ITSC 2181 – Introduction to Computer Systems – Fall 2023 Module 03 – Unit 2: Lab

Objectives

- Practice how to write small C programs, including identifying and correcting `C` compiler error messages and warnings.
- Practice the use of **arrays** in C to process data.
- Practice the use of **functions** in C to process data.

General Instructions

- Please do not write your email or user ID anywhere in the program. To identify your code, you may use your UNC Charlotte 800#

- In this lab, you will write a few short C programs. Each program needs to be in its own source code file, i.e., a file with the *`.c`* extension.

- **You need to test your code thoroughly before submitting it**.

- **To earn any credit, a program has to compile.** *You may comment-out lines that have errors to obtain partial credit for work done.*

- Programs need to compile cleanly to receive full credit, i.e., they do not produce any errors or warnings.

- It is essential that you **do your own work.**

    - **Do not use any external resources** (Internet, AI, friends, etc.). All cases of cheating will be taken very seriously.

    - **If you need help, please ask the instructor, TA/IA or CCI Tutoring center.**

Program 1 (40 points)

1. Write a function named `capitalize` that capitalizes all letters (i.e., alphabetic characters) in its argument.

2. The argument will be a null-terminated string containing arbitrary characters, not just letters.

3. You **must use array scripting to access each character** in the string individually.

4. To check if a character is alphabetic, you can use the `isalpha` function from the C Standard Library, see:
   https://en.wikibooks.org/wiki/C_Programming/ctype.h/isalpha

5. To convert a character to uppercase, you can use the `toupper` function from the C Standard Library, see:
   https://en.wikibooks.org/wiki/C_Programming/ctype.h/toupper

6. You cannot use any other C Standard Library functions, except for `printf` and `scanf`.

Use the following code to test your function:

```
char the_str[] = "test";

capitalize(the_str);
printf("%s\n", the_str);

char the_str2[] = "This IS a tesT!";

capitalize(the_str2);
printf("%s\n", the_str2);
```

Sample output is shown below:

```
TEST
THIS IS A TEST!
```

Implement a program named `strings_practice.c` that uses the function you wrote. You may use the code shown above. However, we recommend testing with other strings as well.

Program 2 (60 points)

A very common operation with arrays is sorting array elements by their values. For example, elements in the array may be reordered so that smaller elements are placed before larger elements. One of the simplest sorting algorithms is **insertion sort**, which is described in the following Wikipedia article: https://en.wikipedia.org/wiki/Insertion_sort. Make sure to **read the article before proceeding**.

Write a program, named **`sort_ints.c`**, that does the following:

1. Asks the user to enter 10 integers.
2. Stores the input values in an array.
3. Sorts the contents of the array in **ascending order**, using the insertion sort algorithm.
4. Displays the contents of the array after they have been sorted.
5. To implement the insertion sort algorithm, it is necessary to swap elements in the same array.
    a. You need to **implement a function** to do this.
    b. To swap any two elements in an array, declare an extra variable (let's call it **temp**). First, store the contents of the first element in **temp**. Copy the contents of the second element onto the first. Finally, copy the value of **temp** onto the second element. See the following code:

```
temp = numArray[firstElementIndex];
numArray[firstElementIndex] = numArray[secondElementIndex];
numArray[secondElementIndex] = temp;
```

A few sample runs are provided below:

```
Please enter 10 integers separated by a space:
97 93 92 97 90 100 95 97 96 96

The list after sorting:
90 92 93 95 96 96 97 97 97 100



Please enter 10 integers separated by a space:
9 8 7 6 5 4 3 2 1 0

The list after sorting:
```

```
0 1 2 3 4 5 6 7 8 9



Please enter 10 integers separated by a space:
999 54 5 125 77 45 1000 37 65 12

The list after sorting:
5 12 37 45 54 65 77 125 999 1000
```

For extra practice (optional), you could create a modified version of your code that uses the ***Quicksort*** *algorithm, see https://en.wikipedia.org/wiki/Quicksort you do not need to submit this alternate version.*

Submission Instructions

1. Create a folder (directory) on your computer.

2. Name the folder **`ITSC_2181_M03_U2_Lab_`*`student-id`***
   Replace *student-id* with your UNCC student ID (800#), e.g. *8001231234*

3. Download and copy your program (source code) files into this folder.

4. Compress (Zip) the folder with all its contents. You should consider keeping the files for every lab in a separate folder (directory) from your other course materials. You can then use the *Send to, Compressed Folder* command on Windows or the *Compress* command on a Mac to create the Zip file.

5. Submit a single Zip file (created in #4, above) via *Canvas*.

Grading Rubric

- This lab is worth a total of **100 points**.

- **Do your own work. Do not use any external resources** (Internet, friends, etc.). All cases of cheating will be taken very seriously. **If you need help, please ask the instructor, TA/IA or CCI Tutoring center.**

- Programs need to compile to earn any credit. *You may comment-out lines that have errors to obtain partial credit for work done.*

- Your work will be graded on three (3) major components: Logic and flow of program, output, and formatting/organization. Refer to the following table for details.

Logic and Flow of Program - 60%	
Fully Correct and code compiles without errors.	Full Credit
Minor Errors and code compiles without errors OR some required functionality is missing.	75% Credit
Major Errors and/or code compiles with warnings OR significant portions of the required functionality are missing.	50% Credit
Completely incorrect/missing/does not compile.	No Credit
Output - 30%	
Fully Correct and matches formatting and layout of sample output provided.	Full Credit
Minor Errors.	75% Credit
Major Errors.	50% Credit
Completely incorrect output.	No credit
Formatting/Organization of Code - 10%	
Code is clear, easy to read and formatter according to the guidelines. Whitespace has been used appropriately, including indentation and blank lines. Comments are used when needed.	Full Credit
Needs minor improvement.	75% Credit
Needs major improvement.	50% Credit
No formatting/organization at all.	No Credit

Additional Deductions

- Code that produces warnings: -10% per program
- Incorrectly named files: -2 points per file
- *Students who cheat on <u>any</u> course assignment, lab, test or other activity will have their course grade reduced by one letter grade, regardless of the activity's point value, if it is their first offense at UNC Charlotte.*