University of Science & Technology – Zewail City

Nanotechnology and Nanoelectronics Engineering

C/C++ Programming Lab – NANENG 112/312

Instructor: Dr Yasser Elawady

Tutorial 8 – Object Oriented Programming

Prepared By:

Ibrahim Mostafa - John Wafeek

Agenda

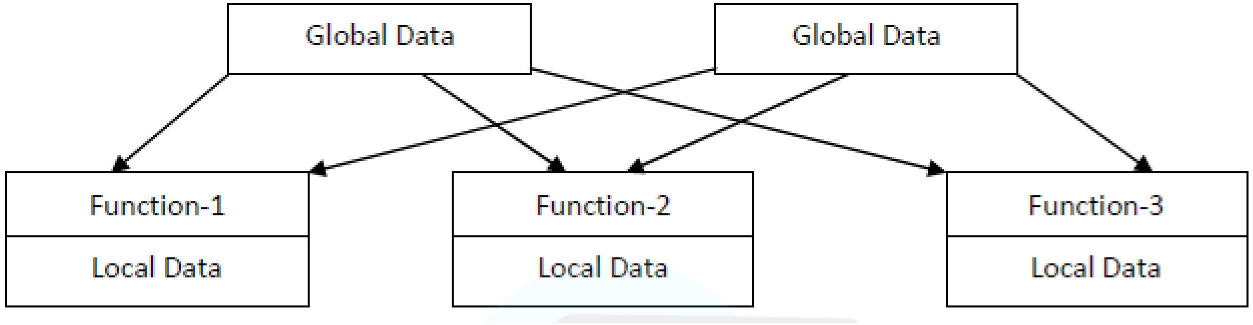
- ➢ Programming types
- ➢ Basic OOP Concepts
- ➢ OOP Classes & Objects
- ➢ Const and Static class members
- ➢ Object as Function Argument
- ➢ Constructors & Destructors

Programming types

Procedural Programming

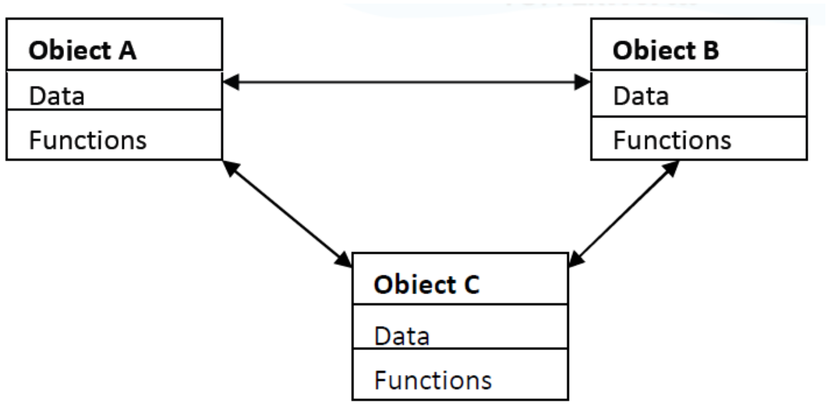
- Conventional programming where a problem is viewed as a sequence of steps/procedures.

  Ex: reading data, calculating, printing result.
- Number of functions are written to accomplish these steps.



Object Oriented Programming (OOP)

- Programs are divided into what is called (objects).
- Data can be hidden and not directly accessed by external functions (i.e. it provides controlled access of the data to protect data from corruption).
- Objects may communicate with each other through functions.



Basic OOP Concepts

Basic OOP Concepts:

- Data Abstraction: Representing the essential features without including all background details.

  Classes use the concept of data abstraction and are called Abstract Data Types (ADT)

- Encapsulation(data hiding): making the variables in a class private and accessible only by a controlled interface of public or protected functions.

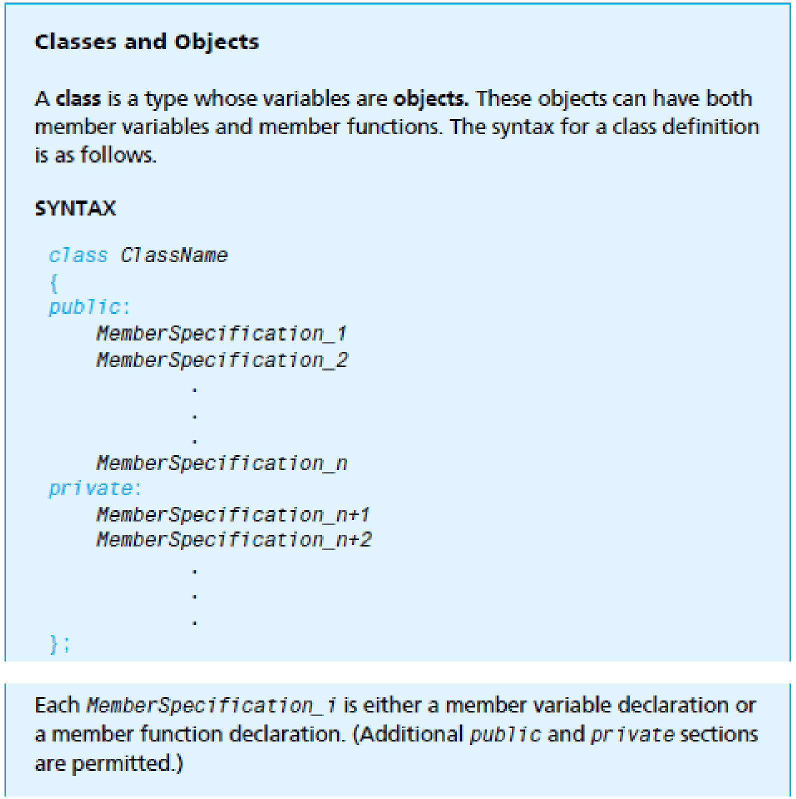
- Inheritance: process of acquiring properties of other objects (i.e. member variables/functions) and add additional new features without modifying the existing one. It is useful for code reusability.

- Polymorphism: Ability of an object to perform in more than one form.

Classes & Objects

Class Definition:

- Class: a data type that contains member variables and functions used to perform operations on their variables (objects).
- Object: an instance/variable of the class.
- Class is declared globally outside of main function.
- Objects are created within the main function by specifying the class name followed by object name.



Classes and Objects

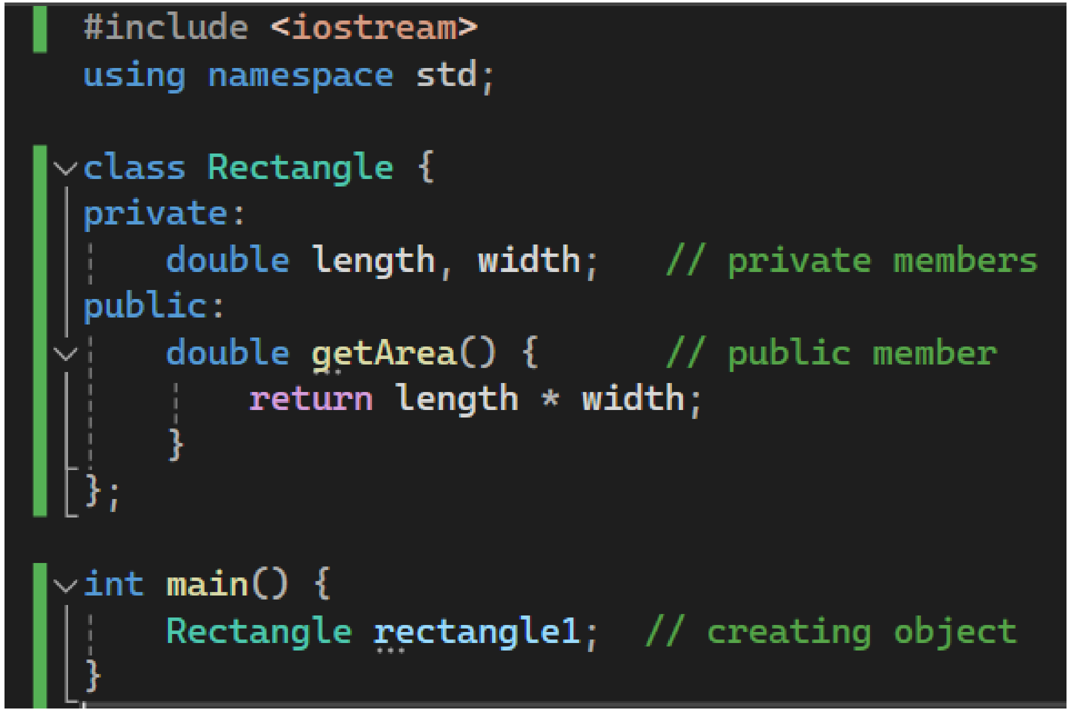
A **class** is a type whose variables are **objects.** These objects can have both member variables and member functions. The syntax for a class definition is as follows.

SYNTAX

```
class ClassName
{
public:
    MemberSpecification_1
    MemberSpecification_2
            .
            .
            .
    MemberSpecification_n
private:
    MemberSpecification_n+1
    MemberSpecification_n+2
            .
            .
            .
};
```

Each *MemberSpecification_i* is either a member variable declaration or a member function declaration. (Additional `public` and `private` sections are permitted.)

Ex: Class Definition



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

class Rectangle {
private:
    double length, width;    // private members
public:
    double getArea() {       // public member
        return length * width;
    }
};

int main() {
    Rectangle rectangle1;  // creating object
}
```

Access Specifiers/Modifiers:

- Private:

  Private members can be accessed only from within the class.

- Public:

  Public members can be accessed from outside the class.

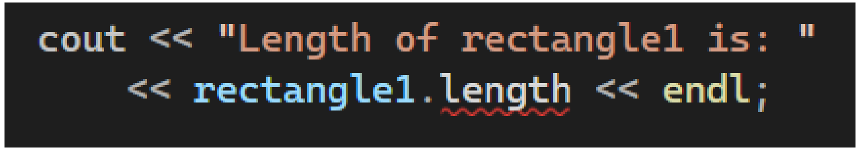
- Protected:

  Protected members can be accessed from within the class and its derived classes.

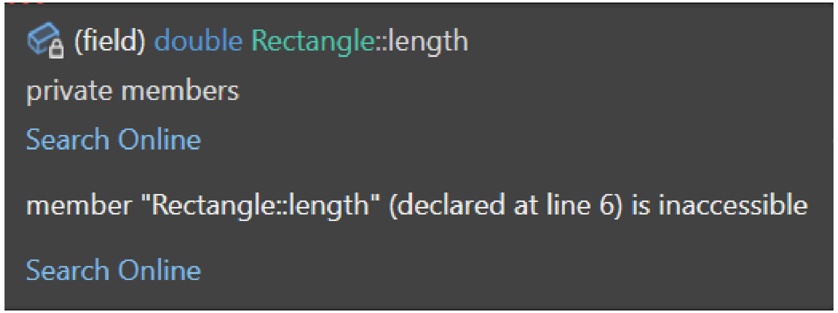
- By default, if not stated, Class members are private, Unlike struct.

- Private data members can be accessed by functions that are wrapped inside the class.

Ex: Access Specifiers



```
cout << "Length of rectangle1 is: "
    << rectangle1.length << endl;
```



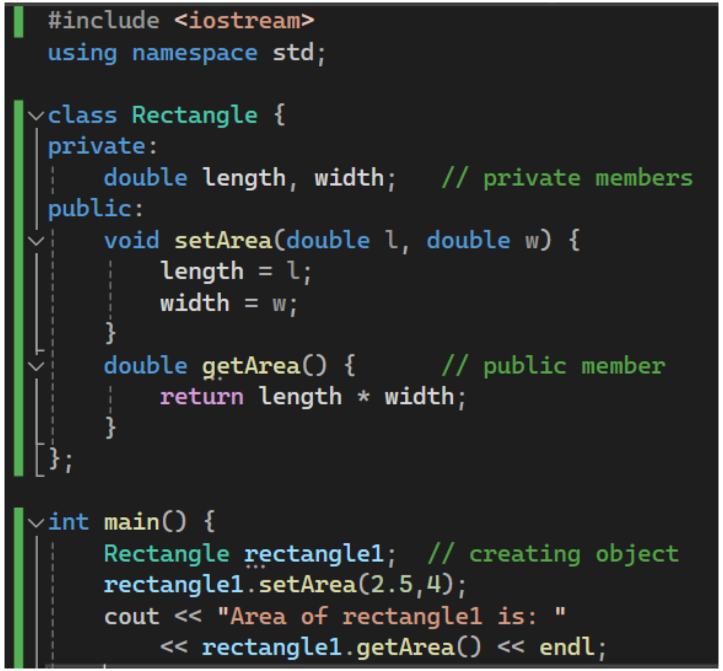
(field) double Rectangle::length

private members

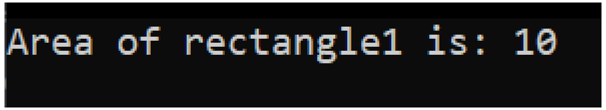
Search Online

member "Rectangle::length" (declared at line 6) is inaccessible

Search Online



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

class Rectangle {
private:
    double length, width;    // private members
public:
    void setArea(double l, double w) {
        length = l;
        width = w;
    }
    double getArea() {        // public member
        return length * width;
    }
};

int main() {
    Rectangle rectangle1;  // creating object
    rectangle1.setArea(2.5,4);
    cout << "Area of rectangle1 is: "
        << rectangle1.getArea() << endl;
```

