Lecture 06: Operations on One-Dimensional Arrays

Course : Introduction to Programming

Dr. Farouk KECITA

Department of Process Engineering
Faculty of Science and Technology
University of Mila

Level: 1st year ST - ENG & LMD

Academic Year 2025/2026

Outline

Operations on 1-D Arrays

What We'll Cover

This presentation covers various operations that can be performed on one-dimensional arrays in C:

Operations on 1-D Arrays

What We'll Cover

This presentation covers various operations that can be performed on one-dimensional arrays in C:

- Searching (Linear Search)

Operations on 1-D Arrays

What We'll Cover

This presentation covers various operations that can be performed on one-dimensional arrays in C:

- Searching (Linear Search)
- Sorting (Selection Sort)

Operations on 1-D Arrays

What We'll Cover

This presentation covers various operations that can be performed on one-dimensional arrays in C:

- Searching (Linear Search)
- Sorting (Selection Sort)
- Copying Arrays

Operations on 1-D Arrays

What We'll Cover

This presentation covers various operations that can be performed on one-dimensional arrays in C:

- Searching (Linear Search)
- Sorting (Selection Sort)
- Copying Arrays
- Modifying Elements

Operations on 1-D Arrays

What We'll Cover

This presentation covers various operations that can be performed on one-dimensional arrays in C:

- Searching (Linear Search)
- Sorting (Selection Sort)
- Copying Arrays
- Modifying Elements
- Deleting Elements

Operations on 1-D Arrays

What We'll Cover

This presentation covers various operations that can be performed on one-dimensional arrays in C:

- Searching (Linear Search)
- Sorting (Selection Sort)
- Copying Arrays
- Modifying Elements
- Deleting Elements
- Sum of Elements

Operations on 1-D Arrays

What We'll Cover

This presentation covers various operations that can be performed on one-dimensional arrays in C:

- Searching (Linear Search)
- Sorting (Selection Sort)
- Copying Arrays
- Modifying Elements
- Deleting Elements
- Sum of Elements
- Arithmetic Operations

Operations on 1-D Arrays

What We'll Cover

This presentation covers various operations that can be performed on one-dimensional arrays in C:

- Searching (Linear Search)
- Sorting (Selection Sort)
- Copying Arrays
- Modifying Elements
- Deleting Elements
- Sum of Elements
- Arithmetic Operations
- Statistical Analysis

Important

Understanding these operations is fundamental for efficient programming with arrays!

Linear Search (Sequential Search)

Definition

The linear search method is a straightforward approach that compares each array element to the target value sequentially until a match is found or the entire array is traversed.

Characteristics

- Simple to implement
- Works on unsorted arrays
- Time complexity: O(n)
- Best for small arrays

Linear Search - Method 01

```c
#include <stdio.h>
int main() {
    int arr[] = {5, 10, 9, 12};
    int i, index = -1, target;

    printf("The target is: ");
    scanf("%d", &target);

    // Determine the index if the target is found
    for (i = 0; i < 4; i++) {
        if (arr[i] == target) {
            index = i;
        }
    }

    if (index == -1)
        printf("Element not found\n");
    else
        printf("Element found at index: %d\n", index);
    return 0;
}
```

Linear Search - Output Examples

Run 1 - Element Found

```
The target is: 9
Element found at index: 2
```

Run 2 - Element Not Found

```
The target is: 16
Element not found
```

Index 0	Index 1	Index 2	Index 3
5	10	9	12

Linear Search - Method 02 (Alternative)

```c
#include <stdio.h>
int main() {
    int arr[] = {5, 10, 9, 12};
    int i, target;

    printf("The target is: ");
    scanf("%d", &target);

    for (i = 0; i < 4; i++) {
        if (arr[i] == target) {
            printf("Element found at index: %d\n", i);
            return 0;
        }
    }

    printf("Element not found\n");
    return 0;
}
```

Note

This method exits immediately when the element is found, making it more efficient!

Sorting Arrays

What is Sorting?

Sorting involves arranging elements in a particular order, usually ascending or descending.

Applications

- Data analysis
- Data compression
- Searching optimization
- Database management

Selection Sort

The most popular method for educational purposes. It repeatedly finds the minimum element from the unsorted part and puts it at the beginning.

