C Programming: Arrays (1D)

Prof. Jyotiprakash Mishra
mail@jyotiprakash.org

January 16, 2026

Topics Covered

1. Introduction to Arrays
2. Array Initialization
3. Array Input and Output
4. Array Operations
5. Array Manipulation
6. Array Analysis
7. Array Comparison
8. Frequency and Duplicates
9. Common Mistakes
10. Summary

What are Arrays?

- Collection of elements of same type
- Stored in contiguous memory locations
- Fixed size (determined at declaration)
- Elements accessed by index (0-based)
- Efficient for storing related data

Why Use Arrays?

- Store multiple values in one variable
- Process collections of data
- Use loops to access elements
- More efficient than separate variables

Index Range:

- Array of size n: indices 0 to n-1
- First element: index 0
- Last element: index n-1

Array Declaration and Syntax

Declaration:

```c
data_type array_name[size];
```

Examples:

```c
int numbers[5];        // array of 5 integers
float scores[10];      // array of 10 floats
char letters[26];      // array of 26 characters
```

Accessing Elements:

```c
array_name[index]      // index from 0 to size-1
numbers[0]             // first element
numbers[4]             // fifth element (last in size 5)
```

Important: Size must be a constant or known at compile time

Program 1: Array Declaration and Initialization

```c
#include <stdio.h>
int main() {
  int arr1[5] = {10, 20, 30, 40, 50};
  int arr2[5] = {1, 2, 3};
  int arr3[] = {5, 10, 15, 20};
  int i;
  printf("arr1: ");
  for (i = 0; i < 5; i++) {
    printf("%d ", arr1[i]);
  }
  printf("\narr2: ");
  for (i = 0; i < 5; i++) {
    printf("%d ", arr2[i]);
  }
  printf("\narr3 size: %lu\n",
         sizeof(arr3)/sizeof(arr3[0]));
  return 0;
}
```

Output:

```
arr1: 10 20 30 40 50
arr2: 1 2 3 0 0
arr3 size: 4
```

Explanation:

- arr1: fully initialized
- arr2: partial init, rest are 0
- arr3: size inferred as 4
- Uninitialized elements are 0

Program 2: Zero Initialization

```c
#include <stdio.h>
int main() {
  int arr1[5] = {0};
  int arr2[5] = {};
  int arr3[5];
  int i;
  printf("arr1 (={0}): ");
  for (i = 0; i < 5; i++) {
    printf("%d ", arr1[i]);
  }
  printf("\narr2 (={}): ");
  for (i = 0; i < 5; i++) {
    printf("%d ", arr2[i]);
  }
  printf("\narr3 (no init): ");
  for (i = 0; i < 5; i++) {
    printf("%d ", arr3[i]);
  }
  printf("\n");
  return 0;
}
```

Output:

```
arr1 (={0}): 0 0 0 0 0
arr2 (={}): 0 0 0 0 0
arr3 (no init): 0 0 0 0 0
```

Note:

- {0} sets all to zero
- {} also sets all to zero
- Uninitialized: undefined (often 0 for global/static)
- Always initialize arrays!

Program 3: Individual Element Assignment

```c
#include <stdio.h>
int main() {
  int arr[5];
  int i;
  arr[0] = 100;
  arr[1] = 200;
  arr[2] = 300;
  arr[3] = 400;
  arr[4] = 500;
  printf("Array elements:\n");
  for (i = 0; i < 5; i++) {
    printf("arr[%d] = %d\n", i, arr[i]);
  }
  return 0;
}
```

Output:

```
Array elements:
arr[0] = 100
arr[1] = 200
arr[2] = 300
arr[3] = 400
arr[4] = 500
```

Note:

- Elements assigned individually
- Index starts at 0
- Index ends at size-1

Program 4: Array Input from User

```c
#include <stdio.h>
int main() {
  int arr[5];
  int i;
  int inputs[] = {5, 10, 15, 20, 25};
  printf("Enter 5 numbers:\n");
  for (i = 0; i < 5; i++) {
    arr[i] = inputs[i];
    printf("%d ", inputs[i]);
  }
  printf("\n\nYou entered:\n");
  for (i = 0; i < 5; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
  return 0;
}
```