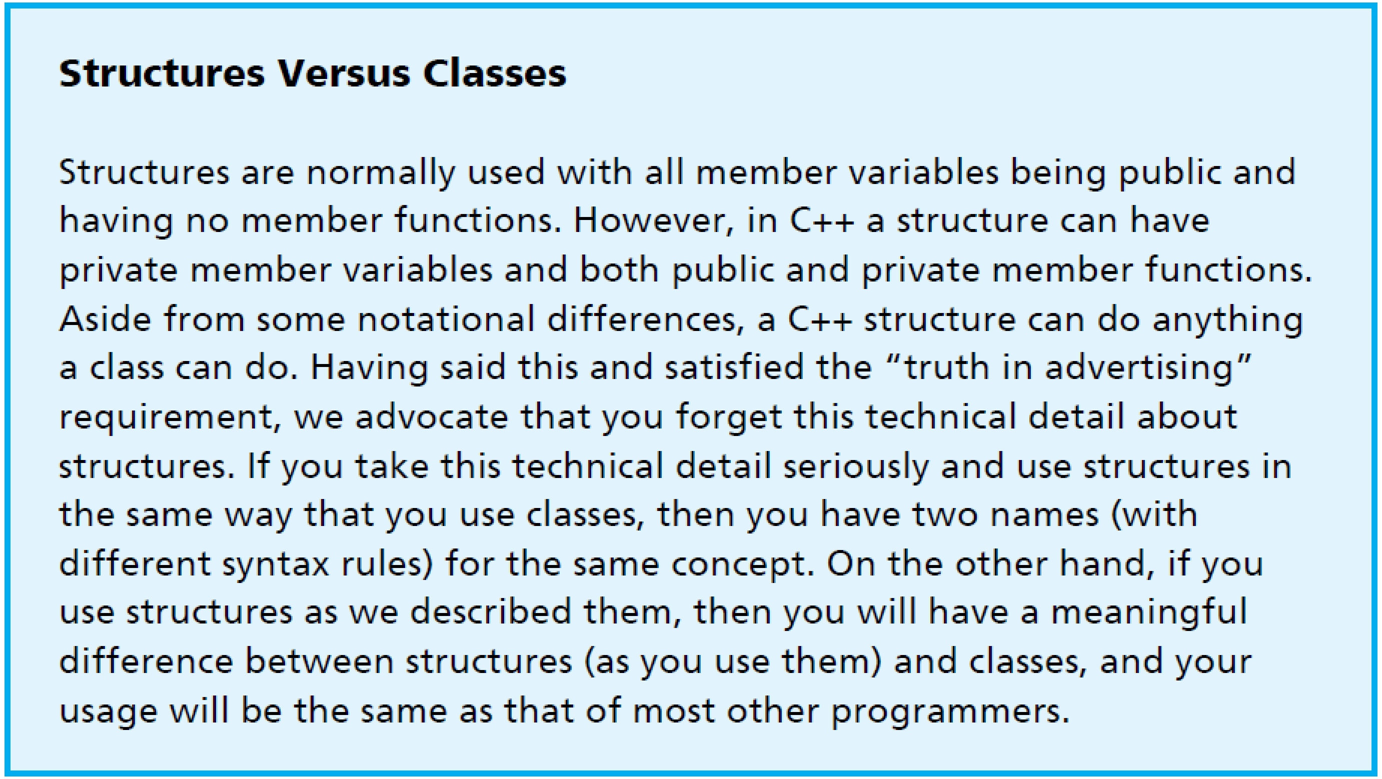


```
Area of rectangle1 is: 10
```

Difference between Class and Struct

Class	Struct
Members of class are private by default	Members of struct are public by default
Normally used for data abstraction and inheritance	Normally used for grouping relevant data types
Upon creation, memory is allocated on the heap	Upon creation, memory is allocated on the stack
It is a reference type	It is a value type
It can have null values	It can not have null values

Difference between Class and Struct

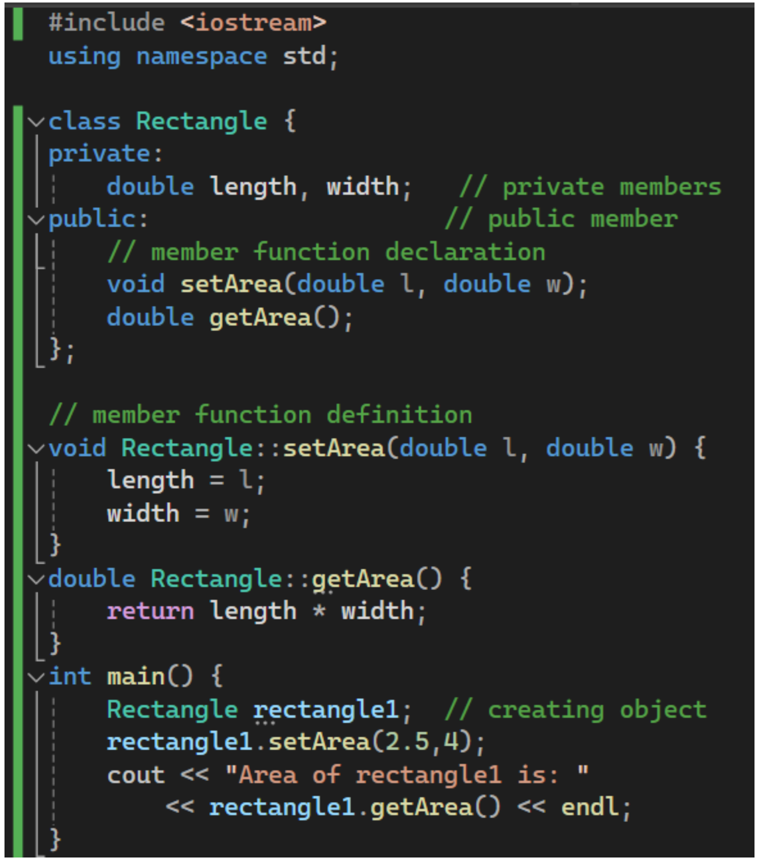


Structures Versus Classes

Structures are normally used with all member variables being public and having no member functions. However, in C++ a structure can have private member variables and both public and private member functions. Aside from some notational differences, a C++ structure can do anything a class can do. Having said this and satisfied the “truth in advertising” requirement, we advocate that you forget this technical detail about structures. If you take this technical detail seriously and use structures in the same way that you use classes, then you have two names (with different syntax rules) for the same concept. On the other hand, if you use structures as we described them, then you will have a meaningful difference between structures (as you use them) and classes, and your usage will be the same as that of most other programmers.

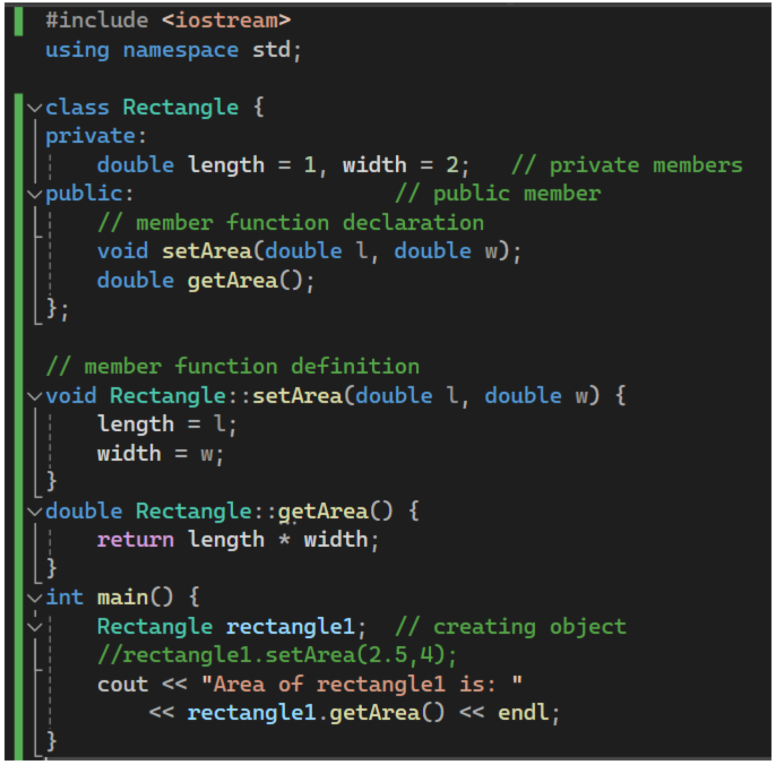
Member Variable/Function Definition

- You can define member variable/function of a class inside it normally, or outside of the class using scope resolution operator (::)

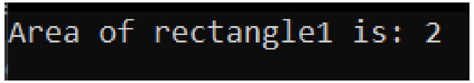


```cpp
#include <iostream>
using namespace std;

class Rectangle {
private:
    double length, width;    // private members
public:                      // public member
    // member function declaration
    void setArea(double l, double w);
    double getArea();
};

// member function definition
void Rectangle::setArea(double l, double w) {
    length = l;
    width = w;
}
double Rectangle::getArea() {
    return length * width;
}
int main() {
    Rectangle rectangle1;  // creating object
    rectangle1.setArea(2.5,4);
    cout << "Area of rectangle1 is: "
        << rectangle1.getArea() << endl;
}
```

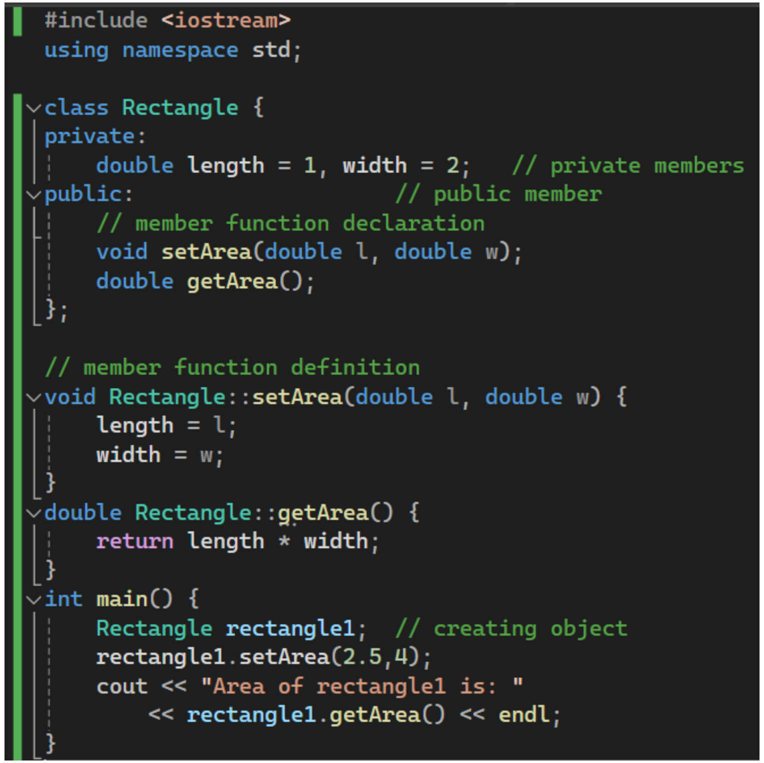
Class default values



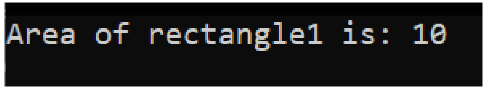
```cpp
#include <iostream>
using namespace std;

class Rectangle {
private:
    double length = 1, width = 2;    // private members
public:                        // public member
    // member function declaration
    void setArea(double l, double w);
    double getArea();
};

// member function definition
void Rectangle::setArea(double l, double w) {
    length = l;
    width = w;
}
double Rectangle::getArea() {
    return length * width;
}
int main() {
    Rectangle rectangle1;  // creating object
    //rectangle1.setArea(2.5,4);
    cout << "Area of rectangle1 is: "
        << rectangle1.getArea() << endl;
}
```



```
Area of rectangle1 is: 2
```