Selection Sort Algorithm - Visualization

Initial Array: [5, 10, 9, 12]			
Step 1	5	10	9
Step 2	5	9	10
Step 3	5	9	10
Step 4	5	9	10

Selection Sort Algorithm - Visualization

Initial Array: [5, 10, 9, 12]			
Step 1	5	10	9
Step 2	5	9	10
Step 3	5	9	10
Step 4	5	9	10

- **Step 1:** Compare 5 with 10, 9, 12 $\rightarrow$ 5 is smallest, keep at position 0

Selection Sort Algorithm - Visualization

Initial Array: [5, 10, 9, 12]			
Step 1	5	10	9
Step 2	5	9	10
Step 3	5	9	10
Step 4	5	9	10

- **Step 1:** Compare 5 with 10, 9, 12 $\rightarrow$ 5 is smallest, keep at position 0
- **Step 2:** Compare 10 with 9 $\rightarrow$ 9 is smaller, swap positions 1 and 2

Selection Sort Algorithm - Visualization

Initial Array: [5, 10, 9, 12]			
Step 1	5	10	9
Step 2	5	9	10
Step 3	5	9	10
Step 4	5	9	10

- **Step 1:** Compare 5 with 10, 9, 12 $\rightarrow$ 5 is smallest, keep at position 0
- **Step 2:** Compare 10 with 9 $\rightarrow$ 9 is smaller, swap positions 1 and 2
- **Step 3:** Compare 10 with 12 $\rightarrow$ 10 is smaller, keep at position 2

Selection Sort Algorithm - Visualization

Initial Array: [5, 10, 9, 12]			
Step 1	5	10	9
Step 2	5	9	10
Step 3	5	9	10
Step 4	5	9	10

- **Step 1:** Compare 5 with 10, 9, 12 $\rightarrow$ 5 is smallest, keep at position 0
- **Step 2:** Compare 10 with 9 $\rightarrow$ 9 is smaller, swap positions 1 and 2
- **Step 3:** Compare 10 with 12 $\rightarrow$ 10 is smaller, keep at position 2
- **Step 4:** Last element (12) automatically in place

Selection Sort - C Program

```c
#include <stdio.h>
int main() {
    int arr[] = {5, 10, 9, 12};
    int temp, i, j;

    printf("Array elements Before Sorting\n");
    for(i = 0; i < 4; i++) {
        printf("arr[%d] = %d\n", i, arr[i]);
    }
    for(i = 0; i < 4; i++) {
        for(j = i + 1; j < 4; j++) {
            if(arr[i] > arr[j]) {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }

    printf("\nArray elements After Sorting\n");
    for(i = 0; i < 4; i++) {
        printf("arr[%d] = %d\n", i, arr[i]);
    }
    return 0;
}
```

Selection Sort - Output

Before Sorting

```
arr[0] = 5
arr[1] = 10
arr[2] = 9
arr[3] = 12
```

After Sorting

```
arr[0] = 5
arr[1] = 9
arr[2] = 10
arr[3] = 12
```

Visual Representation

[5, 10, 9, 12] $\rightarrow$ [5, 9, 10, 12]

Copying 1-D Arrays

Important Note

The statement `array2 = array1;` is WRONG and will produce an error!

Correct Approach

Arrays must be copied element by element using a loop:

```
for(i = 0; i < size; i++) {

    array2[i] = array1[i];

}
```

Requirements

- Size of array2 must be $\geq$ size of array1
- Must know the size of array1 in advance

Copying Arrays - Complete Program

```c
#include <stdio.h>
#define MAX_SIZE 50
int main() {
    int arr1[MAX_SIZE], arr2[MAX_SIZE];
    int i, size;
    printf("Enter the size of arr1: ");
    scanf("%d", &size);
    printf("Enter elements of arr1: \n");
    for(i = 0; i < size; i++) {
        scanf("%d", &arr1[i]);
    }
    // Copy all elements from arr1 to arr2
    for(i = 0; i < size; i++) {
        arr2[i] = arr1[i];
    }
    printf("\nElements of arr1 are: ");
    for(i = 0; i < size; i++) {
        printf("%d\t", arr1[i]);
    }
    printf("\nElements of arr2 are: ");
    for(i = 0; i < size; i++) {
        printf("%d\t", arr2[i]);
    }
    return 0;
}
```

Copying Arrays - Output Example

Input

```
Enter the size of arr1: 4
Enter elements of arr1:
15
18
9
10
```

Output

```
Elements of arr1 are: 15   18   9   10
Elements of arr2 are: 15   18   9   10
```

Both Arrays Contain			
15	18	9	10

Modifying Elements in 1-D Arrays

Syntax

array_name[i] = new_value;

Example

```
int arr[] = {15, 18, 9, 10};
printf("Third element: %d\n", arr[2]);  // Output: 9
arr[2] = 20;  // Modify the third element
printf("Third element now: %d\n", arr[2]); // Output:20
```

Before Modification			
15	18	9	10
After Modification			
15	18	20	10

Deleting an Element from a 1-D Array

Important

Deletion from an array is a costly operation because elements must be shifted.