Output:

```
Enter 5 numbers:
5 10 15 20 25

You entered:
5 10 15 20 25
```

Pattern:

- Loop to read input
- Store in array using index
- Loop again to display

Program 5: Print Array Elements

```c
#include <stdio.h>
int main() {
  int numbers[] = {2, 4, 6, 8, 10};
  int size = 5;
  int i;
  printf("Method 1 - Horizontal:\n");
  for (i = 0; i < size; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n\nMethod 2 - Vertical:\n");
  for (i = 0; i < size; i++) {
    printf("numbers[%d] = %d\n",
           i, numbers[i]);
  }
  return 0;
}
```

Output:

```
Method 1 - Horizontal:
2 4 6 8 10

Method 2 - Vertical:
numbers[0] = 2
numbers[1] = 4
numbers[2] = 6
numbers[3] = 8
numbers[4] = 10
```

Note:

- Two display methods
- Loop through all elements

Program 6: Sum of Array Elements

```c
#include <stdio.h>
int main() {
  int arr[] = {10, 20, 30, 40, 50};
  int size = 5;
  int sum = 0;
  int i;
  printf("Array: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
    sum += arr[i];
  }
  printf("\n\nSum = %d\n", sum);
  printf("Average = %.2f\n",
         (float)sum / size);
  return 0;
}
```

Output:

```
Array: 10 20 30 40 50

Sum = 150
Average = 30.00
```

Logic:

- Initialize sum to 0
- Loop through array
- Add each element to sum
- Calculate average

Program 7: Find Maximum Element

```c
#include <stdio.h>
int main() {
  int arr[] = {34, 12, 89, 5, 67, 23};
  int size = 6;
  int max = arr[0];
  int maxIndex = 0;
  int i;
  for (i = 1; i < size; i++) {
    if (arr[i] > max) {
      max = arr[i];
      maxIndex = i;
    }
  }
  printf("Array: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n\nMax: %d at index %d\n",
         max, maxIndex);
  return 0;
}
```

Output:

```
Array: 34 12 89 5 67 23

Max: 89 at index 2
```

Logic:

- Assume first element is max
- Compare with each element
- Update max if larger found
- Track index too

Program 8: Find Minimum Element

```c
#include <stdio.h>
int main() {
  int arr[] = {34, 12, 89, 5, 67, 23};
  int size = 6;
  int min = arr[0];
  int minIndex = 0;
  int i;
  for (i = 1; i < size; i++) {
    if (arr[i] < min) {
      min = arr[i];
      minIndex = i;
    }
  }
  printf("Array: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n\nMin: %d at index %d\n",
         min, minIndex);
  return 0;
}
```

Output:

```
Array: 34 12 89 5 67 23

Min: 5 at index 3
```

Logic:

- Assume first element is min
- Compare with each element
- Update min if smaller found
- Same pattern as max

Program 9: Linear Search

```c
#include <stdio.h>
int main() {
  int arr[] = {10, 25, 30, 15, 20};
  int size = 5;
  int target = 15;
  int found = 0;
  int i;
  printf("Array: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\nSearching for: %d\n\n",
         target);
  for (i = 0; i < size; i++) {
    if (arr[i] == target) {
      printf("Found at index %d\n", i);
      found = 1;
      break;
    }
  }
  if (!found) {
    printf("Not found\n");
  }
  return 0;
}
```

Output:

```
Array: 10 25 30 15 20
Searching for: 15

Found at index 3
```

Logic:

- Check each element
- If match, set found flag
- Break out of loop
- Report result