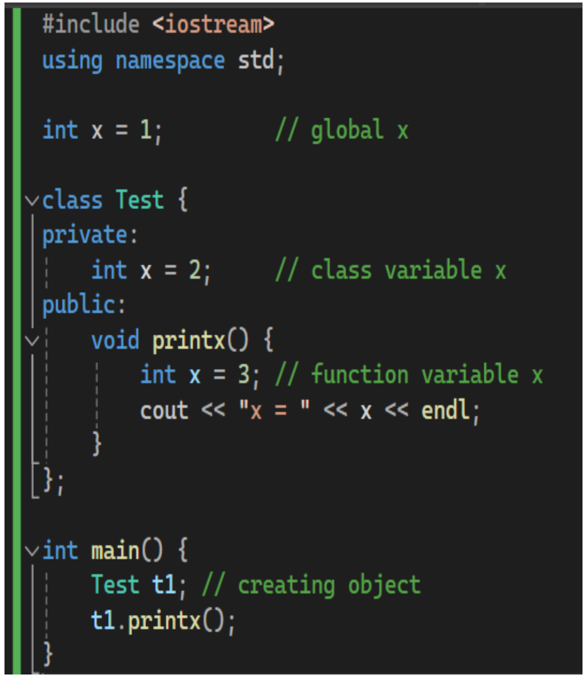


```cpp
#include <iostream>
using namespace std;

class Rectangle {
private:
    double length = 1, width = 2;    // private members
public:                        // public member
    // member function declaration
    void setArea(double l, double w);
    double getArea();
};

// member function definition
void Rectangle::setArea(double l, double w) {
    length = l;
    width = w;
}
double Rectangle::getArea() {
    return length * width;
}
int main() {
    Rectangle rectangle1;  // creating object
    rectangle1.setArea(2.5,4);
    cout << "Area of rectangle1 is: "
        << rectangle1.getArea() << endl;
}
```

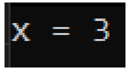


```
Area of rectangle1 is: 10
```

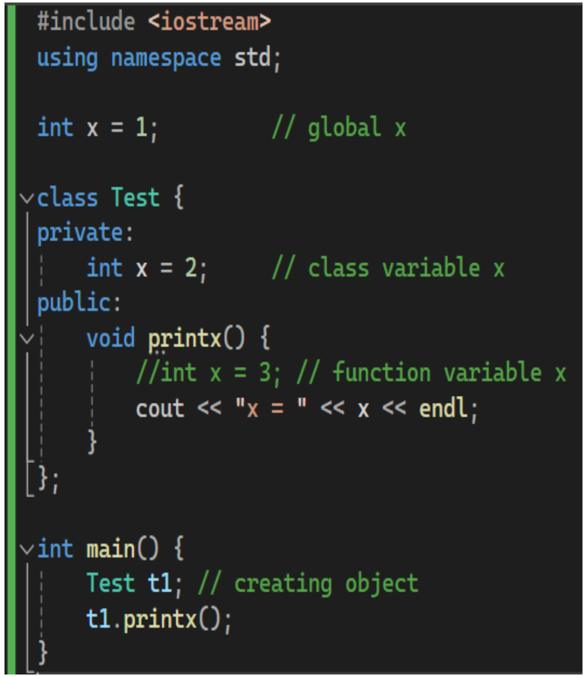
Scope of local variables



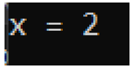
```
#include <iostream>
using namespace std;

int x = 1;          // global x

class Test {
private:
    int x = 2;      // class variable x
public:
    void printx() {
        int x = 3; // function variable x
        cout << "x = " << x << endl;
    }
};

int main() {
    Test t1; // creating object
    t1.printx();
}
```



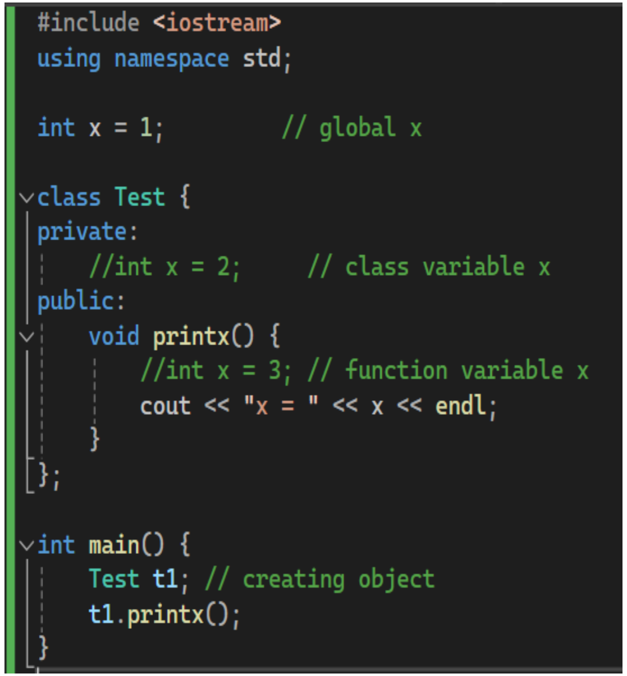
```
x = 3
```



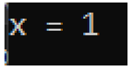
```
#include <iostream>
using namespace std;

int x = 1;          // global x

class Test {
private:
    int x = 2;      // class variable x
public:
    void printx() {
        //int x = 3; // function variable x
        cout << "x = " << x << endl;
    }
};

int main() {
    Test t1; // creating object
    t1.printx();
}
```



```
x = 2
```

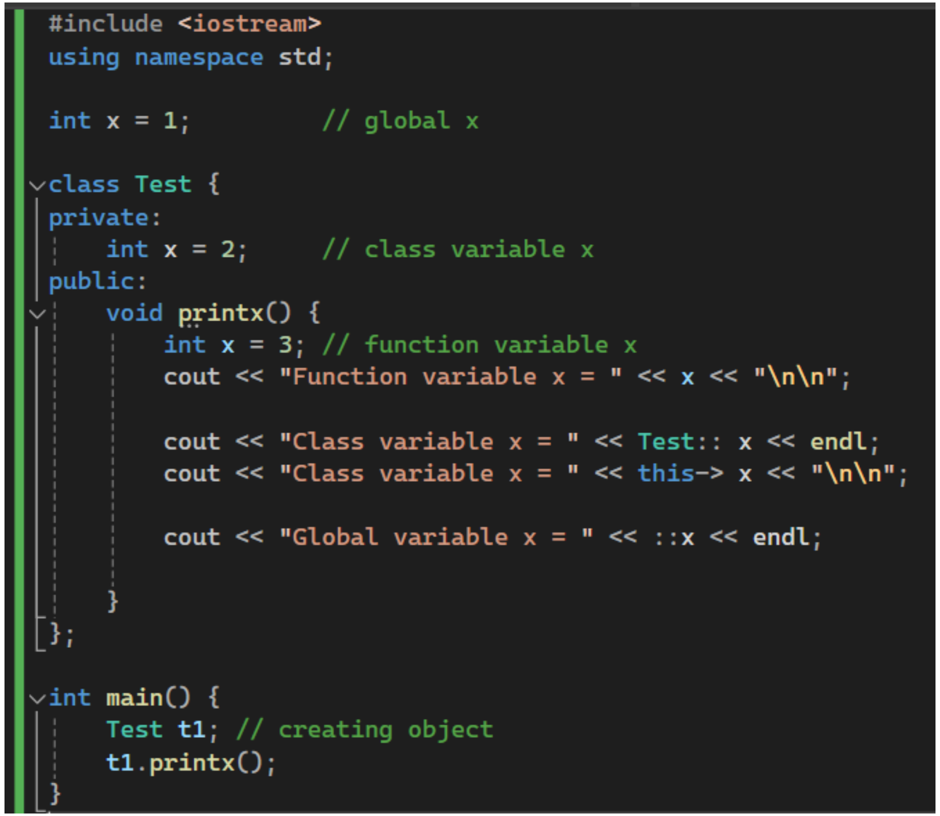


```
#include <iostream>
using namespace std;

int x = 1;          // global x

class Test {
private:
    //int x = 2;      // class variable x
public:
    void printx() {
        //int x = 3; // function variable x
        cout << "x = " << x << endl;
    }
};

int main() {
    Test t1; // creating object
    t1.printx();
}
```

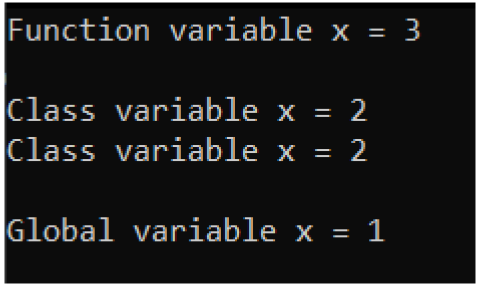


```
x = 1
```

Scope of local variables



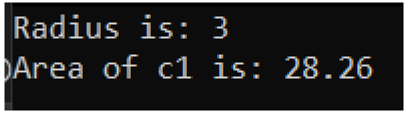
```
#include <iostream>
using namespace std;

int x = 1;          // global x

class Test {
private:
    int x = 2;      // class variable x
public:
    void printx() {
        int x = 3; // function variable x
        cout << "Function variable x = " << x << "\n\n";

        cout << "Class variable x = " << Test:: x << endl;
        cout << "Class variable x = " << this-> x << "\n\n";

        cout << "Global variable x = " << ::x << endl;

    }
};

int main() {
    Test t1; // creating object
    t1.printx();
}
```



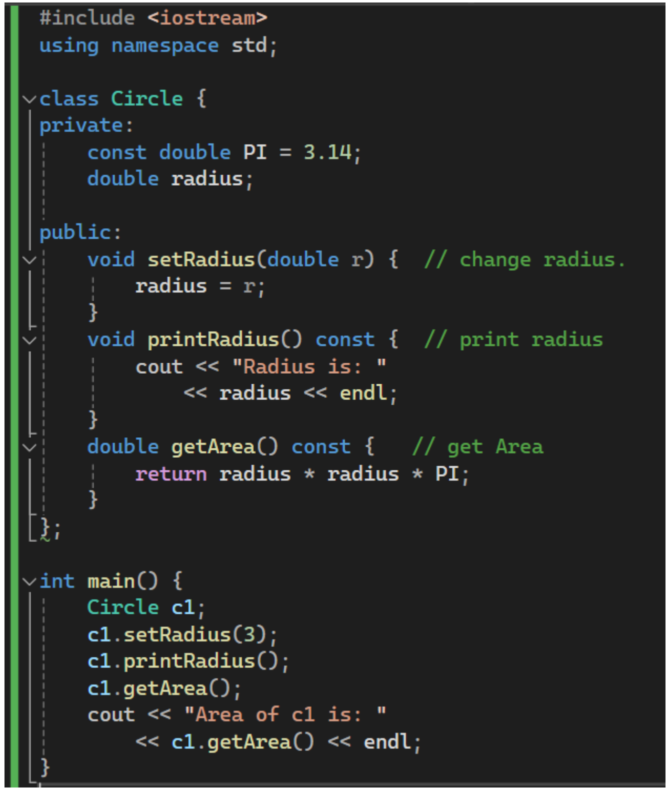
```
Function variable x = 3

Class variable x = 2
Class variable x = 2

Global variable x = 1
```

Const members

- Const member variables keep their value unchanged.
- Const member functions do not change (protect) the value of member variables.



```
Radius is: 3
Area of c1 is: 28.26
```



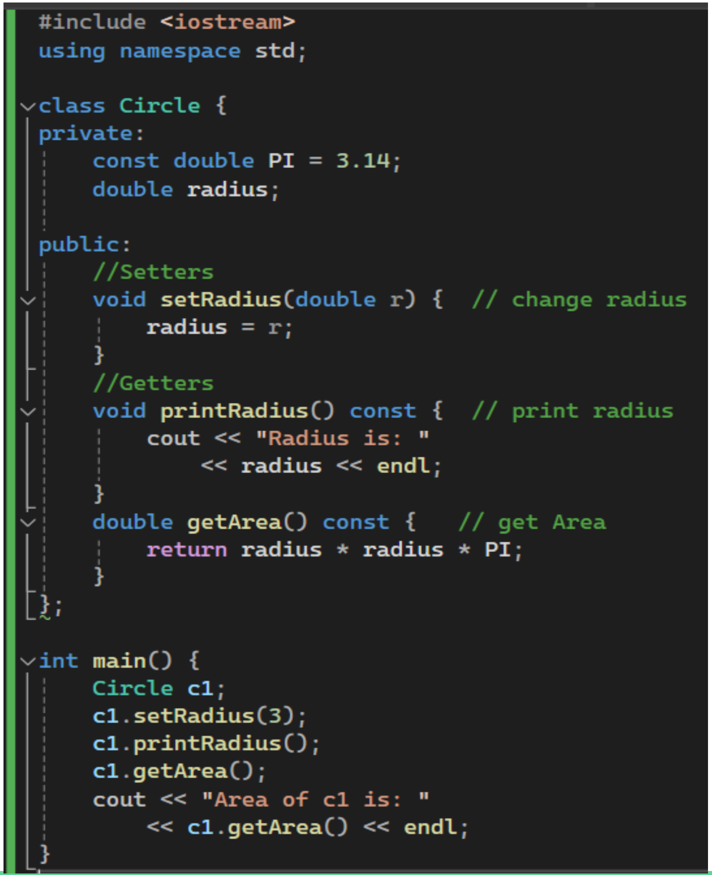
```cpp
#include <iostream>
using namespace std;

class Circle {
private:
    const double PI = 3.14;
    double radius;

public:
    void setRadius(double r) {  // change radius.
        radius = r;
    }
    void printRadius() const {  // print radius
        cout << "Radius is: "
            << radius << endl;
    }
    double getArea() const {   // get Area
        return radius * radius * PI;
    }
};

int main() {
    Circle c1;
    c1.setRadius(3);
    c1.printRadius();
    c1.getArea();
    cout << "Area of c1 is: "
        << c1.getArea() << endl;
}
```

Setters & Getters

- Setters provide controlled access for class private member variables to set (change) their value.

