Arrays in C

By: Yehia M. Abu Eita

Outlines

- Introduction
- 1-D arrays
- 2-D arrays
- Strings
- String functions

Introduction

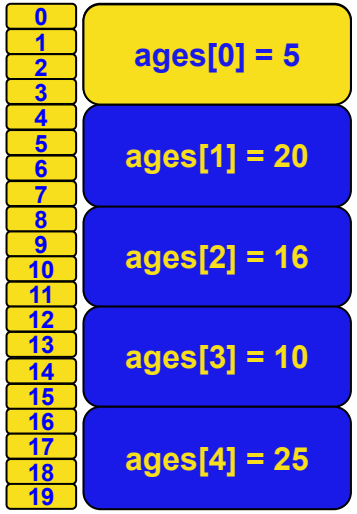
- Array is a non-primitive data type that can **hold multiple** elements of the **same data type**.
- Arrays are used to **group** the **same** type of data together, ex. array of ages.
- Array **size/number of elements** must be defined before compiling.
- Array **size can not be changed** during the **execution**.
- Each array element has its **unique index starting from 0**.

1-D arrays

- **Size/Number of elements:** must be defined **before compiling**.
- **Declaration example:**
  - **int ages[5]; // ages is an array of 5 integers.**
- **Definition example:**
  - **int ages[5] = {15, 20, 16, 10, 25};**
  - **Element with index 0 has value of 15.**
  - **Element with index 4 has value of 25.**
- Note that **THERE IS NO** element with **index 5 or more** !!!!

1-D arrays

- For **set/get** the **value**:
  - array_name[element_index] = value;
- For **getting** the **address**:
  - &array_name[element_index].
- Examples:
  - ages[0] = 5; // changes age[0] value to 5
  - printf("%d", ages[3]); // prints 10
  - scanf("%d", &ages[2]); // takes input from keyboard and stores it in ages[2]

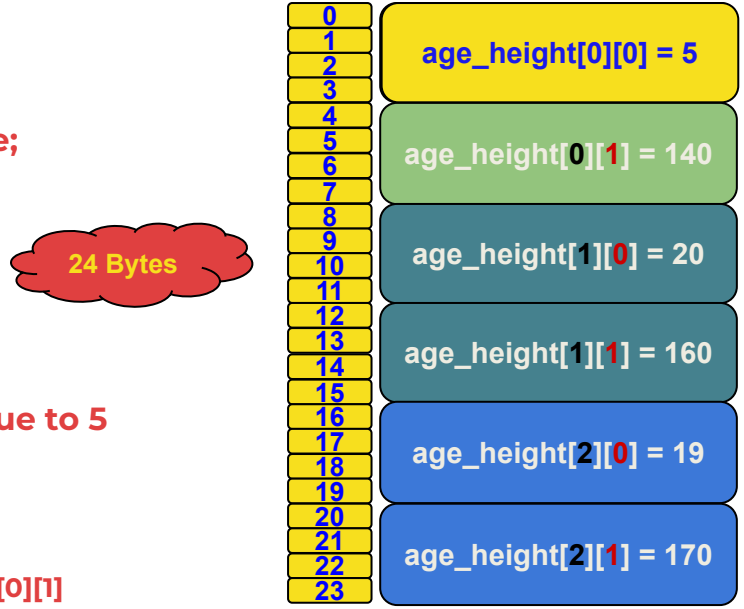


2-D arrays

- **Size/Number of elements:** must be defined **before compiling**.
- **Declaration example:**
  - **int age_height[3][2]; // ages is an array of 3 sub-arrays of 2 integers.**
- **Definition example:**
  - **int age_height[3][2] = {{15, 140}, {20, 160}, {19, 170}};**
  - **Element with index 0 in the sub-array with index 0 has value of 15.**
  - **Element with index 1 in the sub-array with index 2 has value of 170.**
- Note that **THERE IS NO element** with **index 3 or more**, and **THERE IS NO sub-array** with **index 4 or more** !!!!

