ENGR 151

Fall 2023

Lab 7: Arrays, Functions, and Recursion

Files to turn in

This lab assignment is due the day before your next lab session at 11:59pm . Please turn in the following files on autograder.io.

- print2Darray.cpp
- words.h
- fib_recursive.cpp

Background Reference Material

You've learned a lot of new concepts in the past few lectures: functions, arrays, passing by value vs. passing by reference, and recursion. Below are some sections you can use to review or reference information on these topics.

FUNCTIONS

A function allows you to write code once and use it multiple times. When you create a function, it is important to define what **input** it will receive, what **output** it will produce, and what will be **changed** (if you pass a variable by reference).

Remember that passing a variable to a function *by value* means you are giving it a copy of your data. Passing a variable to a function *by reference* means you are giving it access to change your actual data.

Example of declaring a function that passes by reference:
```
void pass_by_ref_fxn(int &var1, int &var2);
```

Example of declaring a function that passes by value:
```
int pass_by_val_fxn(int var1, int var2);
```
Remember to...

- Declare functions before you use them in main. This could mean defining them before main, meaning the entire function is written before main. Or you could declare them before main (only put the function header before main) and then define the function after main. The latter option may be preferable if you have many functions or long functions.
- Return the variable specified in the function header. If the function is "`int my_func(...)`" you should "`return int_variable;`" in the function. If the

function is “`void my_func(...)`”, you should “`return;`” in the function.

ARRAYS

An array is a **fixed-size** (meaning you should not be changing an array’s size after the array has already been declared!!), indexed structure whose elements must all be of the same type. Here are a few examples of how to declare and initialize an array for your reference:

```
// declare a 1D array of type int of size 5:
int arr[5];

// initialize a 1D array of type int of size 5:
int arr[5] = {0, 1, 2, 3, 4};
```

They can also be multi dimensional…

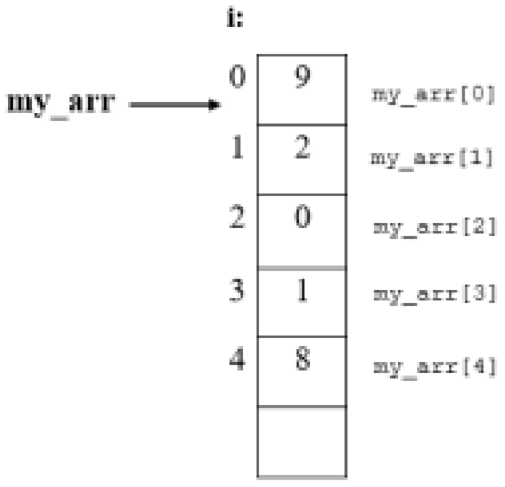
```
// declare a 2D array of type int of size 2 by 5:
int arr[2][5];

// initialize a 2D array of type int of size 2 by 5:
int arr[2][5] = { {0, 1, 2, 3, 4}, {5, 6, 7, 8, 9} };
```

Remember that the name of an array is automatically passed to functions by reference. (if You’re curious, why, it’s because the name of the array is really a pointer to the first value in the array.)

```
void pass_array_fxn(char array[]);
```

Traversing array by index
The following code iterates through an array and calculates the sum of its elements

```
int my_arr[5] = {9, 2, 0, 1, 8};
int sum = 0;
for (int i=0; i<5; ++i) {
     sum += my_arr[i]; }
```



Traverse by Index

Character arrays or “c-strings”
C-strings are really just arrays of type `char`. C++ also has a `std::string` class which provides many nifty functions, but C-strings are often useful to interface with standard functions. When you create a c-string, as with a regular array, you are allocating memory, which will accommodate the number of elements you ask for. One seemingly different aspect is the way in which c-strings are initialized. When you run:

```
char cat_name[10] = "Cleo";
```

you are getting 10 bytes of memory (since a char is 1 byte) and then filling the first four entries of `cat_name` with “Cleo”. However g++ also inserts a null-terminator (‘\0’) after the string. This is what tells c-string functions where the string ends. What you end up with is the following:

`cat_name =`

C	l	e	o	\0	?	?	?	?	?

Where the “?” indicates that we can’t know what’s there (it’s not initialized).

RECURSION

The word ‘recursion’ means defining something in terms of itself. In Computer Science, it essentially is a programming technique where operations are defined in terms of themselves, say a function that calls itself. **This is similar to looping in the sense that a block of code is repeatedly executed in the program.**

Examine the following code:

```
void recurse(){
     recurse(); /* Function calls itself */
}

int main(){
     recurse();
     return 0;
}
```

What do you think would happen if you run this program? Maybe it runs forever?? Actually, the program crashes giving you a ‘stack overflow’ error. This is because every time a function is called, the program allocates a stack frame and every time a function returns, the frame gets popped off. However, in this case the functions never return, so more and more memory gets allocated and never popped off until the program runs out of stack space and crashes. What we need is a condition which stops the recursive function from calling itself. **Thus**

the most important step in defining a recursive function is to identify the stopping condition or ‘base case’.