- Getters provide controlled access for class private member variables to read/return their value.

- Getters do not change class data so, they can be Const.



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

class Circle {
private:
    const double PI = 3.14;
    double radius;

public:
    //Setters
    void setRadius(double r) {  // change radius
        radius = r;
    }
    //Getters
    void printRadius() const {  // print radius
        cout << "Radius is: "
            << radius << endl;
    }
    double getArea() const {   // get Area
        return radius * radius * PI;
    }
};

int main() {
    Circle c1;
    c1.setRadius(3);
    c1.printRadius();
    c1.getArea();
    cout << "Area of c1 is: "
        << c1.getArea() << endl;
}
```

Scope Resolution and Dot Operator:

- Dot operator (.) is used to:

  - access public members of a class (member variables/functions).

- Scope resolution operator (::) is used to:

  - access global variables

  - define member functions outside the class.

  - initialize static member variables outside the class.

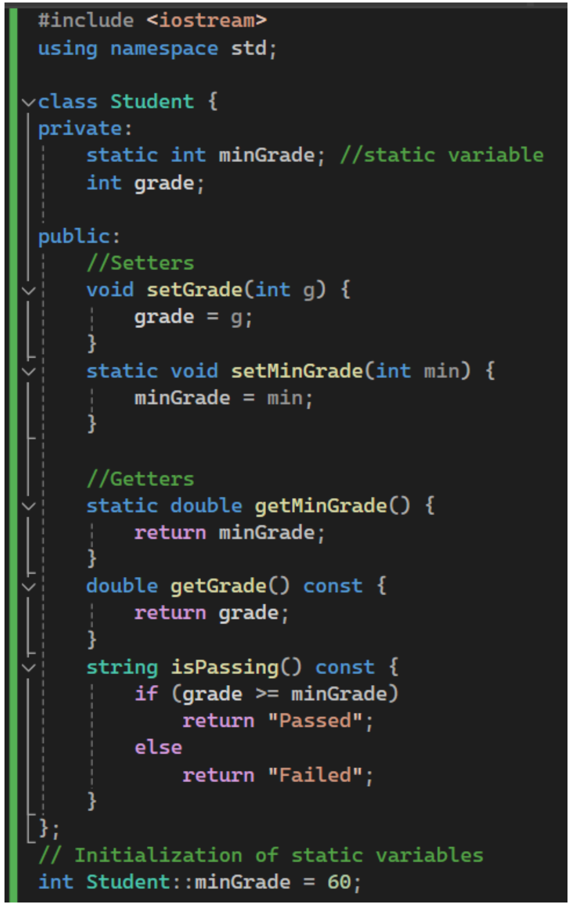
  - define static member functions outside the class.

  - access static member functions of a class.

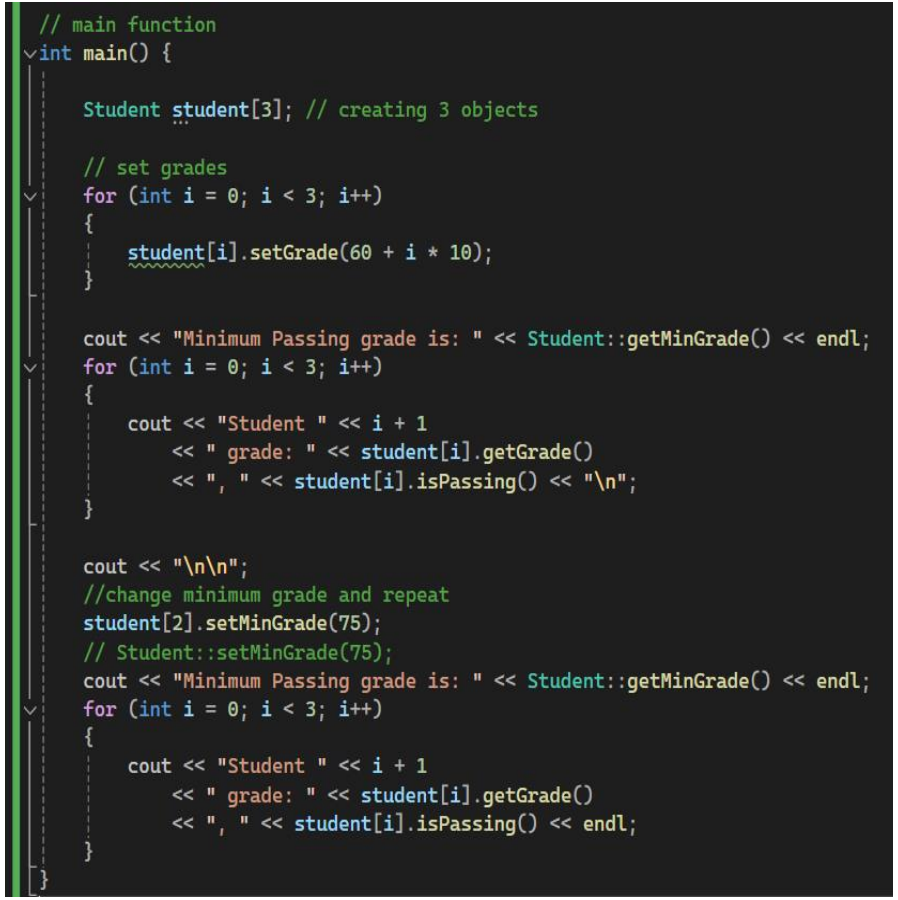
Static Member Variables/Functions

- When we declare a static member variable, We are telling the compiler that only one copy of this variable will exist for all objects. (i.e. static member variable is shared among all objects).
- Static members are called class variables, while non-static (instance) members are called object variables.
- Static member functions are the ones that can operate only on static member variables.
- Static member variables are declared inside the class but, Initialized outside the class using scope resolution operator (::)

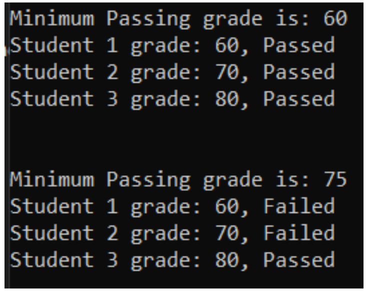
Ex: static data members/Functions



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

class Student {
private:
    static int minGrade; //static variable
    int grade;

public:
    //Setters
    void setGrade(int g) {
        grade = g;
    }
    static void setMinGrade(int min) {
        minGrade = min;
    }

    //Getters
    static double getMinGrade() {
        return minGrade;
    }
    double getGrade() const {
        return grade;
    }
    string isPassing() const {
        if (grade >= minGrade)
            return "Passed";
        else
            return "Failed";
    }
};
// Initialization of static variables
int Student::minGrade = 60;
```



```cpp
// main function
int main() {

    Student student[3]; // creating 3 objects

    // set grades
    for (int i = 0; i < 3; i++)
    {
        student[i].setGrade(60 + i * 10);
    }

    cout << "Minimum Passing grade is: " << Student::getMinGrade() << endl;
    for (int i = 0; i < 3; i++)
    {
        cout << "Student " << i + 1
            << " grade: " << student[i].getGrade()
            << ", " << student[i].isPassing() << "\n";
    }

    cout << "\n\n";
    //change minimum grade and repeat
    student[2].setMinGrade(75);
    // Student::setMinGrade(75);
    cout << "Minimum Passing grade is: " << Student::getMinGrade() << endl;
    for (int i = 0; i < 3; i++)
    {
        cout << "Student " << i + 1
            << " grade: " << student[i].getGrade()
            << ", " << student[i].isPassing() << endl;
    }
}
```



```
Minimum Passing grade is: 60
Student 1 grade: 60, Passed
Student 2 grade: 70, Passed
Student 3 grade: 80, Passed


Minimum Passing grade is: 75
Student 1 grade: 60, Failed
Student 2 grade: 70, Failed
Student 3 grade: 80, Passed
```

Object as Function Argument

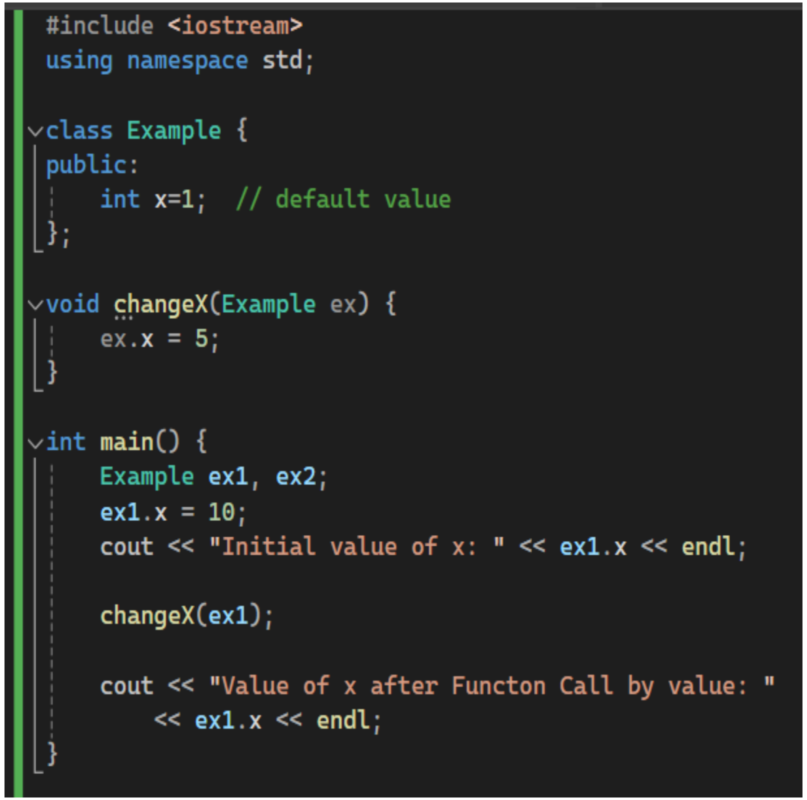
Object as Function Argument

- You can pass an object as a function argument by value and by reference but,
  It is more convenient to pass it by reference.

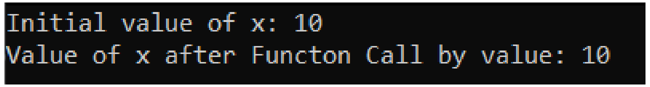
- Pass by value: creates a local (shallow) copy of the object in the function scope.

- Pass by reference: passes the reference of the object to the function, anything you modify will be reflected on the object.