2-D arrays

- For **set/get** the **value**:
  - array_name[sub_array_index][element_index] = value;
- For **getting** the **address**:
  - &array_name[sub_array_index][element_index].
- Examples:
  - age_height[0][0] = 5; // changes age_height[0][0] value to 5
  - printf("%d", age_height[1][1]); // prints 160
  - scanf("%d", &age_height[0][1]);
    - // takes input from keyboard and stores it in age_height[0][1]



Strings

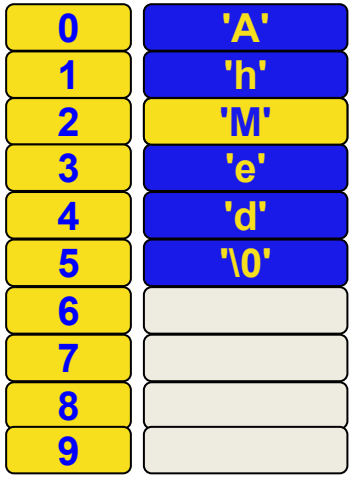
- Strings in C language are simply defined as an **array of ASCII characters terminated by a NULL** character **'\0' or 0**.
- **ASCII code**:
  - **Stands for: A**merican **S**tandard **C**ode for **I**nformation **I**nterchange.
  - It is a code that represents all characters you can type.
  - 'A', 'B', 'C',.....'a', 'b', 'c', ..., '\n', '?', ' ', '+', ..., '1', '2', etc.
- ASCII characters range from **ASCII code 0** to **ASCII code 127**.
- Each character's ASCII can be written in C by writing the character between two single quotes, ex. **ASCII of A is 'A' or 65 in decimal**.

Strings

- **Size/Number of elements:** must be defined **before compiling**.
- **Declaration example:**
  - **char name[10]; // name is an array of 10 characters.**
- **Definition example:**
  - **char name[10] = {'A', 'h', 'm', 'e', 'd', '\0'};** or simply **char name[10] = "Ahmed";**
  - The string name can hold up to **9 characters + the NULL character.**
  - If you entered 10 characters, then the array name will not be treated as a string.
  - If you entered **more than 10** characters, the **application may crash.**

Strings

- For **set/get** the **value**:
  - **array_name[character_index] = 'value';**
- For getting the address:
  - **&array_name[character_index].**
- Examples:
  - **name[2] = 'M'; // changes name[2] value to 'M'**
  - **Printing:**
    - **printf("%s", name); // prints Ahmed**
    - **puts(name); // prints Ahmed**
  - **Scanning:**
    - **gets(name); //takes all the string from the keyboard and stores it in the string array name with the same order**



String functions

- **stdio.h** functions:
  - **puts:**
    - **Operation:** It prints the string.
    - **Syntax:** puts(srting_name);
    - **Example:** puts("Ahmed"); // prints Ahmed
  - **gets:**
    - **Operation:** It reads the input from the keyboard and stores it in the string in the same order.
    - **Syntax:** gets(string_name);
    - **Example:** gets(name);

String functions

- **string.h** functions:
  - **strlen:**
    - **Operation:** It gets the string length, the number of characters in the string without the NULL character.
    - **Syntax:** strlen(srting_name/string);
    - **Example:** int length = strlen("Ahmed"); // length of Ahmed is 5.
  - **strupr:**
    - **Operation:** It converts all characters in the string to uppercase.
    - **Syntax:** strupr(string_name);
    - **Example:** strupr(name); // Converts Ahmed to AHMED.
  - **strlwr:**
    - **Operation:** It converts all characters in the string to lowercase.
    - **Syntax:** strlwr(string_name);
    - **Example:** strlwr(name); // Converts Ahmed to ahmed.

String functions

- **string.h functions:**
  - **strcmp:**
    - **Operation:** It compares two strings if they are identical it returns 0.
    - **Syntax:** strcmp(string_1_name, string_2_name);
    - **Example:** strcmp("Ahmed", "ahmed"); // It returns a non-zero value because 'A' ≠ 'a'.
  - **strcat:**
    - **Operation:** It concatenates two strings.
    - **Syntax:** strcat(string_1_name, string_2_name);
    - **Example:** strcat(name, " Mohamed"); // name will become "Ahmed Mohamed".
  - **strcpy:**
    - **Operation:** It copies a string to another.
    - **Syntax:** strcpy(destinamtion_string, source_string);
    - **Example:** strcpy(name, "Zaki"); // name will become "Zaki".

Summary

- Now you are familiar with array declaration, definition, and accessing array elements.
- Remember, array size is defined before compiling.
- Remember, array size can not be changed during run-time.
- Remember, the last element has index of array_size - 1.
- Remember, strings must be terminated by ‘\0’.