CSE30: Lab #4 – Functions

Overview

This lab will explore **functions** in the C++ programming language, and implement variations of the problems in the previous labs, using functions. The knowledge required to complete this lab will be everything learned so far, either in the lectures or in the labs. Specifically, significant portions of code written for the previous labs can be re-used for this lab.

Note: You need to have a separate program for each part of this lab. When you submit your assignment through CatCourses, make sure that ALL PARTS are included.

Getting started

Create a new directory in your main development directory (probably on Desktop/CSE30) called Lab_04. Try to use the terminal on your own, without getting help from the TA to setup the new directories (try to learn/remember the terminal commands).

Reminder: Once you have created a file and understood its content, you want to compile the source file (ex. main.cpp) into an executable so that you can run it on your computer. Note: EVERYTIME you modify your source file, you MUST re-compile your source file for the changes to take effect. You will compile from the command line by running:

```
g++ <source> -o <executable>
```

where g++ is a program (already installed on your Linux system) that compiles C++ source files, **source** is the source file for your program (main.cpp in this example), **-o** tells the compiler that you want to give the executable its own name, and **executable** is the name you want to give your program. In order to compile your first program, you would have to run:

```
g++ main.cpp -o main
```

Assuming that your program compiled successfully (i.e. no errors were found), you can run your program by typing **"./main"** in the terminal/console.

Good coding practices (worth 2 points!)

Writing code that is understandable by humans is as important as being correct for compilers. Writing good code will help you complete the code, debug it and … get good grades. It is very important to learn as soon as possible, because bad habits are hard to get rid of and good habits become effortless. Someone (guess who) reads your code will be in a better mood if it is easy to understand … leading to better grades! This lab will include 2 points (10% for code quality):

- Explanations with comments
- Meaningful names
- Indenting of blocks { } and nesting …
- Proper use of spaces, parentheses, etc. to
- Visible, clear logic
- One / simple statements per line
- Anything that keeps your style consistent

(Exercise) Create – combineString.cpp

In this part of the lab, you will create a function (**combineStr**) that concatenates a string by a number of times. There should be two functions in this program: main and combineStr. The descriptions of the functions are as follows.

The **main** function will

1. Prompt the user to "**Enter a string:** ".
2. Prompt the user to "**Enter a number of times:** ".
3. Call the function **combineStr**.
4. Output "**The resulting string is:** " and the resulting string.
5. Start over again. User can terminate the program by entering 0 (zero) for the number of times.

The **combineStr** function will

1. Takes in a **string** and an **integer** as arguments.
2. Concatenates the string argument by a number of times according to the integer argument.
3. Return the resulting string.

Before starting to write your program, use a piece of paper or a text editor to write the pseudocode of **BOTH** functions. You will need to submit the pseudocode in order to receive full credit. Again, there is no unique way to write pseudocode. It will be good enough as long as you can understand it and translate it to C++ code. Ask your TA if you are not sure about the structure of the pseudocode.

Example runs (input is in italic and bold):

Enter a string: ***0***
Enter a number of times: ***7***
The resulting string is: 0000000
Enter a string: ***bla***
Enter a number of times: ***3***
The resulting string is: blablabla
Enter a string: ***cse30***
Enter a number of times: ***0***

(Exercise) Create – sortArray1.cpp

In this part of the lab you will write a function (**sortArr**) that sorts an array of integers in either ascending or descending order. This function:

- Is a procedure with no return value.
- Takes in a variable of your choice (bool or int) as an argument to indicate ascending or descending order of the sorting array.
- Takes in an array of integers as an argument.
- Takes in a number as argument indicating the size of the array.
- Sorts the array in ascending/descending order.

Note:

- A 1-dimensional array ***a*** is passed **by reference** with ***a[]*** in the argument list.
- Reuse the selection sort (both ascending and descending versions) code that you developed in Lab 3 in this function.

The **main** function can reuse the codes from Lab 3 to define and read the array. Specifically, you should ask the user to enter the size of the array, by outputting:

"**Enter the size of the array:** "

to the console. If the user enters an incorrect size, you should output the error message

"**ERROR: you entered an incorrect value for the array size!**"

and exit the program. If the input is a valid size for the array, ask the user to enter the data, by outputting

"**Enter the numbers in the array, separated by a space, and press enter:** "

Let the user enter the numbers of the array and store them into a C++ array. Up to now, this should be exactly the same code as Lab 2 and 3. Then, you need to ask the user if the array should be sorted in ascending or descending order. You may choose your criteria to input a selection:

"**Sort in ascending (0) or descending (1) order?** "

Now, call **sortArr** to sort the array. After the function finishes its operations, print to the console if the output is sorted in ascending or descending order (depending on what the user had input):

"**This is the sorted array in <ascending or descending> order:** "

and then output the array in a newline.

Before starting to write your program, use a piece of paper or a text editor to write the pseudocode of the **sortArr function only**. Remember to write the pseudocode for both ascending and descending versions. You will need to submit the pseudocode in order to receive full credit. Again, there is no unique way to write pseudocode. It will be good enough as long as you can understand it and translate it to C++ code. Ask your TA if you are not sure about the structure of the pseudocode.

Example runs (input is in italic and bold):
Enter the size of the array: ***5***
Enter the numbers in the array, separated by a space, and press enter: ***4 6 8 2 5***
Sort in ascending (0) or descending (1) order? ***1***
This is the sorted array in descending order: 8 6 5 4 2

(Exercise) Create – sortArray2.cpp

In this part of the lab, your program will run similarly to sortArray1.cpp. However, the **sortArr** function will use **insertion sort algorithm** instead of selection sort. Here is the pseudocode of an insertion sort in ascending order, but you need to do both ascending and descending versions.

```
Algorithm InsertionSort(A, n)
    A ← new array of n integers
    for i ← 1 to n-1 do
        saved ← A[i ]
        j ← i
        while j > 0  AND A[j-1] > saved  do
            A[j] ← A[j-1]
            j ← j-1
        end while
        A[j] = saved
    end for
```

Before starting to write your program, use a piece of paper or a text editor to write the pseudocode of the **sortArr function only**. Remember to write the pseudocode for both ascending and descending versions. You will need to submit the pseudocode in order to receive full credit. Again, there is no unique way to write pseudocode. It will be good enough as long as you can understand it and translate it to C++ code. Ask your TA if you are not sure about the structure of the pseudocode.

Example runs are the same as that of sortArray1.cpp

What to hand in

When you are done with this lab assignment, you are ready to submit your work. Make sure you have included the following ***before*** you press Submit:

- Your **searchArray.cpp**, **sortArray1.cpp**, **sortArray2.cpp**, and a list of Collaborators.
- File attachments of the pseudocode you create in this lab. If you write your pseudocode on papers, scan them as images and attached the images. If you write your pseudocode in text editor, same them as text files and attach them in your submission.
- Documentation (in a text file) of code you used from the internet. You may want to cut-and-paste the original code as well.