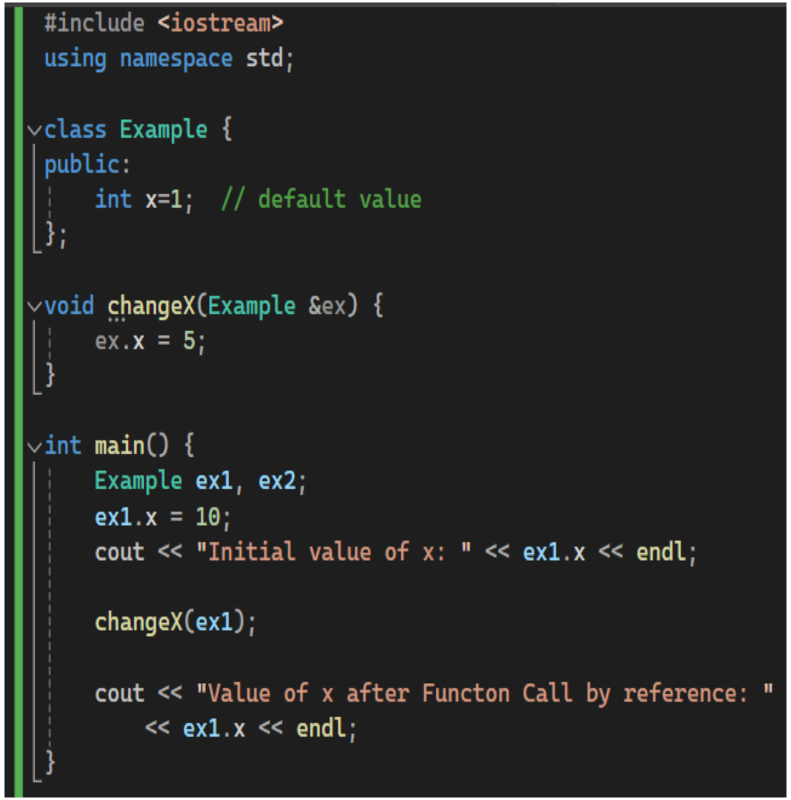
Ex: Object as Function Argument.



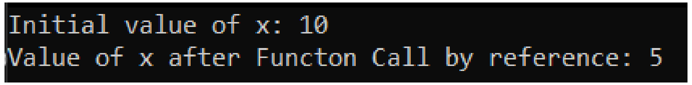
```cpp
#include <iostream>
using namespace std;

class Example {
public:
    int x=1;  // default value
};

void changeX(Example ex) {
    ex.x = 5;
}

int main() {
    Example ex1, ex2;
    ex1.x = 10;
    cout << "Initial value of x: " << ex1.x << endl;

    changeX(ex1);

    cout << "Value of x after Functon Call by value: "
        << ex1.x << endl;
}
```



```
Initial value of x: 10
Value of x after Functon Call by value: 10
```



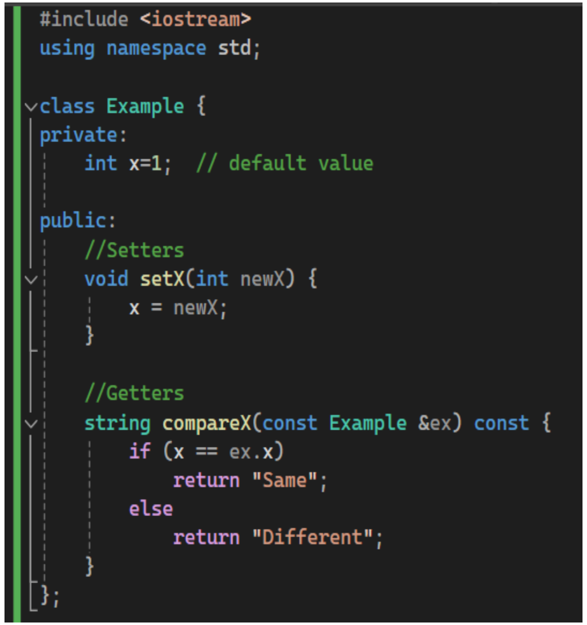
```cpp
#include <iostream>
using namespace std;

class Example {
public:
    int x=1;  // default value
};

void changeX(Example &ex) {
    ex.x = 5;
}

int main() {
    Example ex1, ex2;
    ex1.x = 10;
    cout << "Initial value of x: " << ex1.x << endl;

    changeX(ex1);

    cout << "Value of x after Functon Call by reference: "
        << ex1.x << endl;
}
```



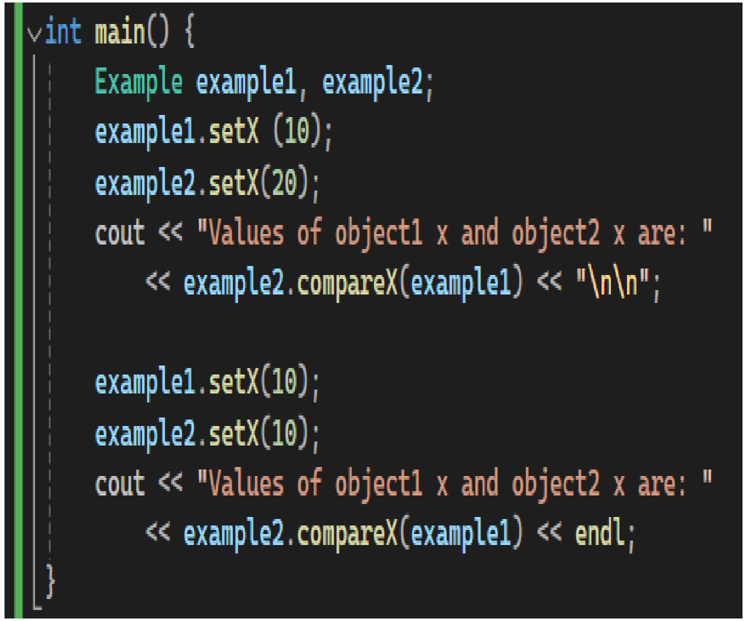
```
Initial value of x: 10
Value of x after Functon Call by reference: 5
```

Object as Function Argument

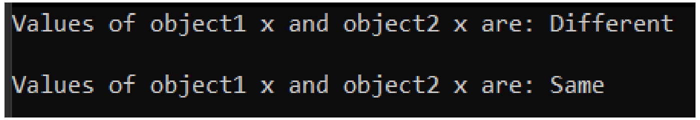
- An object can access private members of another object if it is passed as a member function argument.



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

class Example {
private:
    int x=1;  // default value

public:
    //Setters
    void setX(int newX) {
        x = newX;
    }

    //Getters
    string compareX(const Example &ex) const {
        if (x == ex.x)
            return "Same";
        else
            return "Different";
    }
};
```



```cpp
int main() {
    Example example1, example2;
    example1.setX (10);
    example2.setX(20);
    cout << "Values of object1 x and object2 x are: "
        << example2.compareX(example1) << "\n\n";

    example1.setX(10);
    example2.setX(10);
    cout << "Values of object1 x and object2 x are: "
        << example2.compareX(example1) << endl;
}
```

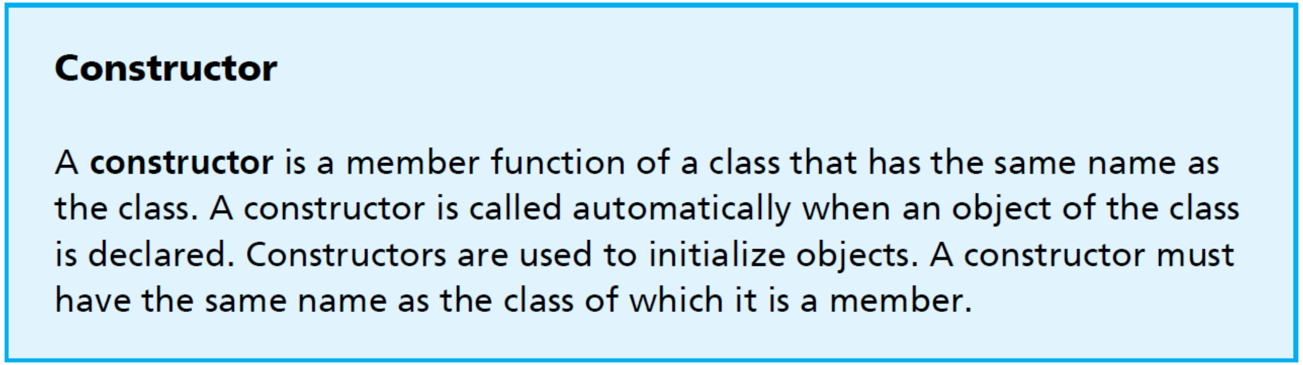


```
Values of object1 x and object2 x are: Different

Values of object1 x and object2 x are: Same
```

Constructors & Destructor

Constructors

- Constructor: is a special member function of a class whose task it to initialize objects of the class.
- It has the same name of the class, and does not have a return type.
- The constructor is automatically executed each time an object of the class is created so, it can keep track to number of objects created of a specific class.
- A constructor can be overloaded.



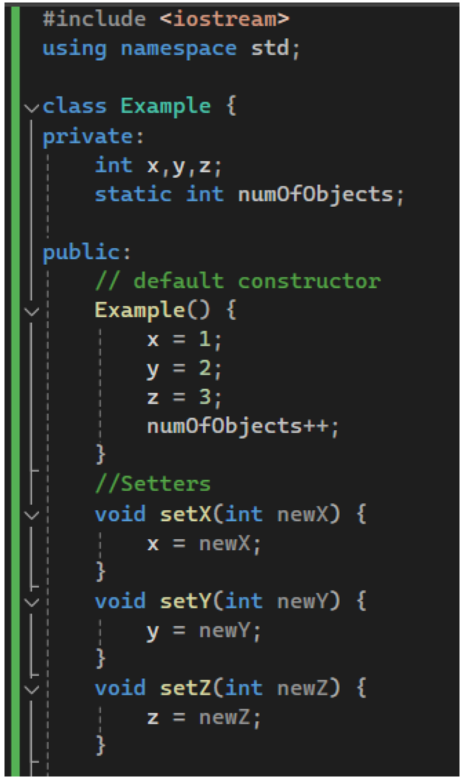
Constructor

A **constructor** is a member function of a class that has the same name as the class. A constructor is called automatically when an object of the class is declared. Constructors are used to initialize objects. A constructor must have the same name as the class of which it is a member.

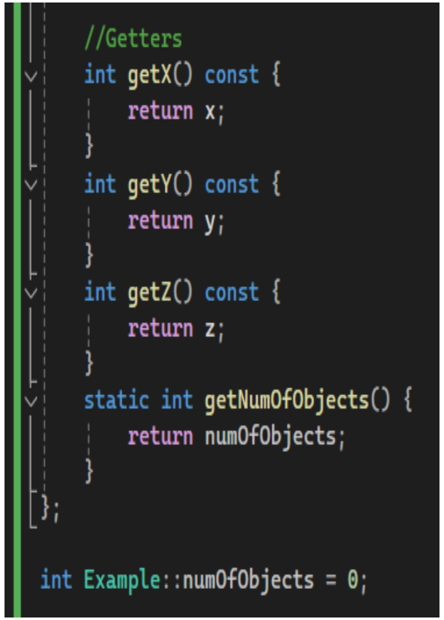
Default vs Parametrized Constructor

- Even if you declare a class with no constructor, each time an object of a class is created, the compiler calls/creates a default (empty) constructor.
- Default constructor: A constructor that has no arguments.
- Parametrized constructor: A constructor that has arguments.
- Constructors can access static and non-static (instance) class members.

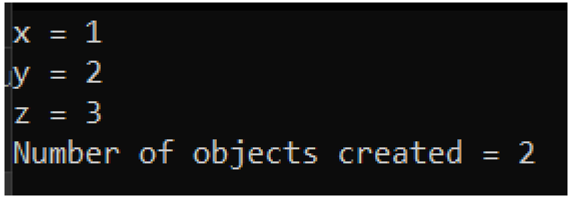
Ex: Default Constructor



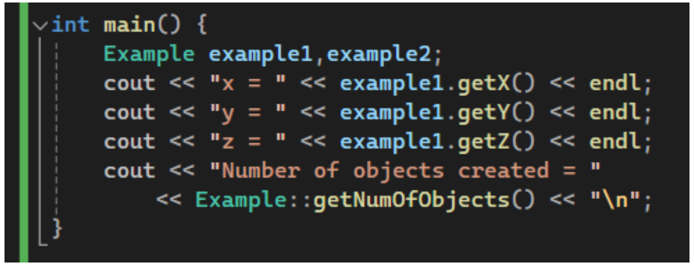
```cpp
#include <iostream>
using namespace std;

class Example {
private:
    int x,y,z;
    static int numOfObjects;

public:
    // default constructor
    Example() {
        x = 1;
        y = 2;
        z = 3;
        numOfObjects++;
    }
    //Setters
    void setX(int newX) {
        x = newX;
    }
    void setY(int newY) {
        y = newY;
    }
    void setZ(int newZ) {
        z = newZ;
    }
```



