National University
of computer and emerging sciences

Lab 8

Items	**Description**
Course Title	Object Oriented Programming
Lab Title	Classes
Duration	3 Hours
Tools	Eclipse/ C++
Objective	To get familiar with the use of different concepts in classes in c++

1. Member Initializer List

Definition: A *member initializer list* is used in C++ to initialize class member variables before the constructor's body is executed. This is important when initializing const members, reference members, or members that don’t have default constructors.

Use:

- It ensures that the members are initialized only once, improving efficiency.
- Essential when initializing constant data members or reference types which must be initialized directly.
- Helps in initializing objects of classes that don’t provide default constructors.

```
class Example {
    int a;
    const int b;

public:
    Example(int x, int y) : a(x), b(y) {  // Member
initializer list
        // Constructor body
    }
};

int main()
{
    Example ex(10,20);
}
```

Advantages:

- Improves performance since it avoids default construction followed by assignment.
- Necessary for initializing const or reference members.

2. `this` Pointer

Definition: In C++, the `this` pointer is an implicit parameter to all non-static member functions. It points to the object for which the member function is called. Using `this` inside a member function allows you to access the current object.

Uses:

- Refers to the current object when dealing with ambiguous variable names.
- Used in operator overloading and returning the current object to allow for chaining.
- Allows the object to return a pointer to itself.

```
class Example {
    int x;

public:
    Example(int x) {
        this->x = x;   // Use this to resolve ambiguity
    }

    Example& setX(int x) {
        this->x = x;
        return *this;  // Enables function chaining
    }
};
```

Advantages:

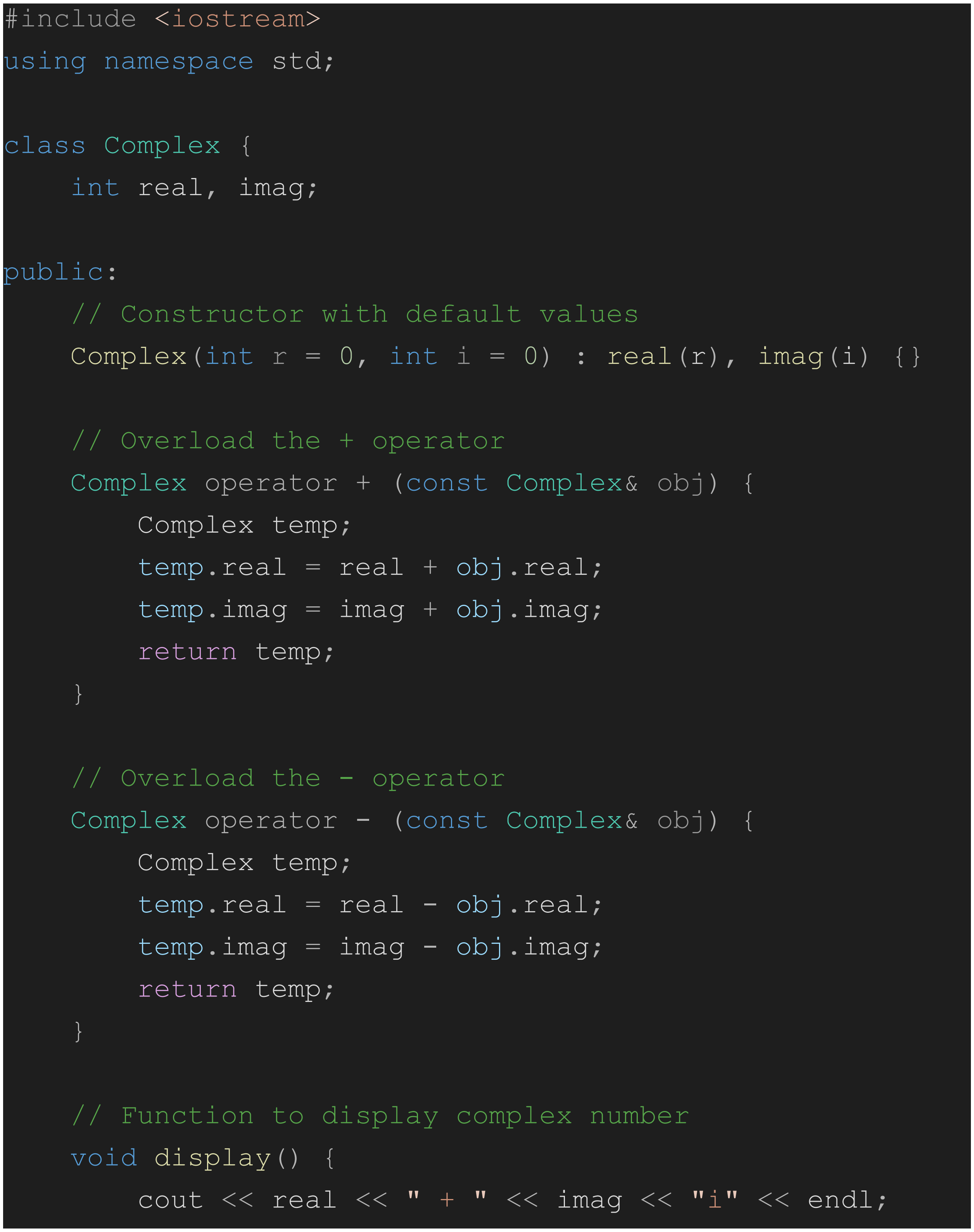
- Helps differentiate between member variables and function parameters.
- Enables method chaining by returning `*this`.