Program 10: Count Occurrences

```c
#include <stdio.h>
int main() {
  int arr[] = {5, 2, 5, 8, 5, 3, 5};
  int size = 7;
  int target = 5;
  int count = 0;
  int i;
  printf("Array: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\nTarget: %d\n\n", target);
  for (i = 0; i < size; i++) {
    if (arr[i] == target) {
      count++;
    }
  }
  printf("Count: %d\n", count);
  return 0;
}
```

Output:

```
Array: 5 2 5 8 5 3 5
Target: 5

Count: 4
```

Logic:

- Initialize count to 0
- Loop through array
- Increment count on match
- Don't break - count all

Program 11: Reverse an Array

```c
#include <stdio.h>
int main() {
  int arr[] = {1, 2, 3, 4, 5};
  int size = 5;
  int temp, i;
  printf("Original: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  for (i = 0; i < size/2; i++) {
    temp = arr[i];
    arr[i] = arr[size-1-i];
    arr[size-1-i] = temp;
  }
  printf("\nReversed: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
  return 0;
}
```

Output:

```
Original: 1 2 3 4 5
Reversed: 5 4 3 2 1
```

Logic:

- Swap first with last
- Swap second with second-last
- Continue to middle
- Loop size/2 times

Program 12: Copy Array

```c
#include <stdio.h>
int main() {
  int src[] = {10, 20, 30, 40, 50};
  int dest[5];
  int size = 5;
  int i;
  for (i = 0; i < size; i++) {
    dest[i] = src[i];
  }
  printf("Source: ");
  for (i = 0; i < size; i++) {
    printf("%d ", src[i]);
  }
  printf("\nDestination: ");
  for (i = 0; i < size; i++) {
    printf("%d ", dest[i]);
  }
  printf("\n");
  return 0;
}
```

Output:

```
Source: 10 20 30 40 50
Destination: 10 20 30 40 50
```

Note:

- Cannot do: $\text{dest} = \text{src}$
- Must copy element by element
- Loop through entire array
- Both arrays must exist

Program 13: Shift Elements Left

```c
#include <stdio.h>
int main() {
  int arr[] = {1, 2, 3, 4, 5};
  int size = 5;
  int first, i;
  printf("Original: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  first = arr[0];
  for (i = 0; i < size-1; i++) {
    arr[i] = arr[i+1];
  }
  arr[size-1] = first;
  printf("\nShifted left: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
  return 0;
}
```

Output:

```
Original: 1 2 3 4 5
Shifted left: 2 3 4 5 1
```

Logic:

- Save first element
- Shift all left by one
- Put first at end
- Circular rotation

Program 14: Shift Elements Right

```c
#include <stdio.h>
int main() {
  int arr[] = {1, 2, 3, 4, 5};
  int size = 5;
  int last, i;
  printf("Original: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  last = arr[size-1];
  for (i = size-1; i > 0; i--) {
    arr[i] = arr[i-1];
  }
  arr[0] = last;
  printf("\nShifted right: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
  return 0;
}
```

Output:

```
Original: 1 2 3 4 5
Shifted right: 5 1 2 3 4
```

Logic:

- Save last element
- Shift all right by one
- Put last at beginning
- Reverse of left shift

Program 15: Check if Sorted

```c
#include <stdio.h>
int main() {
  int arr1[] = {1, 2, 3, 4, 5};
  int arr2[] = {1, 3, 2, 4, 5};
  int size = 5;
  int sorted = 1;
  int i;
  printf("Array 1: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr1[i]);
  }
  for (i = 0; i < size-1; i++) {
    if (arr1[i] > arr1[i+1]) {
      sorted = 0;
      break;
    }
  }
  printf("\nSorted: %s\n",
         sorted ? "Yes" : "No");
  return 0;
}
```

Output:

```
Array 1: 1 2 3 4 5
Sorted: Yes
```

Logic:

- Assume sorted = true
- Check consecutive pairs
- If any pair out of order, not sorted
- Break early if found