```cpp
    //Getters
    int getX() const {
        return x;
    }
    int getY() const {
        return y;
    }
    int getZ() const {
        return z;
    }
    static int getNumOfObjects() {
        return numOfObjects;
    }
};

int Example::numOfObjects = 0;
```

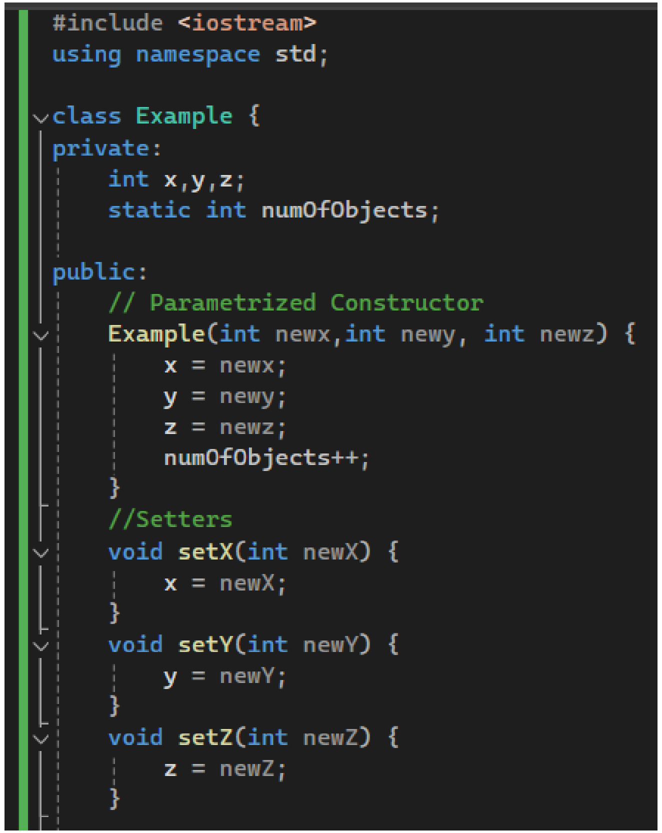


```
x = 1
y = 2
z = 3
Number of objects created = 2
```

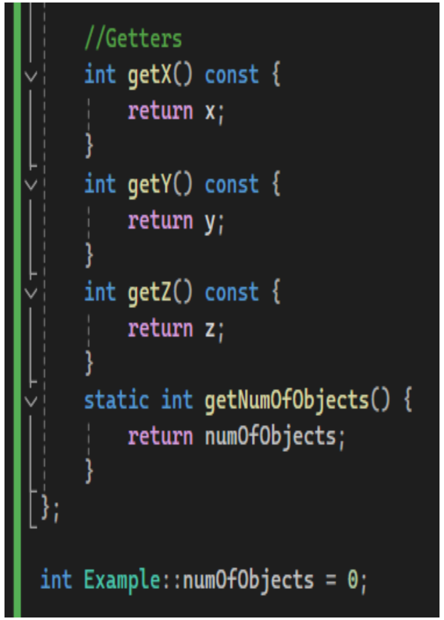


```cpp
int main() {
    Example example1,example2;
    cout << "x = " << example1.getX() << endl;
    cout << "y = " << example1.getY() << endl;
    cout << "z = " << example1.getZ() << endl;
    cout << "Number of objects created = "
        << Example::getNumOfObjects() << "\n";
}
```

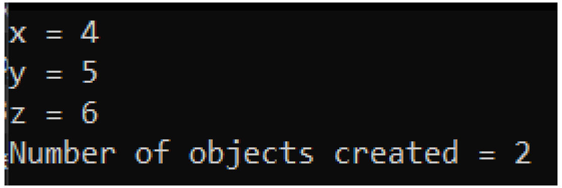
Parametrized Constructor



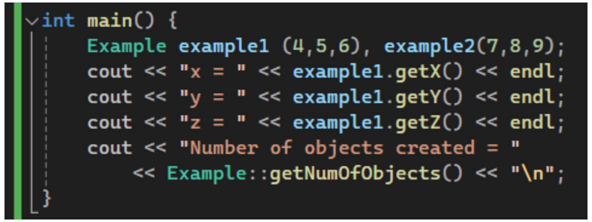
```cpp
#include <iostream>
using namespace std;

class Example {
private:
    int x,y,z;
    static int numOfObjects;

public:
    // Parametrized Constructor
    Example(int newx,int newy, int newz) {
        x = newx;
        y = newy;
        z = newz;
        numOfObjects++;
    }
    //Setters
    void setX(int newX) {
        x = newX;
    }
    void setY(int newY) {
        y = newY;
    }
    void setZ(int newZ) {
        z = newZ;
    }
```



```cpp
    //Getters
    int getX() const {
        return x;
    }
    int getY() const {
        return y;
    }
    int getZ() const {
        return z;
    }
    static int getNumOfObjects() {
        return numOfObjects;
    }
};

int Example::numOfObjects = 0;
```

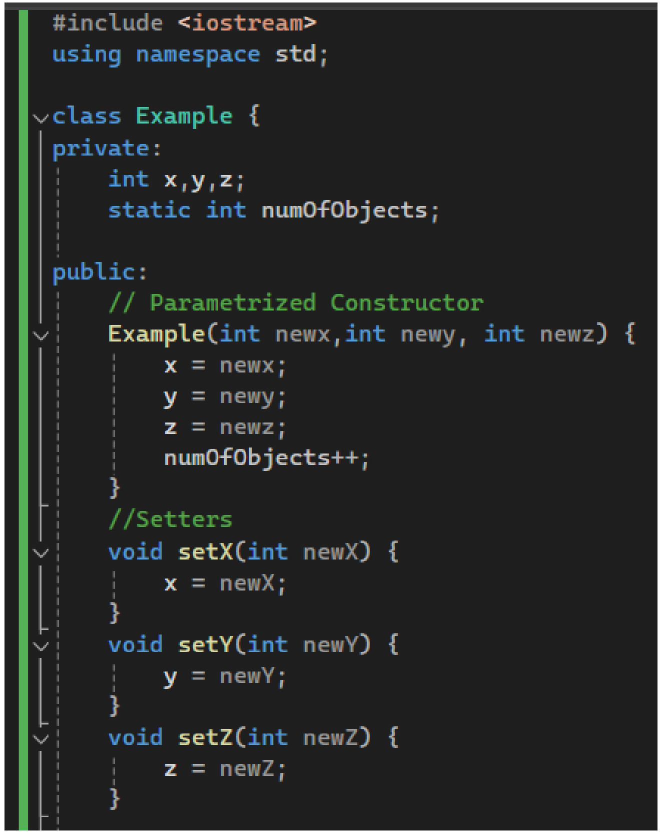


```
x = 4
y = 5
z = 6
Number of objects created = 2
```

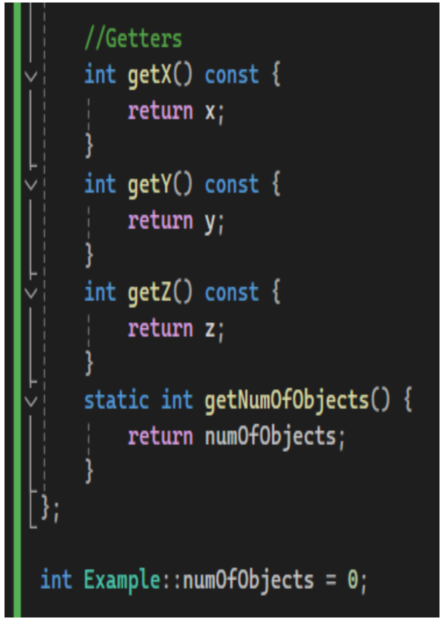


```cpp
int main() {
    Example example1 (4,5,6), example2(7,8,9);
    cout << "x = " << example1.getX() << endl;
    cout << "y = " << example1.getY() << endl;
    cout << "z = " << example1.getZ() << endl;
    cout << "Number of objects created = "
        << Example::getNumOfObjects() << "\n";
}
```

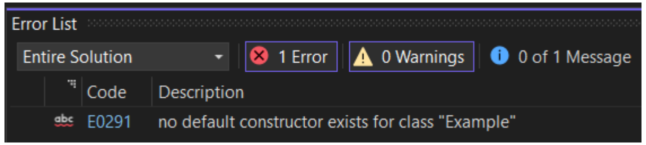
Parametrized Constructor



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

class Example {
private:
    int x,y,z;
    static int numOfObjects;

public:
    // Parametrized Constructor
    Example(int newx,int newy, int newz) {
        x = newx;
        y = newy;
        z = newz;
        numOfObjects++;
    }
    //Setters
    void setX(int newX) {
        x = newX;
    }
    void setY(int newY) {
        y = newY;
    }
    void setZ(int newZ) {
        z = newZ;
    }
```



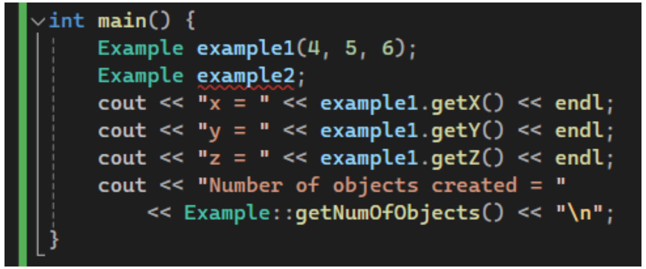
```cpp
    //Getters
    int getX() const {
        return x;
    }
    int getY() const {
        return y;
    }
    int getZ() const {
        return z;
    }
    static int getNumOfObjects() {
        return numOfObjects;
    }
};

int Example::numOfObjects = 0;
```



Error List

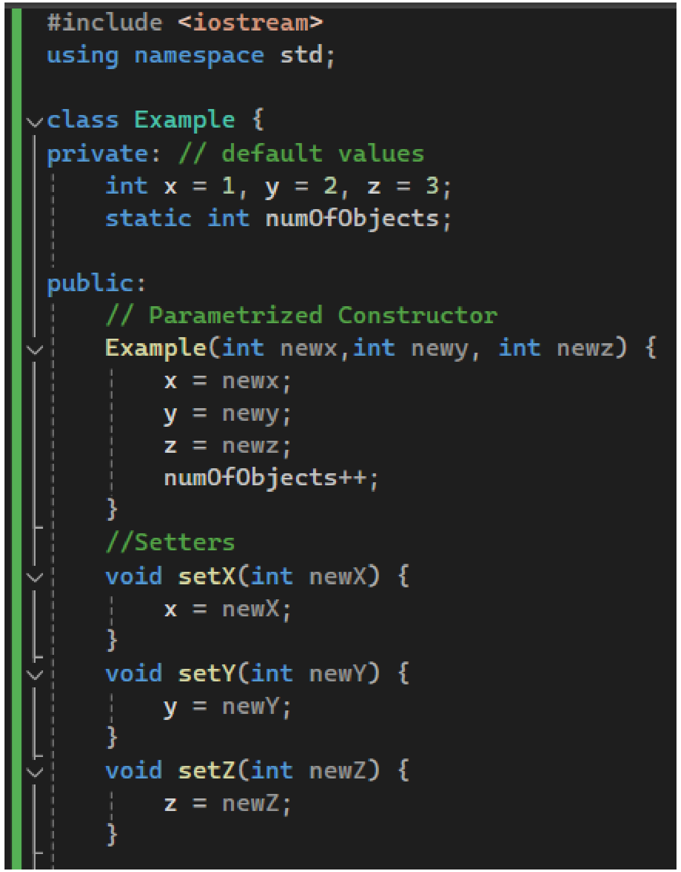
Entire Solution | 1 Error | 0 Warnings | 0 of 1 Message

Code	Description
E0291	no default constructor exists for class "Example"

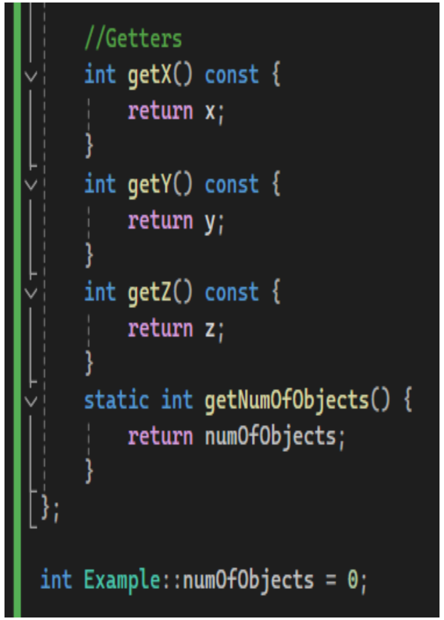


```cpp
int main() {
    Example example1(4, 5, 6);
    Example example2;
    cout << "x = " << example1.getX() << endl;
    cout << "y = " << example1.getY() << endl;
    cout << "z = " << example1.getZ() << endl;
    cout << "Number of objects created = "
        << Example::getNumOfObjects() << "\n";
}
```

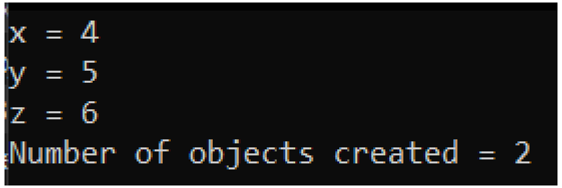
Constructor Initialization overrides default values.



