كليـــة العلـــــوم
قسم الأمن السيبراني

**Subject: Object Oriented Programming (OOP)**

**Second Stage**

**Lecturer: Dr. Abdulkadhem A. Abdulkadhem**

**Lecture (10)**

**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 subjectspublic: 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.

**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 objectspublic: 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.