Lab 02 – Command Line Arguments, File I/O, and C-strings

Dalhousie University

Due Date: 2020/JUN/08

1 Purpose

The purpose of this lab is to familiarize you with accepting arguments from the command line (or drag and drop), file I/O and strings using the C programming language.

2 Objective

In this lab you will create a C project. Instructions on how to do so can be found here. Once the project is setup, you will demonstrate how to accept command line arguments, manipulate C-strings and perform file input and output.

3 Background

3.1 Command Line Arguments

Command line arguments provide a way of program control from outside the program (traditionally the command line). In C, the typical syntax is:

```
int main( int argc, char *argv[] ){
    /* your code here */
}
```

In this configuration, **argc** is the argument count (i.e., the number of arguments in the command line, including the name of the executable module) and, **argv** is the list of argument values (an array of pointers to the arguments). For example, the following command line execution:

```
$ ./mycompiler infile.c infile.obj
```

Results in:

```
argc = 3
argv[0]: "mycompiler.exe"
argv[1]: "infile.c"
argv[2]: "infile.obj"
```

It is important to note that all of the above strings are NUL-terminated. When a program is setup to take command line arguments as above, it is possible to 'drag and drop' a file onto the binary (.exe in Windows, no default extension in Linux) and have the file name in the **argv** array. Otherwise, you can pass arguments from the command line interface (CLI) as shown above.

3.2 File I/O

In computer programming, a file represents a sequence of bytes. How this sequence of bytes is interpreted depends on the software accessing the file; a text-editor treats the sequence differently than image-processing software does. In this lab, the contents of the file is to be treated as text. C (and C++) programs can access and manipulate files on your storage devices using libraries of existing high-level functions, as well as low level (OS level) calls. This lab will make you aware of the following functions: fopen(), fclose(), fgets(), fgetc(), fputs(), fputc().

3.3 C-Strings

In C, strings are represented as a pointer (covered in the next lab) to the first character in a contiguous sequence that is terminated by a NUL (`'\0'`) (the C++ STL string object will not be covered in this course). The C programming language has provided us with some utility functions that perform some string manipulation. This lab will familiarize you with the following string functions: strcat(), strtok(), strlen(), strcmp(), strchr(), strstr().

4 Procedure

4.1 Project Configuration

Create a project following the guidelines in the video Project Setup with VS Code.

4.2 File I/O and C-strings

Now that the project is setup, you will write a trivial program that uses the functions mentioned above. The program will be composed of the following.

1. Your program is built by typing "`make`" in a bash terminal
2. Your program is executed from the command line by typing "`./lab2 Dimension-.Munro.txt SortMe.txt Checksum.txt`" (command line arguments)
3. Open Dimension-A.Munro.txt (fopen())
4. Read each record (fgets())
5. Write each word (strtok()) and its length (strlen()) on a single line to an output file (fputs())
6. Count the number of commas in the Dimension-A.Munro.txt (strchr()) and display the number in **decimal format** on the screen
7. Count the number of occurrences of the word 'she' in Dimension-A.Munro.txt (strstr()) and display the number in **hexadecimal format** on the screen
8. Determine the first word in the list if SortMe.txt were to be sorted into alphabetical order (strcmp()) and print it to the screen
9. Calculate the parity byte checksum for the data in Checksum.txt. Once computed, display the contents of Checksum.txt and append the parity byte checksum to the end (fgetc(), fputc()) Hint: you must convert each character from ASCII text to an integer to correctly calculate the checksum
10. Calculate the sum compliment checksum for the data in Checksum.txt and display the results.
11. Ensure all files are closed (fclose())

5 Submission

Zip your project without the binaries (run 'make clean' before zipping) and submit the zipped folder to Brightspace before the end of lab on June 9th.

6 Grading

This lab will be graded for sufficient effort and completion. However, feedback will be given that will be useful for the implementation and testing portion of all assignments.