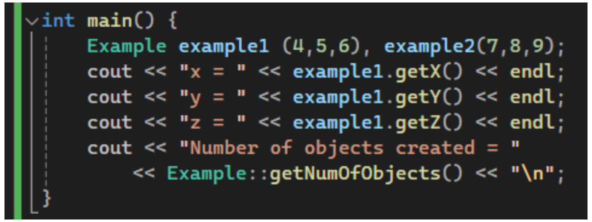
```cpp
#include <iostream>
using namespace std;

class Example {
private: // default values
    int x = 1, y = 2, z = 3;
    static int numOfObjects;

public:
    // Parametrized Constructor
    Example(int newx,int newy, int newz) {
        x = newx;
        y = newy;
        z = newz;
        numOfObjects++;
    }
    //Setters
    void setX(int newX) {
        x = newX;
    }
    void setY(int newY) {
        y = newY;
    }
    void setZ(int newZ) {
        z = newZ;
    }
```



```cpp
    //Getters
    int getX() const {
        return x;
    }
    int getY() const {
        return y;
    }
    int getZ() const {
        return z;
    }
    static int getNumOfObjects() {
        return numOfObjects;
    }
};

int Example::numOfObjects = 0;
```

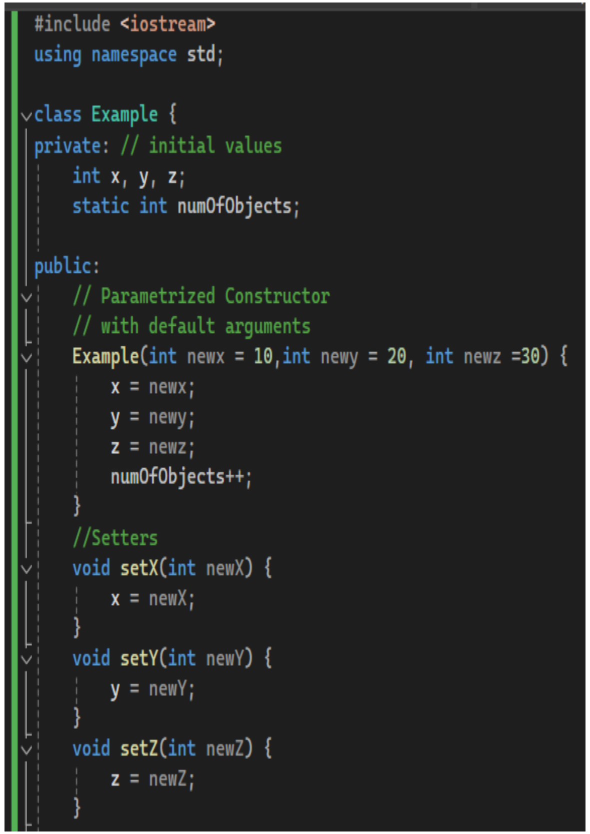


```
x = 4
y = 5
z = 6
Number of objects created = 2
```

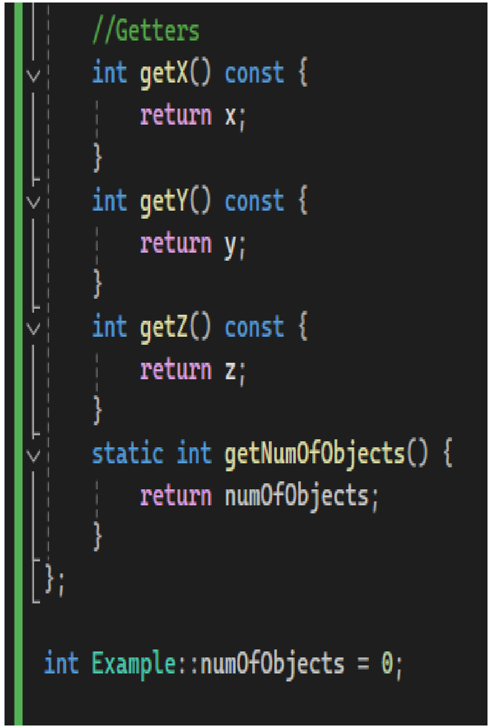


```cpp
int main() {
    Example example1 (4,5,6), example2(7,8,9);
    cout << "x = " << example1.getX() << endl;
    cout << "y = " << example1.getY() << endl;
    cout << "z = " << example1.getZ() << endl;
    cout << "Number of objects created = "
        << Example::getNumOfObjects() << "\n";
}
```

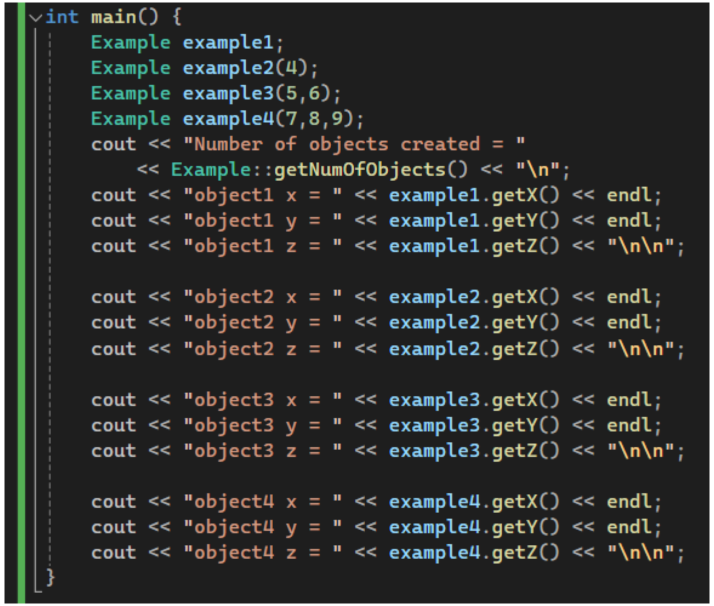
Constructor Overloading



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

class Example {
private: // initial values
    int x, y, z;
    static int numOfObjects;

public:
    // Parametrized Constructor
    // with default arguments
    Example(int newx = 10,int newy = 20, int newz =30) {
        x = newx;
        y = newy;
        z = newz;
        numOfObjects++;
    }
    //Setters
    void setX(int newX) {
        x = newX;
    }
    void setY(int newY) {
        y = newY;
    }
    void setZ(int newZ) {
        z = newZ;
    }
```

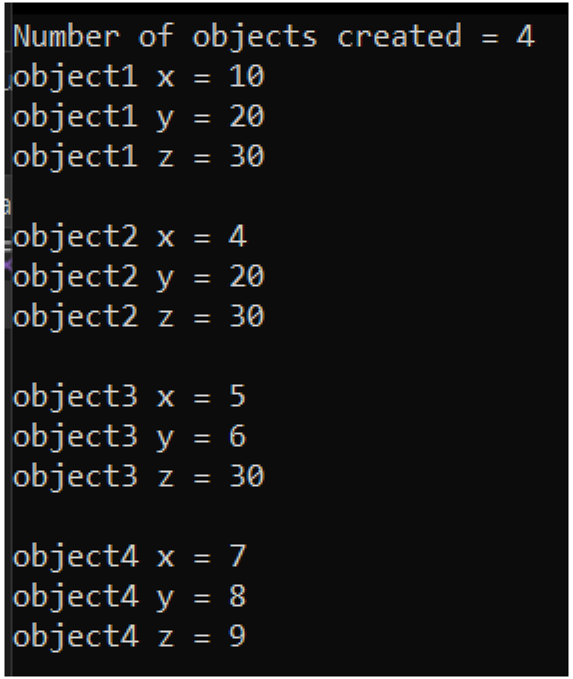


```cpp
    //Getters
    int getX() const {
        return x;
    }
    int getY() const {
        return y;
    }
    int getZ() const {
        return z;
    }
    static int getNumOfObjects() {
        return numOfObjects;
    }
};

int Example::numOfObjects = 0;
```



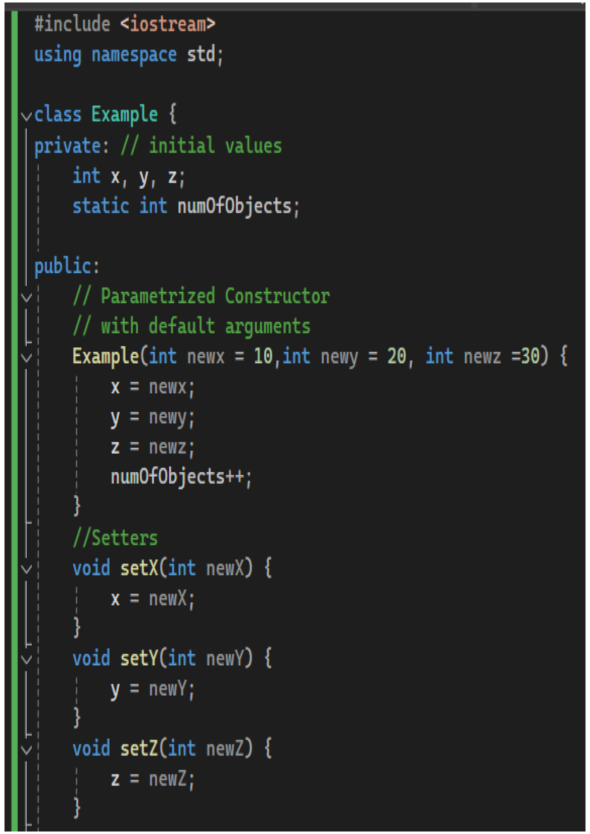
```cpp
int main() {
    Example example1;
    Example example2(4);
    Example example3(5,6);
    Example example4(7,8,9);
    cout << "Number of objects created = "
        << Example::getNumOfObjects() << "\n";
    cout << "object1 x = " << example1.getX() << endl;
    cout << "object1 y = " << example1.getY() << endl;
    cout << "object1 z = " << example1.getZ() << "\n\n";

    cout << "object2 x = " << example2.getX() << endl;
    cout << "object2 y = " << example2.getY() << endl;
    cout << "object2 z = " << example2.getZ() << "\n\n";

    cout << "object3 x = " << example3.getX() << endl;
    cout << "object3 y = " << example3.getY() << endl;
    cout << "object3 z = " << example3.getZ() << "\n\n";

    cout << "object4 x = " << example4.getX() << endl;
    cout << "object4 y = " << example4.getY() << endl;
    cout << "object4 z = " << example4.getZ() << "\n\n";
}
```

Constructor Overloading

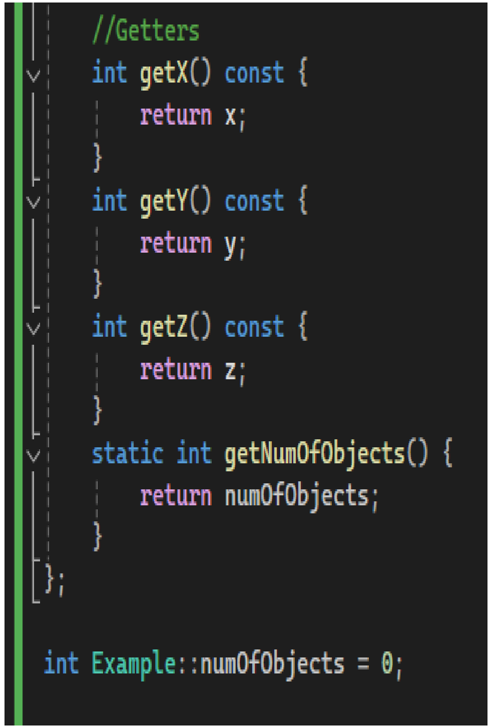


```
Number of objects created = 4
object1 x = 10
object1 y = 20
object1 z = 30

object2 x = 4
object2 y = 20
object2 z = 30

object3 x = 5
object3 y = 6
object3 z = 30

object4 x = 7
object4 y = 8
object4 z = 9
```

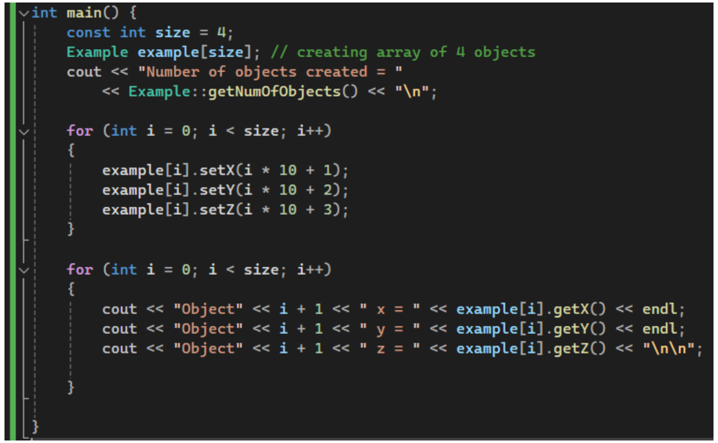
Array of objects



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

class Example {
private: // initial values
    int x, y, z;
    static int numOfObjects;

public:
    // Parametrized Constructor
    // with default arguments
    Example(int newx = 10,int newy = 20, int newz =30) {
        x = newx;
        y = newy;
        z = newz;
        numOfObjects++;
    }
    //Setters
    void setX(int newX) {
        x = newX;
    }
    void setY(int newY) {
        y = newY;
    }
    void setZ(int newZ) {
        z = newZ;
    }
```

