Wesleyan University, Fall 2022, COMP 211

Lab 3: Functions and Arrays

---

1. Overview

In today's lab we will look at how to work with functions and arrays, including passing an array to a function. You will first modify your leap year program to perform the leap year check in a function. Then you'll write a program to read in a user's first name and then their last name, and store the full name in a single character array. For this lab, you should create a new directory lab3 in your labs directory to hold your code for this lab.

I will begin the lab by walking through some sample code (posted as `lab3 example.c`) on functions, arrays, and strings and then leave you to implement the `leapfunc.c` and `name.c` programs.

2. A leap year program that uses a function: `leapfunc.c`

In the leap year program you wrote in Lab 2, you may or may not have had a chance to put the computation for the leap year check in a function. Typically, code such as a check like this belongs in a function: doing so makes the code cleaner and easier to understand. Thus, if you haven't put your leap year check code in a function please do so now and then call your function to perform the leap year check.

3. A name entry and echo program: `name.c`

Your goal here is to write a program that first asks the user for their first name and stores it in a `char` array, and then asks the user for their last name and stores it in a second `char` array. Your program will then create a new `char` array that contains the user's full name and uses that new array to print the user's full name to the screen.

Hint 1: it actually isn't quite correct to say that a "string" in C is just an array of `char` values. A "string" in C is actually an array of `char` values with the requirement that at least one of the values be ASCII code 0 (usually written `'\0'` and called the *null character*). The "string" consists of the characters from the beginning of the array up to *but not including* the first null character. Characters from the first null character to the end of the array are not considered part of the string. So if `s` is a `char` array of length 256, and `s[0] = 'H'`, `s[1] = 'i'`, and `s[2] = '\0'`, then s represents the string `"Hi"` regardless of the remaining values of `s`. You will not need to do any explicit calculations with the null character in this assignment, but it is helpful in understanding some of the built-in functions. You might want to look at pp.30, 38–39 of the Kernighan and Ritchie *The C Programming Language* for more information on strings.

Hint 2: The function `scanf` reads in characters until it hits white space. Since the user may enter a white space character such as a space, you will not be able to use `scanf`. Instead you should use the `fgets` function. An example usage is

```
fgets(&plaintext[0], 256, stdin)
```

which will read from standard input (which corresponds to the keyboard, so the user typing) until either a new line character (`'\n'`) is entered by the user or $256 - 1 = 255$ characters

are entered. That is, the resulting string consists of the characters up to *and including* the first newline character; if there is no newline character in the first 255 characters, then the resulting string consists of the first 255 characters. The reason that `fgets` reads in one fewer characters than its second argument is that it adds a null character at the end of the characters that the user types. The resulting characters are then stored in the array `plaintext` (which should have been set to have size 256). When you use `fgets`, the third argument should always be exactly `stdin`.

Hint 3: To determine how many characters have been entered by the user, there are several options. One option is to use the `strlen` function: if `s` is a string (i.e., a `char` array with at least one null character), then

```
strlen(s)
```

returns the number of characters in the array up to but not including the first null character. Note that with respect to the program you must write for this problem, if the user enters a string of fewer than 255 characters, the last character of the string will be the newline character. To use `strlen` in your programs, you must have the following directive at the beginning of your program:

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

Challenge: Repeated code is often undesirable in a program, instead the code should be put in a function. Can you modify your code to use a function to perform the step to read in a string? Can you modify your code to use a function to copy the contents of a given array to the specified portion of another array?