

UNIT 7

Array

(6 Hours) — C Programming (CACS151)

7.1 Introduction

An array is a derived data type that stores a fixed-size, ordered collection of elements, all of the same data type, in contiguous memory locations. Instead of declaring many separate variables to hold related values (e.g. marks of 50 students), a single array variable can hold all of them, accessed conveniently through a common name and a numeric index.

Arrays are one of the most fundamental data structures in C and form the basis for implementing lists, tables, matrices, strings, and many of the sorting and searching algorithms covered later in this unit.

Why Arrays Matter

- ✓ Group related data of the same type under a single variable name
- ✓ Allow efficient, direct access to any element using its index
- ✓ Provide the foundation for strings, matrices, and many algorithms
- ✓ Simplify code that would otherwise need many separate variables

7.2 Declaration

An array must be declared before use, specifying its element data type, its name, and its size (the number of elements it can hold), enclosed in square brackets.

Example — Array Declarations

```
int marks[50]; /* array of 50 integers */  
float price[10]; /* array of 10 floats */  
char name[20]; /* array of 20 characters (a string) */  
int matrix[3][4]; /* 2-D array: 3 rows, 4 columns */
```

Note: The size specified in the declaration must be a positive integer constant, and it is fixed for the lifetime of the array — C does not support resizing a regular array once declared.

7.3 Initialization

An array can be given initial values at the point of declaration. If fewer values are supplied than the declared size, the remaining elements are automatically set to zero. If the size is omitted entirely, the compiler determines it from the number of values provided.

Form	Example	Result
Full initialization	<code>int a[5] = {1,2,3,4,5};</code>	All 5 elements set as given
Partial initialization	<code>int a[5] = {1,2};</code>	<code>a[0]=1, a[1]=2, rest = 0</code>
Size inferred	<code>int a[] = {1,2,3};</code>	Compiler sets size to 3
String initialization	<code>char s[] = "Hi";</code>	Stores H, i, and <code>\0</code> automatically

7.4 One Dimensional Array

A one-dimensional (1-D) array stores a simple linear list of elements, all accessed using a single index. In C, indices always start at 0, so an array of size *n* has valid indices from 0 to *n*-1.

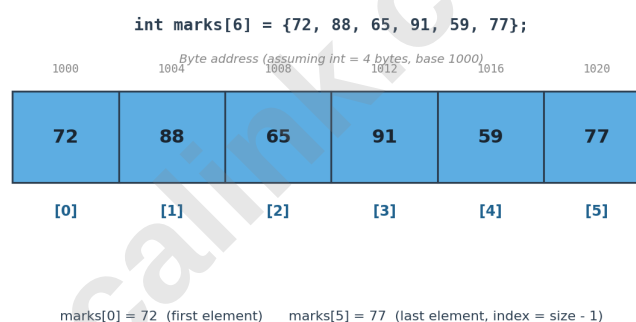


Figure 7.1 — Memory layout of a 1-D integer array with indices and addresses

Example — Reading and Printing a 1-D Array

```

int marks[5];
for (int i = 0; i < 5; i++)
    scanf("%d", &marks[i]);
for (int i = 0; i < 5; i++)
    printf("%d ", marks[i]);

```

*Note: Because array elements sit in contiguous memory, the address of element *i* can be computed as: $\text{base_address} + i * \text{sizeof}(\text{element_type})$.*

7.5 Multi Dimensional Array

A multi-dimensional array extends the idea of a 1-D array to two or more indices, most commonly used to represent tables or matrices. A two-dimensional (2-D) array is declared with two sizes: the number of rows and the number of columns.

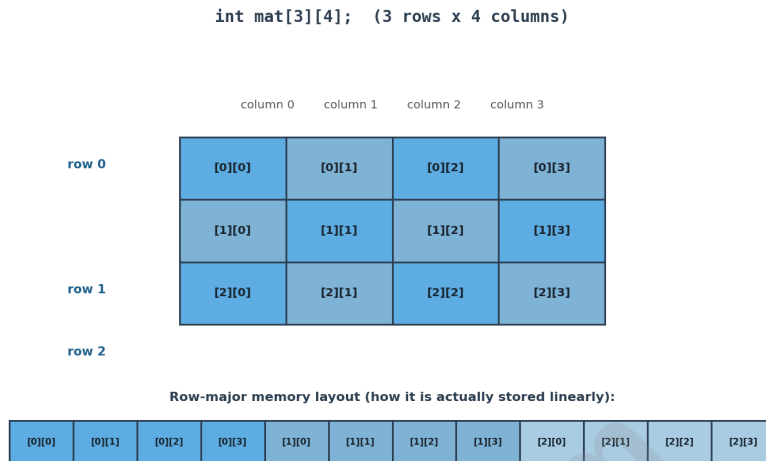


Figure 7.2 — Logical layout of a 2-D array and its actual row-major storage in memory

