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: (8)

Arrays as Class Data Members

In C++, arrays can be used as data members in a class. They allow you to store multiple values or objects of the same type within a single class. This is particularly useful when handling multiple elements that belong to the same category or when performing operations on a group of objects.

This lecture covers:

1. **Arrays as Class Data Members**
2. **Object Arrays**
3. **An Array of Pointers to Objects**

---

1. Arrays as Class Data Members

When an array is used as a data member in a class, it can store multiple values related to that class. Arrays can be of basic data types (like `int` or `float`) or user-defined types (like objects of a class). This approach allows encapsulation of multiple values or objects within a single class instance.

Example: Storing Marks of Multiple Subjects

In the following example, a class `Student` has an array `marks` as a data member to store the scores of multiple subjects for a single student.

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

class Student {
private:
    int marks[5];         // Array to store marks of 5 subjects

public:
    void setMarks(int m[]) {
        for (int i = 0; i < 5; ++i) {
            marks[i] = m[i];
        }
    }

    void displayMarks() const {
        cout << "Marks: ";
        for (int i = 0; i < 5; ++i) {
            cout << marks[i] << " ";
        }
        cout << endl;
```

```
    }
};

int main() {
    Student stu1;
    int subjectMarks[5] = {90, 85, 76, 88, 92};

    stu1.setMarks(subjectMarks);
    stu1.displayMarks();

    return 0;
}
```

Explanation:

- The `Student` class has an array `marks` to store scores of 5 subjects.
- The `setMarks` function accepts an array as input and assigns it to the `marks` array.
- The `displayMarks` function outputs the marks stored in the array.

Using the constructor to initialize the array - see the appendix

2. Object Arrays

An **Object Array** is an array where each element is an object of a particular class. This is useful when dealing with multiple instances of a class.

Example: Managing Multiple Employees Using an Object Array

In this example, the `Company` class has an array of `Employee` objects as a data member. Each `Employee` object contains information about individual employees.

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

class Employee {
private:
    string name;
    int id;
public:
    void setData(string n, int i) {
        name = n;
        id = i;
    }
    void display() const {
        cout << "Employee ID: " << id << ", Name: " << name << endl;
    }
};

class Company {
private:
    Employee employees[3]; // Array of Employee objects
public:
    void setEmployeeData() {
```

```
        string name;
        int id;
        for (int i = 0; i < 3; ++i) {
          cout << "Enter ID and name for employee " << (i + 1) << ": ";
          cin >> id >> name;
          employees[i].setData(name, id);
        }
    }

    void displayEmployees() const {
        cout << "Company Employees:" << endl;
        for (int i = 0; i < 3; ++i) {
            employees[i].display();
        }
    }
};

int main() {
    Company company;

    company.setEmployeeData();
    company.displayEmployees();

    return 0;
}
```

Explanation:

- `Company` class contains an array of `Employee` objects.
- The `setEmployeeData` method allows input of employee details for each object in the array.
- The `displayEmployees` method outputs the details of each employee in the array.

3. An Array of Pointers to Objects

An **Array of Pointers to Objects** is an array where each element is a pointer to an object. This allows dynamic allocation and more flexibility, as you can decide when to create or delete objects.

Example: Managing Library Books Using an Array of Pointers to Objects

In this example, the `Library` class has an array of pointers to `Book` objects. This allows creating books only as needed.

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

class Book {
private:
    string title;
    string author;
```

```
public:
    Book(string t, string a) : title(t), author(a) {}

    void display() const {
        cout << "Title: " << title << ", Author: " << author << endl;
    }
};

class Library {
private:
    Book* books[5]; // Array of pointers to Book objects
    int count;

public:
    Library() : count(0) {}

    void addBook(string title, string author) {
        if (count < 5) {
            books[count] = new Book(title, author); // Dynamically
allocate a new Book
            ++count;
        } else {
            cout << "Library is full." << endl;
        }
    }

    void displayBooks() const {
        cout << "Library Books:" << endl;
        for (int i = 0; i < count; ++i) {
            books[i]->display();
        }
    }

