جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

قسم الامن السيبراني
Department of Cyber Security

Subject:

Structured Programming

Class:

1st stage

Lecturer:

Dr. Abdulkadhem A. Abdulkadhem

Lecture: (6)

Array

1. Introduction to Arrays

An **array** is a collection of variables of the same type, stored in contiguous memory locations. Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

In C++, arrays are indexed starting from 0, meaning the first element of an array has index 0, the second has index 1, and so on.

2. Array of One Dimension (1D Array)

A **one-dimensional array** (1D array) is simply an ordered list of elements of the same type. It is the most basic form of an array, representing a single row of data.

Syntax:

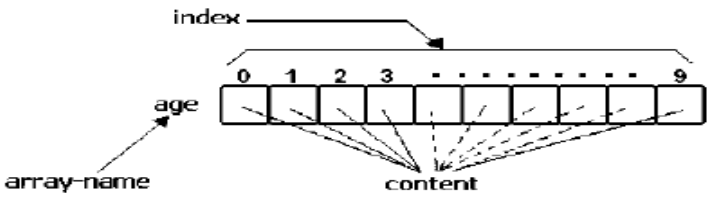
General Form of 1D-Array:
data-type Array-name [*size*];

Where:

- **datatype**: the type of elements the array will hold (e.g., int, float, char, etc.)
- **array_name**: the name of the array
- **size**: the number of elements the array can hold.

Example:

```
int     age [10];
int     num [30];
float   degree[5];
char    a [15];
```



Here, `age` is an array that can hold 10 integers, indexed from 0 to 9.

3. Initializing Array Elements

Arrays can be initialized in multiple ways in C++:

a. Static Initialization

If you know the values that should be stored in the array at the time of declaration, you can initialize the array with those values.

```
int numbers[5] = {10, 20, 30, 40, 50};
```

This initializes the array `numbers` with the specified values. If the number of elements in the initialization list is less than the size of the array, the remaining elements will be initialized to zero.

```
int numbers[5] = {10, 20};            // {10, 20, 0, 0, 0}
```

b. Default Initialization (without values)

If no values are provided, elements of a local array are uninitialized and may contain garbage values.

```
int numbers[5];      // Uninitialized array, contains garbage values
```

c. Zero Initialization

You can also initialize an array to zero by specifying only one value (`0`), which will initialize all elements to zero.

```
int numbers[5] = {0};                          // {0, 0, 0, 0, 0}
```

4. Accessing Array Elements

To access an element of an array, use the index of the element within square brackets.

```
int numbers[5] = {10, 20, 30, 40, 50};
int x = numbers[2];                          // Access the 3rd element (index 2)
```

In this example, `numbers[2]` refers to the element at index 2, which is `30`.

5. Reading Array Elements

Reading array elements can be done with the help of a loop, typically a `for` loop, to print or process each element.

Example of reading and printing elements using a loop:

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

int main() {
    int numbers[5] = {10, 20, 30, 40, 50};

    // Reading array elements and printing them
    for (int i = 0; i < 5; i++) {
        cout << "Element at index " << i << " is " << numbers[i] << endl;
    }

    return 0;
}
```

Output:

```
Element at index 0 is 10
Element at index 1 is 20
Element at index 2 is 30
Element at index 3 is 40
Element at index 4 is 50
```

6. Writing to Array Elements

You can modify the values of array elements by directly assigning new values to them using their index.

Example:

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

int main() {
    int numbers[5] = {10, 20, 30, 40, 50};

    // Writing to an array element
    numbers[2] = 100; // Change the 3rd element (index 2) to 100

    // Printing the updated array
    for (int i = 0; i < 5; i++) {
        cout << "Element at index " << i << " is " << numbers[i] << endl;
    }

    return 0;
```

```
}
```

Output:

```
Element at index 0 is 10
Element at index 1 is 20
Element at index 2 is 100
Element at index 3 is 40
Element at index 4 is 50
```

7. Processing Array Elements

You can also process array elements in loops, such as summing all elements, finding the average, etc.