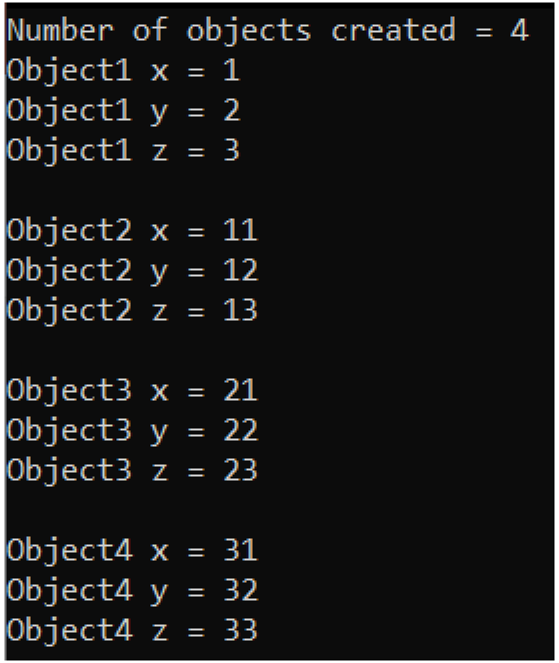


```cpp
    //Getters
    int getX() const {
        return x;
    }
    int getY() const {
        return y;
    }
    int getZ() const {
        return z;
    }
    static int getNumOfObjects() {
        return numOfObjects;
    }
};

int Example::numOfObjects = 0;
```



```cpp
int main() {
    const int size = 4;
    Example example[size]; // creating array of 4 objects
    cout << "Number of objects created = "
        << Example::getNumOfObjects() << "\n";

    for (int i = 0; i < size; i++)
    {
        example[i].setX(i * 10 + 1);
        example[i].setY(i * 10 + 2);
        example[i].setZ(i * 10 + 3);
    }

    for (int i = 0; i < size; i++)
    {
        cout << "Object" << i + 1 << " x = " << example[i].getX() << endl;
        cout << "Object" << i + 1 << " y = " << example[i].getY() << endl;
        cout << "Object" << i + 1 << " z = " << example[i].getZ() << "\n\n";
    }
}
```

Array of objects

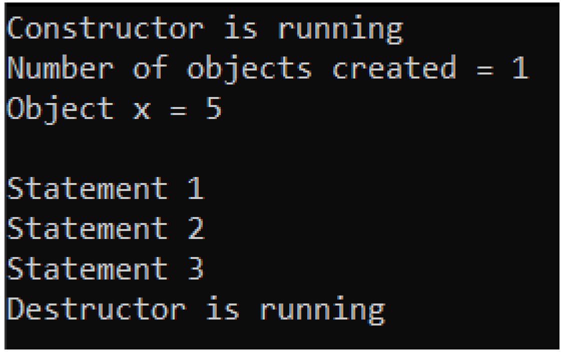


```
Number of objects created = 4
Object1 x = 1
Object1 y = 2
Object1 z = 3

Object2 x = 11
Object2 y = 12
Object2 z = 13

Object3 x = 21
Object3 y = 22
Object3 z = 23

Object4 x = 31
Object4 y = 32
Object4 z = 33
```

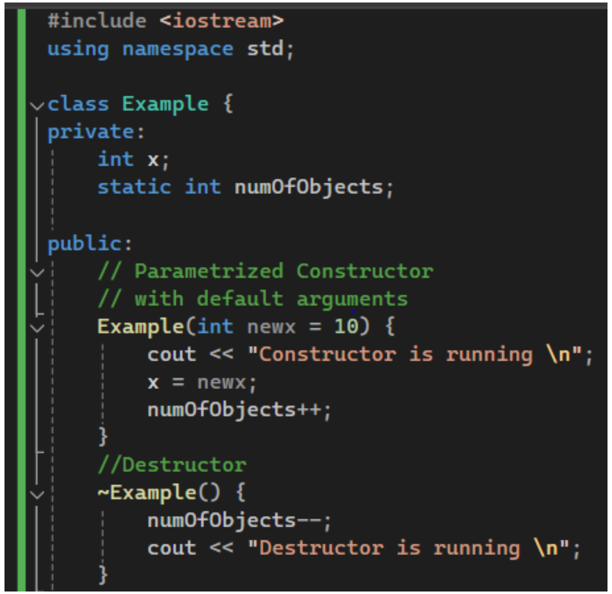
Destructor

- Destructor: is a special member function of a class whose task is to delete memory space occupied by an object when it is destroyed.
- It has the same name of the class but, preceded with the symbol (~)
- It does not have a return type.
- Destructor is automatically executed when an object is destroyed.
- Destructor can not be overloaded as it takes no arguments.

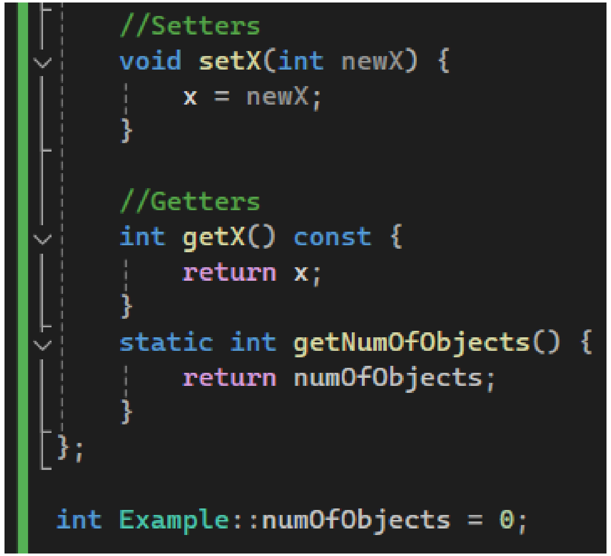
Destructor Ex1:



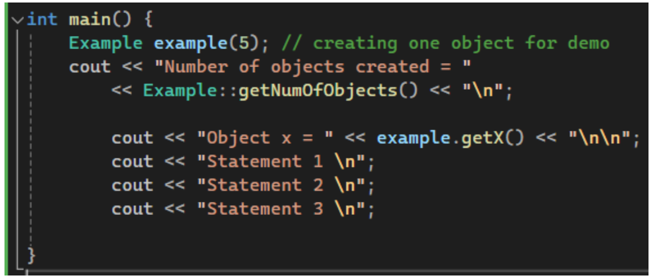
```
Constructor is running
Number of objects created = 1
Object x = 5

Statement 1
Statement 2
Statement 3
Destructor is running
```



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

class Example {
private:
    int x;
    static int numOfObjects;

public:
    // Parametrized Constructor
    // with default arguments
    Example(int newx = 10) {
        cout << "Constructor is running \n";
        x = newx;
        numOfObjects++;
    }
    //Destructor
    ~Example() {
        numOfObjects--;
        cout << "Destructor is running \n";
    }
```

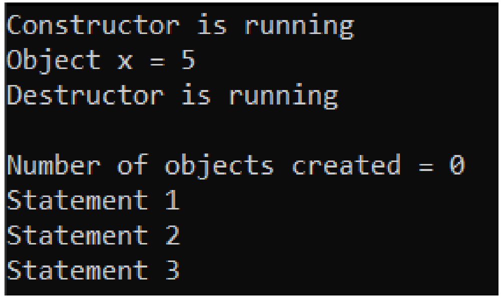


```cpp
    //Setters
    void setX(int newX) {
        x = newX;
    }

    //Getters
    int getX() const {
        return x;
    }
    static int getNumOfObjects() {
        return numOfObjects;
    }
};

int Example::numOfObjects = 0;
```

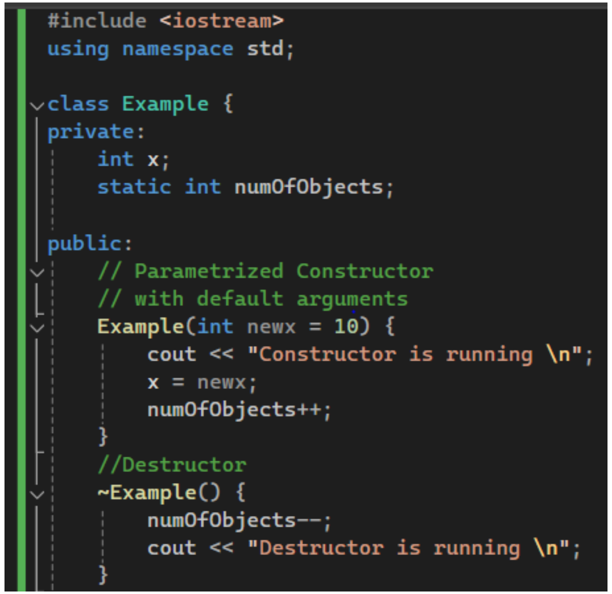


```cpp
int main() {
    Example example(5); // creating one object for demo
    cout << "Number of objects created = "
        << Example::getNumOfObjects() << "\n";

        cout << "Object x = " << example.getX() << "\n\n";
        cout << "Statement 1 \n";
        cout << "Statement 2 \n";
        cout << "Statement 3 \n";

}
```

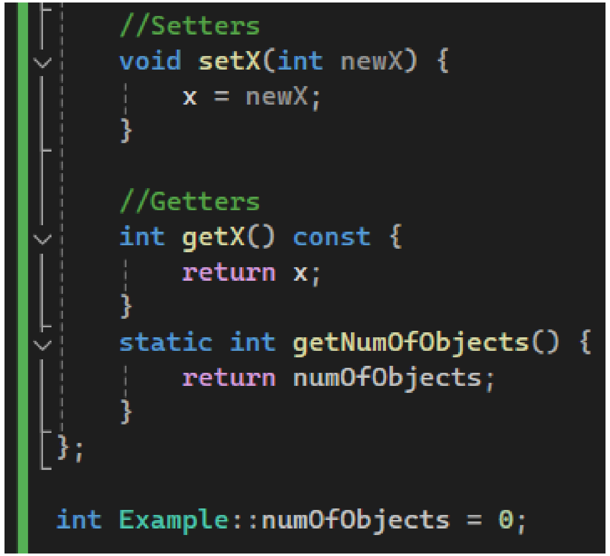
Destructor Ex2:



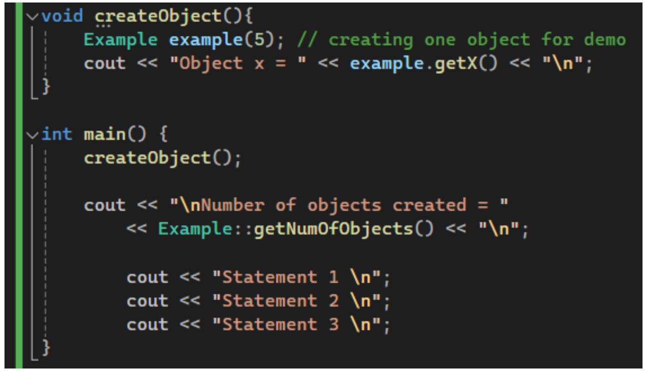
```
Constructor is running
Object x = 5
Destructor is running

Number of objects created = 0
Statement 1
Statement 2
Statement 3
```



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

class Example {
private:
    int x;
    static int numOfObjects;

public:
    // Parametrized Constructor
    // with default arguments
    Example(int newx = 10) {
        cout << "Constructor is running \n";
        x = newx;
        numOfObjects++;
    }
    //Destructor
    ~Example() {
        numOfObjects--;
        cout << "Destructor is running \n";
    }
```



```cpp
    //Setters
    void setX(int newX) {
        x = newX;
    }

    //Getters
    int getX() const {
        return x;
    }
    static int getNumOfObjects() {
        return numOfObjects;
    }
};

int Example::numOfObjects = 0;
```



```cpp
void createObject(){
    Example example(5); // creating one object for demo
    cout << "Object x = " << example.getX() << "\n";
}

int main() {
    createObject();

    cout << "\nNumber of objects created = "
        << Example::getNumOfObjects() << "\n";

        cout << "Statement 1 \n";
        cout << "Statement 2 \n";
        cout << "Statement 3 \n";
}
```

Coding Example