

Arrays and Strings in C

Md. Aminul Islam Shazid

Outline

- 1 Introduction to Arrays
- 2 Multidimensional Arrays
- 3 Strings in C
- 4 Examples
- 5 Summary
- 6 Exercises

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]`, ...
- Example: list of student marks
- 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

```
1 #include <stdio.h>
2
3 int main() {
4     int a[5] = {10, 20, 30, 40, 50};
5     printf("%d\n", a[2]);    // prints 30
6     return 0;
7 }
```

Sample Output:

30

Take User Input Into an Array

```
1  #include <stdio.h>
2
3  int main() {
4      int n, i;
5      printf("How many numbers? ");
6      scanf("%d", &n);
7      int arr[100];    // assume max 100 for simplicity
8      for (i = 0; i < n; i++) {
9          scanf("%d", &arr[i]);
10     }
11     for (i = 0; i < n; i++) {    // printing the values
12         printf("arr[%d] = %d\n", i, arr[i]);
13     }
14 }
```

Summing the Elements of an Array

```
1  #include <stdio.h>
2
3  int main() {
4      int i, sum = 0, arr[5] = {10, 20, 30, 40, 50};
5
6      for (i = 0; i < 5; i++){
7          sum += arr[i];
8      }
9      printf("Sum = %d\n", sum);
10 }
```


Multidimensional Arrays

Multidimensional Array

- An array with more than one dimension
- Real-life analogies:
 - 2D: matrix or spreadsheet (rows and columns), indices are written as `arr[row][col]`
 - 3D: a stack of matrices
- C supports arrays with any number of dimensions

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

```
1  #include <stdio.h>
2
3  int main(){
4      int mat[2][3] = {{1,2,3},{4,5,6}};
5      printf("%d\n", mat[1][2]); // prints 6
6      return 0;
7  }
```

Input and Print a 2D Array

```
1  #include <stdio.h>
2
3  int main() {
4      int r = 2, c = 3, i, j, mat[2][3];
5
6      printf("Please input a 2 by 3 matrix:\n");
7
8      for (i = 0; i < r; i++){
9          for (j = 0; j < c; j++){
10             scanf("%d", &mat[i][j]);
11         }
12     }
13
14     printf("\nYou entered:\n");
```

Continued in next slide

Input and Print a 2D Array (cont.)

```
14     printf("\nYou entered:\n");
15
16     for (i = 0; i < r; i++) {
17         for (j = 0; j < c; j++){
18             printf("%d ", mat[i][j]);
19         }
20         printf("\n");
21     }
22 }
```

3D and Higher Dimensions

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

```
char s1[] = "Hello";  
char s2[10] = "Hi";      // remaining bytes unused  
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)
- However, the string from `fgets()` includes newline (`\n`)
- May want to trim this newline (trimming it, is often not required, depends on use case)

String Input Using `scanf()`

```
1  #include <stdio.h>
2
3  int main(){
4      char username[20];
5      printf("Enter your name: ");
6      scanf("%s", username);
7      printf("Good day to you, %s!", username);
8  }
```

Note that in `scanf()`, we provided `username` as the second argument and not `&username`.

This is because `username` itself holds the memory address of the character array.

Recall that the `&var` returns the memory address of the variable named `var`.

String Input Using fgets()

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main() {
5      char s[100];
6      printf("Enter a line: ");
7      fgets(s, sizeof(s), stdin);
8
9      // remove trailing newline
10     // search for "\n" and replace with "\0"
11     s[strcspn(s, "\n")] = '\0';
12     printf("You wrote: %s\n", s);
13     return 0;
14 }
```

String Input Using `fgets()` (cont.)

```
9      // remove trailing newline
10     // search for "\n" and replace with "\0"
11     s[strcspn(s, "\n")] = '\0';
```

In the above code, `strcspn(s, "\n")` searches for and returns the index of the newline character ("`\n`") in the string named `s`.

`s[strcspn(s, "\n")] = '\0';` replaces the newline character with the null terminator character ("`\0`").

Common String Functions (from <string.h>)

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

Example: strcmp() and strcpy()

```
#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;
}
```


Upper-case and Lower-case Conversion (Manual)

```
#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)

```
#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

```
#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

```
#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 $2 \times 2 \times 2$ and compute the sum of all elements.