    ~Library() {
        for (int i = 0; i < count; ++i) {
            delete books[i]; // Free allocated memory
        }
    }
};

int main() {
    Library library;

    library.addBook("1984", "George Orwell");
    library.addBook("To Kill a Mockingbird", "Harper Lee");

    library.displayBooks();

    return 0;
}
```

Explanation:

- `Library` has an array of pointers to `Book` objects.
- The `addBook` method dynamically allocates a new `Book` object and stores its pointer in the array.

- The `displayBooks` method displays details of each book.
- In the destructor of `Library`, we use `delete` to free the allocated memory, preventing memory leaks.

Summary

- **Arrays as Class Data Members**: Arrays can store multiple values of basic or user-defined types in a single class.
- **Object Arrays**: An array of objects within a class allows handling multiple instances of that class type.
- **Array of Pointers to Objects**: This approach allows dynamic memory allocation, giving flexibility in creating and deleting objects as needed.

MCQ for the lecture

1. **Which of the following best describes an array used as a data member in a class?**
   A. A tool for storing unrelated variables
   B. A structure that stores multiple values of the same type within one class instance
   C. A pointer that must be allocated using `new`
   D. A feature available only for primitive data types
   E. A method for automatic memory management

2. **Why are arrays commonly included as private data members in a class?**
   A. To allow global modification of the array elements
   B. To expose internal data directly to all class users
   C. To encapsulate multiple related values under one object
   D. To eliminate the need for constructors
   E. To force dynamic memory allocation

3. **What characterizes an object array within a class?**
   A. It stores object names only as strings
   B. It holds multiple objects of the same class type in a fixed-size structure
   C. It allows objects of different classes to be stored together
   D. It dynamically resizes based on program input
   E. It eliminates the need for object initialization

4. **In object arrays, how is memory for each object typically allocated?**
   A. Automatically on the stack as part of the containing class
   B. Using `new[]` only
   C. Using manual memory allocation for each element
   D. By calling `malloc()` internally
   E. Through runtime pointer assignment

5. **Which statement correctly describes an array of pointers to objects?**
   A. It stores objects directly, not addresses
   B. It requires objects to be created dynamically
   C. It prevents any object from being deleted
   D. It forces compile-time initialization of all elements
   E. It cannot store different object instances

6. **Why might a programmer choose an array of pointers instead of an array of objects?**
   A. To ensure faster execution in all cases
   B. To allow dynamic creation and deletion of objects as needed
   C. To guarantee constant memory usage
   D. To avoid using destructors
   E. To prevent objects from being modified

7. **A key difference between an object array and an array of object pointers is:**
   A. Only object arrays support encapsulation
   B. Pointer arrays allow flexible memory management for each element
   C. Object arrays can contain different data types
   D. Pointer arrays cannot store polymorphic objects
   E. Object arrays are created using `new` by default

8. **Which of the following is a risk when using an array of pointers to objects?**
   A. Array index out-of-range errors disappear
   B. The compiler automatically manages memory for all objects
   C. Memory leaks can occur if allocated objects are not deleted
   D. The array cannot store more than one element
   E. Each pointer must reference the same object

9. **Which advantage do arrays offer when used as class data members?**
   A. They allow the class to support an unlimited number of elements
   B. They provide a structured way to store multiple homogeneous values
   C. They remove the need for encapsulation
   D. They automatically resize based on stored values
   E. They allow storing unrelated types in the same container

10. **What is one drawback of using fixed-size arrays inside a class?**
    A. They cannot be declared privately
    B. Their size must be known at compile time
    C. They cannot store integer values
    D. They automatically overwrite old data
    E. They cannot be used in constructors

Appendix

Using a Constructor in the Student Class

There are two common ways to use a constructor with arrays as class data members:

✔ Method 1: Constructor With No Parameters (Default Initialization)

The constructor initializes the array internally.

Example

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

class Student {
private:
    int marks[5];   // Array to store marks of 5 subjects

public:
    // Default constructor initializes the array
    Student() {
        for (int i = 0; i < 5; ++i) {
            marks[i] = 0;
        }
    }

    void displayMarks() const {
        cout << "Marks: ";
        for (int i = 0; i < 5; ++i) {
            cout << marks[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    Student stu1;   // Constructor sets all marks to 0
    stu1.displayMarks();
    return 0;
}
```

✔ Method 2: Parameterized Constructor (Passing an Array to the Constructor)

This is the most practical for your lecture: we pass the subject marks directly when creating the object.

Improved Example Using Constructor

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

class Student {
private:
    int marks[5];     // Array to store marks of 5 subjects

public:
    // Parameterized constructor
    Student(int m[]) {
        for (int i = 0; i < 5; ++i) {
            marks[i] = m[i];
        }
    }

    void displayMarks() const {
        cout << "Marks: ";
        for (int i = 0; i < 5; ++i) {
            cout << marks[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    int subjectMarks[5] = {90, 85, 76, 88, 92};

    Student stu1(subjectMarks);   // Pass array to constructor
    stu1.displayMarks();

    return 0;
}
```

Explanation The constructor

- `Student(int m[])` receives an external array and copies its elements into the class data member `marks[]`.
- Since arrays cannot be directly assigned in C++, each element is copied individually using a loop.
- This approach ensures **encapsulation**, as the marks array can only be initialized through a controlled constructor.