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]`, ...
- 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

```c
#include <stdio.h>

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

Example: Take User Input Into an Array

```c
#include <stdio.h>

int main() {
    int n, i;
    printf("How many numbers? ");
    scanf("%d", &n);
    int arr[n];    // define an integer array of size n
    for(i = 0; i < n; i++){
        scanf("%d", &arr[i]);
    }
    for(i = 0; i < n; i++){    // printing the values
        printf("arr[%d] = %d\n", i, arr[i]);
    }
}
```

Example: Summing the Elements of an Array

```c
#include <stdio.h>

int main() {
    int i, sum = 0, arr[5] = {10, 20, 30, 40, 50};

    for(i = 0; i < 5; i++){
        sum += arr[i];
    }
    printf("Sum = %d\n", sum);
}
```

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

```c
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 2x3 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
}
```

Example: 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");
```

Continued in next slide

Example: Input and Print a 2D Array (cont.)

```c
    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 arr[2][3][4];` can be thought of as 2 blocks, each block is a 3×4 matrix
- Real-life: for example, **block × row × 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

```c
char c1 = 'a';    // a single character

char s1[] = "Hello";
char s2[10] = "Hi";    // remaining bytes unused

char s3[6] = {'H','i','!','\0'};    // explicit
```

Note: individual characters denoted by single quotes, whereas strings are denoted by double quotes.

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)

Example: String Input Using `scanf()`

```c
#include <stdio.h>

int main(){
    char username[20];
    printf("Enter your name: ");
    scanf("%s", username);
    printf("Good day to you, %s!", username);
}
```

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`.

Example: String Input Using `fgets()`

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

int main() {
    char s[100];
    printf("Enter a line: ");

    fgets(s, sizeof(s), stdin);

    printf("You wrote: \%s", s);
}
```

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

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

int main() {
    char s[100];
    printf("Enter a line: ");
    fgets(s, sizeof(s), stdin);

    // remove trailing newline
    // search for "\n" and replace with '\0'
    s[strcspn(s, "\n")] = '\0';
    printf("You wrote: %s", s);
}
```

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

```c
    // remove trailing newline
    // search for "\n" and replace with '\0'
    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: Comparing Two Strings: `strcmp()`

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

int main() {
    char a[20], b[20];
    scanf("%s", a);
    scanf("%s", b);
    if (strcmp(a, b) == 0)
        printf("Same");
    else
        printf("Not same");
}
```

`strcmp()` returns 0 if the two strings are same.

Example: Copying a String: `strcpy()`

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

int main() {
    char a[20], b[20];
    scanf("%s", a);

    // copying the contents of a into b:
    strcpy(b, a);    // 2nd argument is the source
    printf("%s", b);
}
```

Example: Concatenate Strings (`strcat()`)

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

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

Example: Upper-case and Lower-case Conversion (Manual)

```c
#include <stdio.h>
#include <string.h>    // for strlen()
#include <ctype.h>    // for toupper(), tolower()

int main(){
    char s[] = "Hello World!";
    for(int i = 0; i < strlen(s); i++){
        s[i] = toupper((unsigned char)s[i]);
    }
    printf("%s", s);    // HELLO WORLD!
}
```

Example: Count Vowels in a String

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main(){
    char s[100];
    fgets(s, sizeof(s), stdin);
    int count = 0;
    for(int i = 0; i < strlen(s); i++){
        char ch = tolower((unsigned char)s[i]);
        if (ch=='a'||ch=='e'||ch=='i'||ch=='o'||ch=='u'){
            count++;
        }
    }
    printf("Vowels = %d", count);
}
```

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'`
- `<string.h>` contains functions that operate on strings
- **Input:** `scanf` or `fgets` (preferred for whole lines)

Array Variable and Memory

- **Important note:** In many contexts (for example, when passing to a function), the array name (say, `arr`) is a pointer to the first element (details on pointers in upcoming lectures)
- Meaning that, `arr` actually holds the memory address in which the first element of the array is stored
- To summarize: the array name points to a contiguous block of memory starting at the first element of the array

Exercises

Exercises

1. Write a program to read `n` integers into an array and print them in reverse order
2. Write a program to find the maximum and minimum values in an integer array
3. Write a program to remove duplicate elements from a small integer array (keep first occurrences)
4. Write a program to count frequency of each digit (0 - 9) in an array of integers
5. Write a program to multiply two 2×2 matrices and print the result

Exercises (cont.)

6. Write a program to rotate the elements of an array to the right by `k` positions
7. Write a program to read a line of text and print its length (without using `strlen`)
8. Write a program to concatenate two strings without using `strcat`
9. Write a program to check if a given string is a palindrome (ignore case and spaces)