Module 29: C-Strings

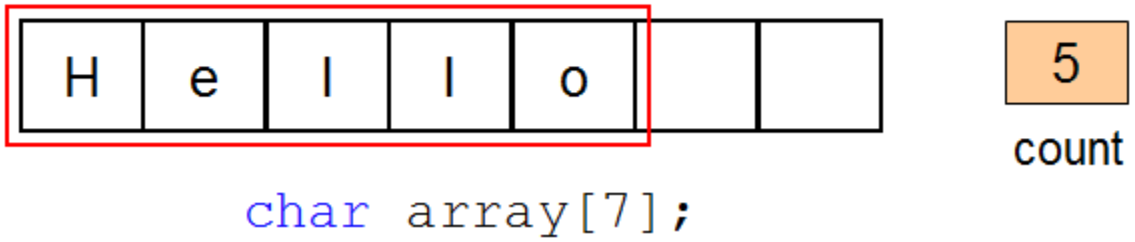
Intro to Computer Science 1 - C++
Professor Scott Frees

Textbook

C-Strings are covered in section 7.11

Array of Characters

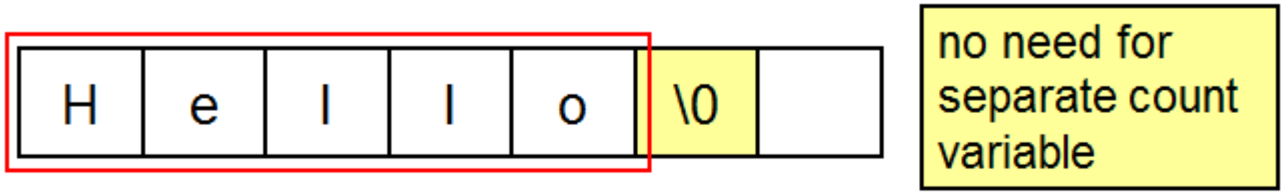
- string is a series of characters (char)
- A string ***literal*** is enclosed in double quotes
    - ...what about string ***variables***?
    - There is no native data type for strings...



However - A string can naturally be represented by an array of characters.

C-Strings

- In a partially filled array, we normally need to keep track of how many elements we have. (count)
  - Must always pass array and count to functions
  - Strings are so common, they have their own special "convention":
  - Keep track of the number of "elements" by using a **sentinel**
    - `'\0'` is called the ***null*** character...
    - `'\0'` is always placed in the last position used in the array



Reading c-strings from user

- We can read a string from the user by calling `cin.getline(…)`
- `cin.getline` takes two parameters:
  1. Array to store the string
  2. Maximum size of the string

`cin.getline` automatically places a **`‘\0’`** character at the end of the string

C-String output

Printing an array of numbers was easy:

```
for ( int i = 0; i < count; i++ ) {
  cout << num[i] << endl;
}
```

For C-Strings, we don't have a count variable, so we might need a while loop

- We don't know how many characters are in the string!

C-String output

As long as our character array has a ‘\0’ character at the end of the string, cout understands how to print it automatically:

```
char input[25];
cin.getline(input, 25);
cout << "You entered:  " << input    << endl;
```

Programming Exercise 33

1. Read 2 c-strings from the user
2. Print out whether or not the strings are equal

- Note: == doesn’t work with arrays (of any kind!)
- Check if same length
- Check if same contents

cstring library

These functions are already provided to us inside <cstring>

```
int strlen(char str[] )  // returns the length
int strcmp(char str1[], char str2[] )
// returns -1 if str1 before str2 in dictionary
// returns 1 if str2 before str1 in dictionary
// returns 0 if they are equal
```

Lets replace our code from the last programming example with the cstring library's implementation

Character Functions

Many function deal with individual characters:

Found in <cctype>

`int isupper(char c)` *returns 1 if c is upper case*

`int islower(char c)` *returns 1 if c is lower case*

`int isalpha(char c)` *returns 1 if c is a-z, A-Z*

`int isspace(char c)` *returns 1 if c is white space*

`char toupper(char c)` *returns uppercase version of c*

`char tolower(char c)` *returns lowercase version of c*

Programming Exercise 34

Read a c-string from user

Determine if it is a *palindrome* - meaning it reads the same backwards and forwards.

- Changes in case (upper/lower) should not prevent it from being a palindrome: *abA is a palindrome*
- Spaces should not count towards the check: *a ba is a palindrome*

Lab 11

Write a program that reads two strings from the user

Determine if the two strings (A & B) are *anagrams*

Anagrams are words that contain the same letters, in any order

Lab 11

Write a function (**countChars**) that accepts a c-string and a character as parameters. Returns the number of times the character appears in the string

Check for anagram using the following steps

Assume you have two strings – A and B

1. If A is not the same length as B, not anagrams
2. Convert A and B to all lower case.
3. For **each** letter in A, call your countChar on A and B. If the results do not match, stop

If each letter in A returns the same number in B, then they are anagrams.