Steps to Delete an Element

1. Input the position (index) of the element to delete
2. Check if position is valid (index $<$ size)
3. Shift all elements to the left: arr[i] $=$ arr[i+1]
4. Decrement the array size

Visual Example: Deleting element at index 2

[3, 5, 7, 10] $\rightarrow$ [3, 5, 10]

Deleting an Element - C Program

```c
#include <stdio.h>
int main() {
    int i, size, index, arr[10];
    printf("Enter the size of the array: ");
    scanf("%d", &size);
    printf("Enter the elements of the array: \n");
    for (i = 0; i < size; i++) {
        printf("arr[%d] = ", i);
        scanf("%d", &arr[i]);
    }
    printf("Enter the index to delete: ");
    scanf("%d", &index);
    if (index >= size) {
        printf("\nDeletion not possible.\n");
    }
    else {
        for (i = index; i < size - 1; i++) {
            arr[i] = arr[i + 1];
        }
        printf("Array after deletion:\n");
        for (i = 0; i < size - 1; i++) {
            printf("arr[%d] = %d\n", i, arr[i]);
        }
    }
    return 0;
}
```

Deleting an Element - Output Examples

Run 1 - Valid Deletion

```
Enter size: 4
arr[0] = 3
arr[1] = 5
arr[2] = 7
arr[3] = 10
Enter index: 2

After deletion:
arr[0] = 3
arr[1] = 5
arr[2] = 10
```

Run 2 - Invalid Index

```
Enter size: 4
arr[0] = 1
arr[1] = 2
arr[2] = 5
arr[3] = 7
Enter index: 4

Deletion not possible.
```

Sum of 1-D Array Elements

Algorithm

1. Declare a variable `sum` and initialize to 0
2. Loop through all elements
3. Add each element to sum: `sum = sum + arr[i]`
4. Print the result

Formula

$$\text{sum} = \sum_{i=0}^{n-1} \text{arr}[i]$$

Sum of Array Elements - C Program

```c
#include <stdio.h>
int main() {
    int arr[100], size, i, sum = 0;
    printf("Enter array size: ");
    scanf("%d", &size);
    printf("Enter array elements: \n");
    for(i = 0; i < size; i++) {
        scanf("%d", &arr[i]);
    }
    for(i = 0; i < size; i++) {
        sum = sum + arr[i];  // sum += arr[i];
    }
    printf("Sum of the array = %d\n", sum);
    return 0;
}
```

Output

```
Enter array size: 4
Enter array elements: 12 8 10 20
Sum of the array = 50
```

Arithmetic Operations on Arrays

Element-wise Operations

Operations are performed on corresponding elements of two arrays.

Operations

- Addition: `c[i] = a[i] + b[i]`
- Subtraction: `c[i] = a[i] - b[i]`
- Multiplication: `c[i] = a[i] * b[i]`
- Division: `c[i] = a[i] / b[i]`

Example:

$$[1, 2, 3] + [4, 5, 6] = [5, 7, 9]$$
$$[10, 8, 6] - [1, 2, 3] = [9, 6, 3]$$
$$[2, 3, 4] \times [3, 2, 1] = [6, 6, 4]$$

Adding Two Arrays - C Program

```c
#include <stdio.h>
#define SIZE 5
int main() {
    int i, arr1[SIZE], arr2[SIZE], sum[SIZE];
    printf("Enter the first array:\n");
    for(i = 0; i < SIZE; i++) {
        printf("arr1[%d] = ", i);
        scanf("%d", &arr1[i]);
    }
    printf("Enter the second array:\n");
    for(i = 0; i < SIZE; i++) {
        printf("arr2[%d] = ", i);
        scanf("%d", &arr2[i]);
    }
    for(i = 0; i < SIZE; i++) {
        sum[i] = arr1[i] + arr2[i];
    }
    printf("Sum of two arrays is:\n");
    for(i = 0; i < SIZE; i++) {
        printf("sum[%d] = %d + %d = %d\n",
                i, arr1[i], arr2[i], sum[i]);
    }
    return 0;
}
```

Adding Two Arrays - Output Example

First Array

```
arr1[0] = 12
arr1[1] = 13
arr1[2] = 4
arr1[3] = 20
arr1[4] = 18
```

Second Array

```
arr2[0] = 20
arr2[1] = 12
arr2[2] = 16
arr2[3] = 18
arr2[4] = 9
```