The base case is usually an if-statement that checks some variable for a condition (such as a number being less than or equal to zero) and if that condition is met, it will not allow the function to call itself again.

Writing recursive functions can be confusing. A good place to start is to think of what the base condition is.

Exercise 1: Printing a 2D array

In this exercise you will modify the file **print2Darray.cpp**, which can be found on Canvas. You will see that a 2D array is created in the main function.

Your task is to create a function that will print the contents of a 2D array to the terminal.

Custom printing functions can be useful to have on hand for when you are debugging. Unlike MATLAB, you cannot simply "`cout << array`" to print out a 2D array. This will in fact give you the memory address of the base element of the 2D array (`array[0][0]`) in the RAM (most variables are stored in the random access memory). Try this out! To see a 2D array printed, you need to iterate through each element and print it in the format you want.

Your function should be a void function that takes in a 2D array of chars with dimensions M by N. (M and N are constants specified at the top of the .cpp file.) For example, you could use the following function definition:

```
void print_array(char arr[M][N]);
```

Your task is to print each row of the array on a new line and print a space in between the elements of each row. **The last element of each row must have a space character after it before a new line is initiated.**

Declare your function at the top, define it at the bottom, and call it in main by passing it the array `arr`, which was defined in main.

You may compare the expected output with **printed_array.txt** before submitting!

Exercise 2: Word Functions

In this exercise you will write 3 functions to practice working with functions, arrays, and passing by reference vs. by value.

Very important!!! Make sure to test each function as you write it! There is a test case suggestion in the description of every function in the starter code file **starter_code_words.h**. You will write your test cases in **starter_code_words.cpp**, which is also on Canvas!

Your task is to **complete (define)** the three functions given in the .**h** file. Here are the 3 functions you have to write:

// Receives a C-string (array of type chars with a null terminator '\0' at the end) // Returns the integer length of the string

int word_length(char word[])

//**Note**: the length of a string does not include the null terminator

//**This function may come in very handy for writing the next 2 functions**

// Receives two C-strings **word1** and **word2** that contain a word each

// Returns nothing

// After the function completes:

/* **word1 will contain the concatenated words followed by '\0'. That is, the variable word1 will contain the concatenation of the words present in the arguments word1 and word2. ***/

// word2 will remain the same

void concatenate(char word1[], char word2[])

/* **Note:** word1 must have enough space to hold word2 and '\0'. The assumption is word1 has "space" greater than the length of word1 and word2 and the null character '\0' (this takes up 1 character). "Space" refers to the number of characters the char array is **predefined** to hold. */

/* eg. char word1[100] can hold 100 characters and the length of both your words combined with the null character at the end should be less than 100.*/

// Receives a C-string, num_upper passed by reference, num_lower passed by reference // Returns nothing

// After the function completes:

// **num_upper should be the number of uppercase letters in the word** // **num_lower should be the number of lowercase letters in the word**

// **word should be the same word except with all letters made lowercase**

void check_case(char word[], int &num_upper, int &num_lower)

//Variables of type "char" in C++ are actually encoded as numbers (called ASCII //representation). Upper case letters are assigned numbers 65 - 90 where 65 is A and 90 //is Z. Lower case letters are assigned numbers 97 - 122 where 97 is a and 122 is z. //More information can be found here: www.asciitable.com.

Note: For this lab, you are not allowed to use built-in string functions, like strlen(), toupper(), tolower() etc. String concatenation through built-in functions is also not allowed. Use the ASCII representation to manipulate the string as necessary.

A sample **starter_code_words.cpp** file is given on Canvas where you can include and test your functions with the suggested test cases. Rename **starter_code_words.h** to **words.h** before submission. No need to upload your **.cpp** file. It is for testing your code!

Exercise 3: Revisiting Fibonacci - Recursive Edition!

Last lab, you wrote two versions of a Fibonacci number generator: one using while loops and one using for loops. In this exercise, you are to write a recursive function that generates the nth number in the Fibonacci sequence. Write your function using the starter code given in the file **fib_recursive.cpp** on canvas. You should call your function in the main function to make sure it works as you expect.

In this version, you only have to print the nth number as opposed to all numbers in the sequence up to n. For example, calling

```
cout << "The 8th fibonacci number is: " << fib(8) <<
endl;
```

Would result in the following message being printed:

```
The 8th fibonacci number is: 13
```

Hints: Your function should take one integer input and return one integer output, for instance:

```
int fib(int num)
```

where `num` is which number in the Fibonacci sequence you will print. Remember that recursion is similar to looping in the sense that a block of code is repeatedly executed in the program. Think about what code should be repeated. Don't forget to define your base cases so that the function will eventually end. Recursion may be hard to wrap your mind around at first, but in this case your code will be more concise than the looping version!

Note: Please make sure you do not have keywords "while" and "for" in your program, even in your comments. The autograder checks for these keywords to make sure your program performs this task through recursion only.

Submission instructions:

1) In your main function, take one user input for the number `n`
2) The format of the output should be as follows: (Please do not make any changes to the given output and input statements in the starter code for this exercise)

```
which element of the fibonacci sequence would you like?
fib(3): 1
```