Arrays and Strings in C

Md. Aminul Islam Shazid

Outline

Introduction to Arrays

What is an Array?

- An **array** is a collection of elements of the *same* type stored in contiguous memory.
- Each element is accessed by an **index** (zero-based in C, meaning that the first element is at index 0): `arr[0]`, `arr[1]`, ...
- Real-life analogy: a row of mailboxes, a list of student marks, or seats in a bus lined up.
- Arrays let us group related values under one name.

1D Array: Declaration and Initialization

Declaration:

```
int arr[5];              // declares an array of 5 integers
```

Declaration and Initialization:

```
int a[5] = {10, 20, 30, 40, 50};
int b[]  = {1, 2, 3};    // size inferred: 3
```

Accessing Individual Elements

- Use square brackets with index: `arr[index]`.
- Example: `arr[2]` accesses the third element.

Example: print elements

```c
#include <stdio.h>

int main() {
    int a[5] = {10, 20, 30, 40, 50};
    printf("%d\n", a[2]);    // prints 30
    return 0;
}
```

Sample Output:

```
30
```

Take User Input Into an Array

```c
#include <stdio.h>

int main() {
    int n, i;
    printf("How many numbers? ");
    scanf("%d", &n);
    int arr[100];    // assume max 100 for simplicity
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    // print them
    for (i = 0; i < n; i++) {
        printf("arr[%d] = %d\n", i, arr[i]);
    }
    return 0;
```

Summing the Elements of an Array

```c
#include <stdio.h>

int main() {
    int n, i, sum = 0;
    printf("n: ");
    scanf("%d", &n);
    int arr[100];
    for (i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }

    for (i = 0; i < n; i++){
        sum += arr[i];
    }
    printf("Sum = %d\n", sum);
```

Multidimensional Arrays

What is a Multidimensional Array?

- A **2D array** is an array of arrays (like a table / matrix).
- Indices are written as `arr[row][col]`.
- Real-life analogies:
    - 2D: matricx or spreadsheet (rows and columns)
    - 3D: a stack of matrices
- C supports arrays with any number of dimensions; common ones: 1D, 2D and 3D

2D Array: Declaration and Initialization

```
int mat[3][4];                     // 3 rows, 4 columns

int mat2[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};

// or flattened initialization:
int mat3[2][3] = {1,2,3,4,5,6};
```

Accessing Values in 2D Arrays

- Access element at row r, column c by `mat[r][c]`.
- Example: `mat[1][2]` refers to second row, third column.

Example: print a 2×3 matrix

```c
#include <stdio.h>

int main() {
    int mat[2][3] = {{1,2,3},{4,5,6}};
    printf("%d\n", mat[1][2]); // prints 6
    return 0;
}
```

Sample Output:

6

Input and Print a 2D Array

```c
#include <stdio.h>

int main() {
    int r = 2, c = 3, i, j, mat[2][3];
    printf("Please input a 2 by 3 matrix:\n");
    for (i = 0; i < r; i++)
        for (j = 0; j < c; j++)
            scanf("%d", &mat[i][j]);

    printf("\nYou entered:\n");
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++)
            printf("%d ", mat[i][j]);
        printf("\n");
    }
```

3D and Higher Dimensions

- A 3D array `int a[2][3][4];` can be thought of as 2 blocks, each block is a 3×4 matrix
- Real-life: for example, **day × row × column** measurements (temperature map over multiple days)
- Indexing: `a[day][row][col]`

Strings in C

What is a String in C?

- In C, a **string** is an array of `char` terminated by the null character `'\0'`.
- Example: `char s[] = "hello";` actually creates 6 chars: `'h','e','l','l','o','\0'`.
- Strings are manipulated through arrays and standard library functions in `<string.h>`.
- You can access individual characters with `s[i]`.

Declare and Initialize Strings

```c
char s1[] = "Hello";
char s2[10] = "Hi";     // remaining bytes unused (but availabl
char s3[6] = {'H','i','!','\0'}; // explicit
```

Reading Strings from User

- Avoid gets() (unsafe). Use fgets() or scanf("%s", ...).
- scanf("%s", s); reads until whitespace — does not read spaces.
- fgets(s, size, stdin); reads a whole line (including spaces), but includes newline — may want to trim it.