Result

```
sum[0] = 12 + 20 = 32
sum[1] = 13 + 12 = 25
sum[2] = 4 + 16 = 20
sum[3] = 20 + 18 = 38
sum[4] = 18 + 9 = 27
```

Statistical Analysis on Arrays

Statistical Measures

- **Mean** ($\bar{X}$): Average of all values
- **Variance** (σ^2): Measure of spread
- **Standard Deviation** (σ): Square root of variance

Formulas

$$\bar{X} = \frac{\sum_{i=1}^{N} X_i}{N}$$

$$\sigma^2 = \frac{\sum_{i=1}^{N} (X_i - \bar{X})^2}{N}$$

$$\sigma = \sqrt{\sigma^2}$$

Statistical Analysis - Algorithm

Steps

1. Read N items (student scores)

Statistical Analysis - Algorithm

Steps

1. Read N items (student scores)
2. Calculate sum and mean

Statistical Analysis - Algorithm

Steps

1. Read N items (student scores)
2. Calculate sum and mean
3. Calculate variance

Statistical Analysis - Algorithm

Steps

1. Read N items (student scores)
2. Calculate sum and mean
3. Calculate variance
4. Calculate standard deviation

Statistical Analysis - Algorithm

Steps

1. Read N items (student scores)
2. Calculate sum and mean
3. Calculate variance
4. Calculate standard deviation

Example Data

Scores: [85, 90, 78, 92, 88]
Number of students: 5

Statistical Analysis - C Program

```c
#include <stdio.h>
#include <math.h>
int main() {
    int scores[100], N, i;
    float sum = 0, mean, variance = 0, std_deviation;
    printf("Enter the number of students: ");
    scanf("%d", &N);
    printf("Enter the scores of %d students:\n", N);
    for(i = 0; i < N; i++) {
        printf("Score %d: ", i + 1);
        scanf("%d", &scores[i]);
        sum += scores[i];
        }
    // Calculate mean
    mean = sum / N;
    // Calculate variance
    for(i = 0; i < N; i++) {
        variance += pow((scores[i] - mean), 2);
    }
    variance = variance / N;
    // Calculate standard deviation
    std_deviation = sqrt(variance);
    // Display results
    printf("Number of students: %d\n", N);
    printf("Sum of scores: %.2f\n", sum);
    printf("Mean (Average) score: %.2f\n", mean);
    printf("Variance: %.2f\n", variance);
    printf("Standard Deviation: %.2f\n", std_deviation);
    return 0;  }
```

Statistical Analysis - Output Example

Input

```
Enter the number of students: 5
Enter the scores of 5 students:
Score 1: 85
Score 2: 90
Score 3: 78
Score 4: 92
Score 5: 88
```

Results

```
Number of students: 5
Sum of scores: 433.00
Mean (Average) score: 86.60
Variance: 22.24
Standard Deviation: 4.72
```

Summary of Array Operations

Operation	**Description**
Searching	Find if an element exists in array
Sorting	Arrange elements in order
Copying	Duplicate array contents
Modifying	Change element value
Deleting	Remove an element
Summation	Calculate total of all elements
Arithmetic	Perform math operations on two arrays
Statistical	Calculate mean, variance, std deviation

Key Takeaways

Important Points

- Arrays are fundamental for storing collections of similar data

Key Takeaways

Important Points

- Arrays are fundamental for storing collections of similar data
- Most array operations require iterating through elements

Key Takeaways

Important Points

- Arrays are fundamental for storing collections of similar data
- Most array operations require iterating through elements
- Array indices start at 0 in C

Key Takeaways

Important Points

- Arrays are fundamental for storing collections of similar data
- Most array operations require iterating through elements
- Array indices start at 0 in C
- Arrays cannot be assigned directly (must copy element by element)

Key Takeaways

Important Points

- Arrays are fundamental for storing collections of similar data
- Most array operations require iterating through elements
- Array indices start at 0 in C
- Arrays cannot be assigned directly (must copy element by element)
- Understanding array operations is essential for efficient programming

Key Takeaways

Important Points

- Arrays are fundamental for storing collections of similar data
- Most array operations require iterating through elements
- Array indices start at 0 in C
- Arrays cannot be assigned directly (must copy element by element)
- Understanding array operations is essential for efficient programming
- Statistical analysis helps in data interpretation

Key Takeaways

Important Points

- Arrays are fundamental for storing collections of similar data
- Most array operations require iterating through elements
- Array indices start at 0 in C
- Arrays cannot be assigned directly (must copy element by element)
- Understanding array operations is essential for efficient programming
- Statistical analysis helps in data interpretation

Remember

Practice these operations regularly to master array manipulation in C!