Program 16: Count Even and Odd

```c
#include <stdio.h>
int main() {
  int arr[] = {12, 7, 18, 5, 9, 14};
  int size = 6;
  int even = 0, odd = 0;
  int i;
  printf("Array: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
    if (arr[i] % 2 == 0) {
      even++;
    } else {
      odd++;
    }
  }
  printf("\n\nEven count: %d\n", even);
  printf("Odd count: %d\n", odd);
  return 0;
}
```

Output:

```
Array: 12 7 18 5 9 14

Even count: 3
Odd count: 3
```

Logic:

- Initialize both counters to 0
- Check each element
- Increment appropriate counter
- Report both counts

Program 17: Second Largest Element

```c
#include <stdio.h>
int main() {
  int arr[] = {34, 12, 89, 5, 67};
  int size = 5;
  int first, second, i;
  first = second = arr[0];
  for (i = 1; i < size; i++) {
    if (arr[i] > first) {
      second = first;
      first = arr[i];
    } else if (arr[i] > second &&
               arr[i] != first) {
      second = arr[i];
    }
  }
  printf("Array: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n\nLargest: %d\n", first);
  printf("Second largest: %d\n",
         second);
  return 0;
}
```

Output:

```
Array: 34 12 89 5 67

Largest: 89
Second largest: 67
```

Logic:

- Track first and second
- If larger than first, update both
- Else if larger than second, update second
- Handle duplicates

Program 18: Compare Two Arrays

```c
#include <stdio.h>
int main() {
  int arr1[] = {1, 2, 3, 4, 5};
  int arr2[] = {1, 2, 3, 4, 5};
  int arr3[] = {1, 2, 9, 4, 5};
  int size = 5;
  int equal = 1;
  int i;
  for (i = 0; i < size; i++) {
    if (arr1[i] != arr2[i]) {
      equal = 0;
      break;
    }
  }
  printf("arr1 vs arr2: %s\n",
         equal ? "Equal" : "Not equal");
  equal = 1;
  for (i = 0; i < size; i++) {
    if (arr1[i] != arr3[i]) {
      equal = 0;
      break;
    }
  }
  printf("arr1 vs arr3: %s\n",
         equal ? "Equal" : "Not equal");
  return 0;
}
```

Output:

```
arr1 vs arr2: Equal
arr1 vs arr3: Not equal
```

Logic:

- Assume equal = true
- Compare element by element
- If any mismatch, not equal
- Can break early

Program 19: Merge Two Arrays

```c
#include <stdio.h>
int main() {
  int arr1[] = {1, 2, 3};
  int arr2[] = {4, 5, 6};
  int merged[6];
  int i;
  for (i = 0; i < 3; i++) {
    merged[i] = arr1[i];
  }
  for (i = 0; i < 3; i++) {
    merged[3+i] = arr2[i];
  }
  printf("Array 1: ");
  for (i = 0; i < 3; i++) {
    printf("%d ", arr1[i]);
  }
  printf("\nArray 2: ");
  for (i = 0; i < 3; i++) {
    printf("%d ", arr2[i]);
  }
  printf("\nMerged: ");
  for (i = 0; i < 6; i++) {
    printf("%d ", merged[i]);
  }
  printf("\n");
  return 0;
}
```

Output:

```
Array 1: 1 2 3
Array 2: 4 5 6
Merged: 1 2 3 4 5 6
```

Logic:

- Create new array of combined size
- Copy first array
- Copy second array after first
- Display all three

Program 20: Find Duplicates

```c
#include <stdio.h>
int main() {
  int arr[] = {1, 2, 3, 2, 4, 3, 5};
  int size = 7;
  int i, j;
  int printed[7] = {0};
  printf("Array: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n\nDuplicates: ");
  for (i = 0; i < size; i++) {
    if (printed[i]) continue;
    for (j = i+1; j < size; j++) {
      if (arr[i] == arr[j] &&
          !printed[i]) {
        printf("%d ", arr[i]);
        printed[i] = 1;
        break;
      }
    }
  }
  printf("\n");
  return 0;
}
```

Output:

```
Array: 1 2 3 2 4 3 5

Duplicates: 2 3
```

Logic:

- Nested loops
- Compare each with rest
- Track printed to avoid duplicates
- Print only once per value

Program 21: Frequency of Each Element

```c
#include <stdio.h>
int main() {
  int arr[] = {1, 2, 1, 3, 2, 1};
  int size = 6;
  int visited[6] = {0};
  int i, j, count;
  printf("Array: ");
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n\nFrequency:\n");
  for (i = 0; i < size; i++) {
    if (visited[i]) continue;
    count = 1;
    for (j = i+1; j < size; j++) {
      if (arr[i] == arr[j]) {
        count++;
        visited[j] = 1;
      }
    }
    printf("%d occurs %d time(s)\n",
           arr[i], count);
  }
  return 0;
}
```

Output:

```
Array: 1 2 1 3 2 1

Frequency:
1 occurs 3 time(s)
2 occurs 2 time(s)
3 occurs 1 time(s)
```

Logic:

- Track visited elements
- Count occurrences
- Mark duplicates as visited
- Print each unique element once

Program 22: Array Index Out of Bounds

```c
#include <stdio.h>
int main() {
  int arr[5] = {10, 20, 30, 40, 50};
  int i;
  printf("Valid access:\n");
  for (i = 0; i < 5; i++) {
    printf("arr[%d] = %d\n", i, arr[i]);
  }
  printf("\nWARNING: Invalid access\n");
  printf("arr[5] would be undefined\n");
  printf("arr[-1] would be undefined\n");
  printf("\nValid indices: 0 to 4\n");
  printf("Array size: 5\n");
  return 0;
}
```

Output:

```
Valid access:
arr[0] = 10
arr[1] = 20
arr[2] = 30
arr[3] = 40
arr[4] = 50

WARNING: Invalid access
arr[5] would be undefined
arr[-1] would be undefined

Valid indices: 0 to 4
Array size: 5
```

Warning:

- C doesn't check bounds
- Out of bounds = undefined behavior

Arrays - Summary

Key Points:

- Collection of same-type elements
- Fixed size, declared at compile time
- Zero-based indexing (0 to size-1)
- Stored in contiguous memory
- Elements accessed by index: `arr[i]`
- Cannot be assigned directly (must copy element-by-element)
- Size calculated: `sizeof(arr)/sizeof(arr[0])`
- Partial initialization fills rest with 0
- Uninitialized arrays have undefined values

Common Operations:

- Traversal, search, sum, max/min, reverse, copy
- Requires loops to process all elements

Best Practices

1. **Always initialize** arrays before use
2. **Check bounds** - valid indices: 0 to size-1
3. **Use constants** for array size (easier to modify)
4. **Pass size** to functions along with array
5. **Use meaningful names** - not just arr, a, b
6. **Validate input** when reading into arrays
7. **Use loops** for array operations
8. **Document assumptions** about array contents
9. **Consider using** `sizeof(arr)/sizeof(arr[0])` for size
10. **Avoid magic numbers** - use named constants

Common Mistakes

1. **Off-by-one errors**: Using `i <= size` instead of `i < size`
2. **Out of bounds access**: Accessing `arr[size]` or negative indices
3. **Uninitialized arrays**: Reading before writing
4. **Wrong size**: Using wrong variable for array size
5. **Direct assignment**: Trying `arr1 = arr2`
6. **Forgetting index 0**: Starting loops at 1
7. **Size confusion**: Forgetting last index is size-1
8. **Not checking empty**: Assuming array has elements
9. **Modifying in loop**: Changing size while iterating
10. **Integer overflow**: Sum of large numbers

Practice Exercises

Try these programs:

1. Insert element at specific position
2. Delete element from specific position
3. Find all pairs that sum to a target
4. Remove duplicates from array
5. Rotate array by k positions
6. Find missing number in sequence 1 to n
7. Move all zeros to end
8. Check if array is palindrome
9. Find intersection of two arrays
10. Separate even and odd numbers