Example using fgets

```
#include <stdio.h>
#include <string.h>

int main() {
    char s[100];
    printf("Enter a line: ");
    fgets(s, sizeof(s), stdin);
    // remove trailing newline
    s[strcspn(s, "\n")] = '\0';
    printf("You wrote: \%s\n", s);
```

Common String Functions (from <string.h>)

- `strlen(s)` — length of string (not counting '\0')
- `strcmp(s1, s2)` — compare strings (0 if equal)
- `strcpy(dest, src)` — copy string
- `strcat(dest, src)` — concatenate

Example: strcmp() and strcpy()

```c
#include <stdio.h>
#include <string.h>

int main() {
    char a[20], b[20];
    strcpy(a, "apple");
    strcpy(b, "apple");
    if (strcmp(a, b) == 0)
        printf("Equal\n");
    else
        printf("Not equal\n");
    return 0;
}
```

Sample Output

Upper-case and Lower-case Conversion (Manual)

```c
#include <stdio.h>
#include <ctype.h>    // for toupper, tolower

int main() {
    char s[] = "Hello World!";
    int i = 0;
    // to upper
    while (s[i]) {
        s[i] = toupper((unsigned char)s[i]);
        i++;
    }
    printf("\%s\n", s);  // HELLO WORLD!
    return 0;
}
```

Example: Concatenate Strings (strcat)

```c
#include <stdio.h>
#include <string.h>

int main() {
    char a[50] = "Hello";
    char b[] = " World";
    strcat(a, b);
    printf("\%s\n", a); // prints "Hello World"
    return 0;
}
```

Sample Output:

```
Hello World
```

Examples

Example: Reverse an Array

```c
#include <stdio.h>

int main() {
    int n = 5, i;
    int a[] = {1,2,3,4,5};
    for (i = 0; i < n/2; i++) {
        int tmp = a[i];
        a[i] = a[n-1-i];
        a[n-1-i] = tmp;
    }
    for (i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");
    return 0;
}
```

Example: Count Vowels in a String

```c
#include <stdio.h>
#include <ctype.h>

int main() {
    char s[100];
    fgets(s, sizeof(s), stdin);
    s[strcspn(s, "\n")] = '\0';
    int i = 0, count = 0;
    while (s[i]) {
        char ch = tolower((unsigned char)s[i]);
        if (ch=='a'||ch=='e'||ch=='i'||ch=='o'||ch=='u'){
            count++;
        }
        i++;
    }
```

Summary

Summary

- **Array:** contiguous collection of same-type elements, accessed by indices `arr[i]`.
- **1D/2D/3D:** use `arr[i]`, `arr[i][j]`, `arr[i][j][k]` respectively.
- **Strings:** arrays of `char` ending with `'\0'`. Use `<string.h>` functions for convenience.
- **Input:** `scanf` or `fgets` (preferred for whole lines).
- **Memory:** array name often decays to pointer to the first element — but arrays are fixed-sized storage

Array Variable and Memory

- **Important note:** In many contexts the array name (e.g., `arr`) *decays* to a pointer (details on pointers in upcoming lectures) to the first element. Example: when passed to a function
- But `arr` itself is not a regular variable containing a value — you cannot reassign it (e.g., `arr = someOtherPointer;` is invalid)
- Internally arrays point to contiguous chunk of memory starting at the first element

Exercises

Exercises

- Write a program to read `n` integers into an array and print them in reverse order.
- Write a program to find the maximum and minimum values in an integer array.
- Write a program to remove duplicate elements from a small integer array (keep first occurrences).
- Write a program to multiply two 2×2 matrices and print the result.
- Write a program to read a line of text and print its length (without using `strlen`).

Exercises (cont.)

- Write a program to check if a given string is a palindrome (ignore case and spaces).
- Write a program to count frequency of each digit (0-9) in an array of integers.
- Write a program to concatenate two strings without using `strcat`.
- Write a program to rotate the elements of an array to the right by `k` positions.
- Write a program to read a 3D array of size 2 × 2 × 2 and compute the sum of all elements.