3. Operator Overloading

Definition: Operator overloading allows you to define custom behaviors for operators (like `+`, `-`, `=`, etc.) when they are used with user-defined types such as objects of a class. This enhances readability and intuitive use of objects in C++.

Use:

- Enables intuitive operations on objects (e.g., adding two objects).
- Can be used to simplify complex operations involving user-defined types.
- Improves code readability and allows user-defined types to behave like primitive types.



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

class Complex {
    int real, imag;

public:
    // Constructor with default values
    Complex(int r = 0, int i = 0) : real(r), imag(i) {}

    // Overload the + operator
    Complex operator + (const Complex& obj) {
        Complex temp;
        temp.real = real + obj.real;
        temp.imag = imag + obj.imag;
        return temp;
    }

    // Overload the - operator
    Complex operator - (const Complex& obj) {
        Complex temp;
        temp.real = real - obj.real;
        temp.imag = imag - obj.imag;
        return temp;
    }

    // Function to display complex number
    void display() {
        cout << real << " + " << imag << "i" << endl;
    }
};

int main() {
    Complex c1(5, 3), c2(2, 4);

    Complex c3 = c1 + c2;  // Using overloaded + operator
    Complex c4 = c1 - c2;  // Using overloaded - operator

    cout << "c1 + c2 = ";
    c3.display();

    cout << "c1 - c2 = ";
    c4.display();

    return 0;
}
```

Explanation:

- The `operator +` adds the real and imaginary parts of two complex numbers.
- The `operator -` subtracts the real and imaginary parts of two complex numbers.
- The `display()` function is used to output the result in the format of a complex number.

National University
of computer and emerging sciences

Lab Tasks

Task 1: Vector Class with Dynamic Array and Operator Overloading

Problem Statement:

Write a C++ class called `Vector` that represents a mathematical vector in 2D space. The class should:

1. Use a **dynamic array** to store the components of the vector (e.g., x and y).
2. Overload the `+` and `-` operators to add and subtract vectors.
3. Implement a **recursive function** to calculate the **magnitude** of the vector.

Requirements:

- The vector should be represented using a dynamic array of size 2 (for x and y components).
- The overloaded `+` and `-` operators should add and subtract vectors component-wise.
- The function should calculate the magnitude of the vector: magnitude

$$\text{magnitude} = \sqrt{x^2 + y^2}$$

Task 2: Polynomial Class with Dynamic Array and Recursion

Problem Statement:

Write a class `Polynomial` that represents a **polynomial** (e.g., 3x2+2x+13x^2 + 2x + 13x2+2x+1). The class should:

1. Use a **dynamic array** to store the coefficients of the polynomial.
2. Overload the `+` and `-` operators to add and subtract polynomials.
3. Implement a **recursive function** to evaluate the polynomial for a given value of xxx using recursion.

Requirements:

- Use dynamic memory allocation to store the coefficients of the polynomial.

- The + and - operators should add and subtract polynomials component-wise (i.e., add/subtract corresponding coefficients).
- The function should evaluate the polynomial for a given value of x:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

Task 3: Dynamic Array Manager with Recursive Input and Operator Overloading

Problem Statement:

Write a class `ArrayManager` that:

1. Uses a **dynamic array** to store a list of integers.
2. Overloads the + and - operators to concatenate and remove elements from the dynamic array.
3. Recursively takes input to fill the dynamic array.

Requirements:

- The dynamic array should hold integers.
- The + operator should concatenate two arrays.
- The - operator should remove elements from the second array if they exist in the first array.
- Recursively input values to fill the dynamic array

Output:

```
ArrayManager arr1({1, 2, 3});
ArrayManager arr2({3, 4, 5});
ArrayManager arr3 = arr1 + arr2; // Should result in {1, 2, 3, 3, 4, 5}
ArrayManager arr4 = arr1 - arr2; // Should result in {1, 2}
```