Example — 2-D Array Declaration and Access

```
int mat[3][4];
mat[1][2] = 25; /* row index 1, column index 2 */
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 4; j++)
        scanf("%d", &mat[i][j]);
```

Note: C stores multi-dimensional arrays in row-major order — all elements of row 0 are stored first, followed by all elements of row 1, and so on.

7.6 Sorting

Sorting is the process of rearranging the elements of an array into a specific order — usually ascending or descending. Two simple, widely taught sorting techniques are Bubble Sort and Selection Sort.

7.6.1 Bubble Sort

Bubble sort repeatedly steps through the array, compares each pair of adjacent elements, and swaps them if they are in the wrong order. With each complete pass, the largest remaining unsorted element “bubbles up” to its correct position at the end of the array.

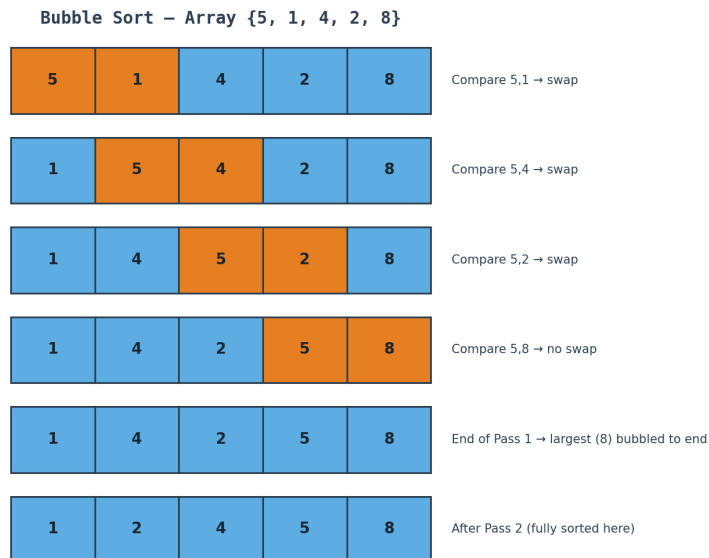


Figure 7.3 — Bubble sort passes on the array {5, 1, 4, 2, 8}

Bubble Sort — C Function

```
void bubbleSort(int a[], int n) {
    for (int i = 0; i < n-1; i++)
        for (int j = 0; j < n-1-i; j++)
            if (a[j] > a[j+1]) {
                int t = a[j]; a[j] = a[j+1]; a[j+1] = t;
            }
}
```

7.6.2 Selection Sort

Selection sort divides the array into a sorted and an unsorted part. On each pass, it selects the smallest element from the unsorted part and swaps it into the next position of the sorted part, gradually growing the sorted section from left to right.

