جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

قسم الامن السيبراني

Department of Cyber Security

Subject:

Object Oriented Programming (OOP)

Class:

Second

Lecturer:

Dr. Abdulkadhem A. Abdulkadhem

Lecture: (4)

Objects and Member Functions in Object-Oriented Programming

1. Objects and Member Access:

Objects are instances of a class, and through them, we can access the class's members (both attributes and functions). The two key concepts related to member access are:

1. **Public Members:** Accessible from outside the class.
2. **Private Members:** Accessible only from within the class.

When using objects, we access class members using the dot operator (.) for direct member access, or through member functions which are responsible for interacting with private data.

Example:

```
#include <iostream>
using namespace std;
class Student {
public:
    string name;
    int age;
    void displayInfo() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Student student1;
    student1.name = "Alice";
    student1.age = 20;
    student1.displayInfo(); // Accessing member function
    return 0;
}
```

Explanation:

- We define a Student class with two public members (name and age).

- The displayInfo() function is a member function that prints the values of the name and age attributes.
- In main(), we create a Student object (student1) and access its members using the dot operator. We call displayInfo() to display the student's information.

2. Defining Member Functions:

There are two ways to define member functions in C++:

1. **Inside the class (Inline functions):** Member functions defined directly inside the class body.
2. **Outside the class (Using scope resolution):** Member functions defined outside the class using the scope resolution operator ::.

Example (Member Function Defined Inside Class):

```
#include <iostream>
using namespace std;
class Rectangle {
public:
    int length, width;
    void setDimensions(int l, int w) {
        length = l;
        width = w;
    }
    int calculateArea() {
        return length * width;
    }
};
int main() {
    Rectangle rect;
    rect.setDimensions(5, 10);
    cout << "Area: " << rect.calculateArea() << endl;
    return 0;
}
```

Explanation:

- The setDimensions() function sets the values for length and width attributes, while calculateArea() computes the area.
- Both member functions are defined inside the class itself, making them **inline functions**.

Example (Member Function Defined Outside Class):

```
#include <iostream>
using namespace std;
class Circle {
private:
   double radius;
public:
   void setRadius(double r);
   double calculateArea();
};
void Circle::setRadius(double r) {
   radius = r;
}
double Circle::calculateArea() {
   return 3.14 * radius * radius;
}
int main() {
   Circle circle;
   circle.setRadius(7);
   cout << "Area of the circle: " << circle.calculateArea() << endl;
   return 0;
}
```

Explanation:

- The setRadius() and calculateArea() functions are defined **outside** the class using the scope resolution operator ::.
- This is useful when you want to keep the class definition clean, especially for large functions.

3. Object as Function Arguments:

In C++, you can pass objects as arguments to functions in two ways:

1. **By Value:** The object is copied, and any changes made inside the function do not affect the original object.
2. **By Reference:** The original object is passed, allowing the function to modify its contents.

Example of Passing Objects by Value:

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

class Book {
public:
    string title;
    int pages;
    void setDetails(string t, int p) {
        title = t;
        pages = p;
    }
    void display() {
        cout << "Title: " << title << ", Pages: " << pages << endl;
    }
};
void printBook(Book b) {  // Object passed by value
    b.title = "New Title";
    b.display();          // Changes do not affect the original object
}
int main() {
    Book book1;
    book1.setDetails("C++ Programming", 350);
    printBook(book1);     // Passing by value
    book1.display();      // Original object remains unchanged
    return 0;
}
```

Explanation:

- In printBook(), the Book object is passed **by value**. The function receives a copy of the object, so changes made inside printBook() do not affect the original book1 object.

Example of Passing Objects by Reference:

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

class Book {
public:
    string title;
    int pages;
    void setDetails(string t, int p) {
        title = t;
        pages = p;
    }
    void display() {
        cout << "Title: " << title << ", Pages: " << pages << endl;
    }
};
void modifyBook(Book &b) {  // Object passed by reference
    b.title = "Advanced C++";
}
int main() {
    Book book1;
    book1.setDetails("C++ Programming", 350);
    modifyBook(book1);      // Passing by reference
    book1.display();        // The original object is modified
    return 0;
}
```

Explanation:

- In modifyBook(), the Book object is passed **by reference**. Any changes made inside the function affect the original book1 object, as demonstrated by the change in the title.

4. Object as Return Type:

A function can also return an object. This is useful when we want to return a modified object or create new objects within a function.

Example of Object as Return Type:

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

class Complex {
private:
   int real, imag;

public:
   void setValues(int r, int i) {
      real = r;
      imag = i;
   }

   void display() {
      cout << "Complex number: " << real << " + " << imag << "i" << endl;
   }

   Complex add(Complex c) {
      Complex result;
      result.real = real + c.real;
      result.imag = imag + c.imag;
      return result;
   }
};

int main() {
   Complex c1, c2, sum;
   c1.setValues(3, 4);
   c2.setValues(5, 6);

   sum = c1.add(c2);  // Returning an object from the function
```

```
    sum.display();     // Displaying the result
    return 0;
}
```

Explanation:

- The add() function returns a new Complex object that represents the sum of two complex numbers.
- This demonstrates how functions can create and return new objects.

5. Conclusion:

- **Member Access:** Objects access class members via the dot operator or member functions.
- **Defining Member Functions:** Functions can be defined either inside or outside the class.
- **Objects as Function Arguments:** Objects can be passed by value (copies the object) or by reference (modifies the original object).
- **Objects as Return Type:** Functions can return objects, enabling more flexible manipulation of class instances.

Questions about the lecture

1. What operator is used to access members (attributes or functions) of an object in C++?
2. What is the difference between public and private members in a class?
3. What keyword or concept allows functions to access private data members indirectly?
4. When a member function is defined inside a class, what type of function is it considered?
5. Which operator is used to define a member function outside the class body?
6. What happens when an object is passed **by value** to a function?
7. What happens when an object is passed **by reference** to a function?
8. In the given `Complex` class example, what is the return type of the `add()` function?
9. Why might a programmer prefer defining member functions outside the class body?
10. How does returning an object from a function enhance flexibility in C++ programming?