : Destructor " << endl;
    }
};
int main() {
    myCuboid object(2.5, 3.0, 4.2);
    // Access private member in main using the getter function
    cout << "Length in main: " << object.get_length() << endl;

    // Access protected member in main using the accessor function
    cout << "Width in main: " << object.get_width() << endl;

    // Access public member in main
    cout << "Height in main: " << object.getheight() << endl;
    system("pause");
    return 0;
}
```

Using private visibility mode:

Code:

```
#include<iostream>
using namespace std;
class Cuboid {
private:
    double length;

protected:
    double width;
public:
    double height;

    Cuboid() : length(0), width(0), height(0)
    {

        cout << " Base Class : Default Constructor " << endl;
    }

    Cuboid(double a, double b, double c) : length(a), width(b), height(c)
    {
        cout << " Base Class : Parameterized Constructor " << endl;
    }

    void setData(double a, double b, double c)
    {
        length = a;
        width = b;
        height = c;
    }
    void displayData(double a, double b, double c)
    {
        cout << "length = " << length << endl;
        cout << "width = " << width << endl;
        cout << "height = " << height << endl;
    }
    double getlength(){
        return  length;
    }
    double getwidth(){
        return  width;
    }
    ~Cuboid()
    {
        cout << " Base Class : Destructor " << endl;
    }
};
class myCuboid : private Cuboid {
public:
    myCuboid()
    {
        cout << "  Derived Class: Default Constructor " << endl;
    }

    myCuboid(double a, double b, double c) : Cuboid(a, b, c)
    {
        cout << " Derived Class : Parameterized Constructor " << endl;
    }
    double get_length(){
        return getlength();
    }
    double get_width(){
        return  getwidth();
    }
    double getheight(){
        return  height;
    }
    void calculateVolume()
    {
        cout << " Volume of cuboid : " << getlength()* width * height;
    }
    ~myCuboid()
    {
        cout << " Derived Class : Destructor " << endl;
    }
};
int main() {
    myCuboid object(2.5, 3.0, 4.2);

    // Access private member in main using the getter function
    cout << "Length in main: " << object.get_length() << endl;

    // Access protected member in main using the accessor function
    cout << "Width in main: " << object.get_width() << endl;

    // Access public member in main
    cout << "Height in main: " << object.getheight() << endl;
    system("pause");
    return 0;
}
```

Output Screenshots



```
 Base Class : Parameterized Constructor
 Derived Class : Parameterized Constructor
Length in derived class: 2.5
Width in derived class: 3
Height in derived class: 4.2
 Derived Class : Destructor
 Base Class : Destructor
Press any key to continue . . .
```

Conclusion:

In this lab, we were able to grasp the concepts of inheritance and how the change of public, private and protected class inheritance changes the access of data within the base class. According to such changes, we were able to make the desired changes in our program and call data members in the main function as well as in the derived class.