Example: Finding the sum of all elements in an array:

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

int main() {
    int numbers[5] = {10, 20, 30, 40, 50};
    int sum = 0;

    // Process array: calculate sum
    for (int i = 0; i < 5; i++) {
        sum += numbers[i];
    }

    cout << "The sum of the array elements is " << sum << endl;

    return 0;
}
```

Output:

```
The sum of the array elements is 150
```

8. Important Notes on Arrays

- **Array Size**: The size of an array must be known at compile time if you're using a statically declared array. You cannot change the size of an array after it is declared.
- **Bounds Checking**: C++ does not perform bounds checking. Accessing an out-of-bounds index (like `numbers[5]` in a 5-element array) results in undefined behavior.
- **Memory Management**: Arrays are stored in contiguous memory, and you can calculate the memory location of any element using pointer arithmetic.
-

9. Common Operations with Arrays

1. **Finding Maximum/Minimum Value**: Loop through the array to find the maximum or minimum value.
2. **Reversing an Array**: Reverse the order of elements in the array.
3. **Searching for an Element**: Use loops to search for a specific element.

Example1: Finding the largest element in an array:

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

int main() {
    int numbers[5] = {10, 20, 30, 40, 50};
    int max = numbers[0];       // Assume first element is the largest

    for (int i = 1; i < 5; i++) {
        if (numbers[i] > max) {
            max = numbers[i];
        }
    }

    cout << "The largest element is " << max << endl;

    return 0;
}
```

Output:

```
The largest element is 50
```

Example 1: Finding the Average of Array Elements

This example demonstrates how to calculate the average of all elements in an array by summing the elements and then dividing the sum by the number of elements.

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

int main() {
    int numbers[5] = {10, 20, 30, 40, 50};
    int sum = 0;

    // Calculate sum of the array elements
    for (int i = 0; i < 5; i++) {
        sum += numbers[i];
    }

    // Calculate average
    double average = sum / 5;
```

```
    cout << "The average of the array elements is " << average << endl;

    return 0;
}
```

Output:

```
The average of the array elements is 30
```

Example 2: Searching for an Element in an Array

This example shows how to search for a specific element in an array. If the element is found, it returns its index; otherwise, it indicates that the element is not in the array.

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

int main() {
    int numbers[6] = {10, 20, 30, 40, 50, 60};
    int target = 40;
    bool found = false;

    // Search for the target element
    for (int i = 0; i < 6; i++) {
        if (numbers[i] == target) {
            cout << "Element " << target << " found at index " << i << endl;
            found = true;
            break; // Exit loop once found
        }
    }

    if (!found) {
        cout << "Element " << target << " not found in the array." << endl;
    }

    return 0;
}
```

Output:

```
Element 40 found at index 3
```

Example 3: Reversing an Array

This example demonstrates how to reverse the elements of an array in place. The approach is to swap elements from the start and end until the middle of the array is reached.

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

int main() {
    int numbers[5] = {10, 20, 30, 40, 50};

    // Reverse the array
    int start = 0;
    int end = 4;
    while (start < end) {
        // Swap the elements
        int temp = numbers[start];
        numbers[start] = numbers[end];
        numbers[end] = temp;

        start++;
        end--;
    }

    // Print the reversed array
    cout << "Reversed array: ";
    for (int i = 0; i < 5; i++) {
        cout << numbers[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Output:

```
Reversed array: 50 40 30 20 10
```

Conclusion

In this lecture, we covered:

- What arrays are and their use in storing multiple elements of the same type.
- How to declare and initialize a one-dimensional array.
- How to access, read, and write array elements.
- Common operations like processing elements (e.g., sum, maximum) using loops.

Arrays are fundamental in programming, providing a way to manage and manipulate multiple values efficiently. Understanding how to work with arrays is essential for solving a variety of programming problems.