Selection Sort – Array {29, 10, 14, 37, 13}

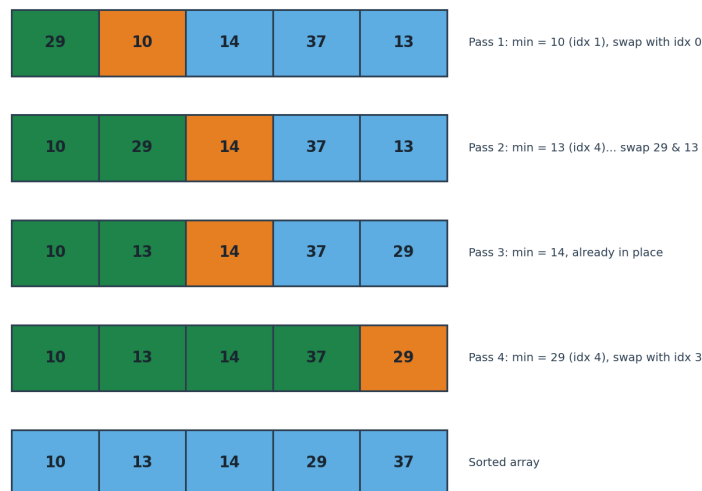


Figure 7.4 — Selection sort passes on the array {29, 10, 14, 37, 13}

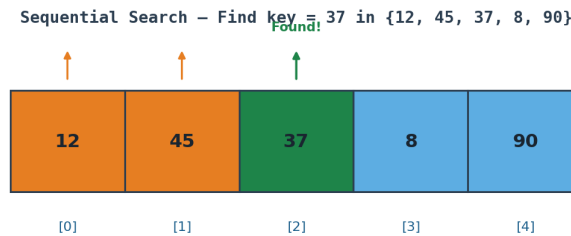
Selection Sort — C Function

```
void selectionSort(int a[], int n) {
    for (int i = 0; i < n-1; i++) {
        int minIdx = i;
        for (int j = i+1; j < n; j++)
            if (a[j] < a[minIdx]) minIdx = j;
        int t = a[i]; a[i] = a[minIdx]; a[minIdx] = t;
    }
}
```

Aspect	Bubble Sort	Selection Sort
Basic idea	Repeatedly swap adjacent out-of-order pairs	Repeatedly select the minimum and place it
Number of swaps	Can be many (one per out-of-order pair)	At most one swap per pass
Best case	$O(n)$ if already sorted (with early-exit flag)	$O(n^2)$ — always scans remaining elements
Typical use	Teaching simplicity, small data sets	Teaching simplicity, when swaps are costly

7.7 Searching (Sequential)

Searching is the process of locating a specific value (the key) within an array. Sequential search (also called linear search) is the simplest method: it examines each element one by one, starting from the first, until the key is found or the entire array has been checked.



Each element is checked one by one, starting at index 0, until the key is found or the array ends (index 0 → 1 → 2 = match).

Figure 7.5 — Sequential search for key = 37 in {12, 45, 37, 8, 90}

Sequential Search — C Function

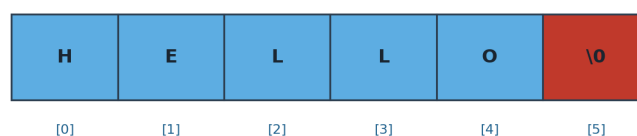
```
int seqSearch(int a[], int n, int key) {
    for (int i = 0; i < n; i++)
        if (a[i] == key)
            return i; /* found at index i */
    return -1; /* not found */
}
```

Note: Sequential search does not require the array to be sorted, but its worst-case performance is $O(n)$, since it may need to check every single element.

7.8 String Handling

In C, a string is simply an array of characters terminated by a special null character ('`\0`'), which marks the end of the actual text. There is no separate built-in string data type — strings are always represented and manipulated as character arrays.

```
char name[6] = "HELLO";
```



The compiler automatically appends a null character ('`\0`') to mark the end of the string — without it, string functions cannot know where it stops.

Figure 7.6 — A C string stored as a character array with a null terminator

The standard library header provides several ready-made functions for common string operations:

Function	Purpose	Example
<code>strlen(s)</code>	Returns the length of string <code>s</code> (excluding <code>\0</code>)	<code>strlen("Hello") → 5</code>
<code>strcpy(dest, src)</code>	Copies <code>src</code> into <code>dest</code>	<code>strcpy(a, "Hi")</code>
<code>strcat(dest, src)</code>	Appends <code>src</code> to the end of <code>dest</code>	<code>strcat(a, "!")</code>
<code>strcmp(s1, s2)</code>	Compares <code>s1</code> and <code>s2</code> ; returns 0 if equal	<code>strcmp("cat", "cat") → 0</code>
<code>strrev(s)</code>	Reverses the string <code>s</code> (non-standard, common on some compilers)	<code>strrev("abc") → "cba"</code>

Example — Basic String Handling

```
char name[20] = "Nepal";  
printf("%d", strlen(name)); /* 5 */  
strcat(name, " Rocks");  
printf("%s", name); /* Nepal Rocks */
```

Key Takeaways — Unit 7

- ✓ An array groups multiple values of the same type under one name, indexed from 0
- ✓ Declaration fixes the element type and size; initialization can set values immediately, with unset elements defaulting to 0
- ✓ A 1-D array is a simple indexed list; a 2-D array adds rows and columns, stored in row-major order
- ✓ Bubble sort repeatedly swaps adjacent out-of-order elements; selection sort repeatedly picks the minimum remaining element
- ✓ Sequential search checks elements one at a time and works on both sorted and unsorted arrays
- ✓ A C string is a character array ending in a null terminator (`\0`), manipulated using functions

— End of Unit 7 —