The University of Saskatchewan

Saskatoon, Canada

Department of Computer Science

CMPT 214– Programming Principles and Practice

Assignment 3

Date Due: October 5, 2020

Total Marks: 15

Submission Instructions

- Assignments must be submitted using Canvas.
- Programs must be written in C conforming to the C11 standard.
- Always include the following identification in your solutions: your name, NSID, student ID, instructor's name, course name, and course section number.
- No late assignments will be accepted. See the course syllabus for the full late assignment policy for this class.
- **VERY IMPORTANT**: Canvas is very fragile when it comes to submitting multiple files. We insist that you package all of the files for all questions for the entire assignment into a **ZIP** archive file. This can be done with a feature built into the Windows explorer (Windows), or with the **zip terminal command** (LINUX and Mac). We **cannot accept** any other archive formats. This means no tar, no gzip, no 7zip. Non-zip archives will not be graded. We will not grade assignments if these submission instructions are not followed.
- Instructions on "how to create zip archives" can be found here `https://canvas.usask.ca/courses/9771/pages/how-to-zip-slash-compress-your-files`

1 Background

1.1 Reading and Writing Files in C

A file is a container in computer storage devices used for storing data. You can access the contents of a file using a few functions in C. Reading input data from a file is very often much more convenient than reading input interactively from the console, especially when more than a handful of input data values are needed.

1.1.1 Types of Files

There are two types of files you should know about:

- Text files
- Binary files

1.1.2 Text files

Plain text files or just *text files* are files that are created using any simple text editors such as Notepad. The contents within the file is stored as plain text and the contents of the files are **readable by humans** without any special decoding. Generic text files often have the extension `.txt`, but many other types of files are plain text files, including program source code files (`.c, .py, .java`), and many system configuration files in LINUX. On some LINUX systems, even passwords are stored in plain text files (although the passwords are stored as encrypted plain text strings).

Plain text files store everything as text strings. In particular, numbers are stored in plain text files as strings of digit characters `'0'` through `'9'`, and possibly a `'.'` for a decimal point, or a `'-'` character to indicate a negative number.

1.1.3 Binary files

Binary data files store data values in a raw binary format identical to their binary representations in computer memory. Such files are **not readable by humans**. Data stored in binary form (0's and 1's) instead of plain text. For example, a 32-bit signed integer is stored in a binary file using exactly the same bit pattern as it has in the computer's memory and every such representable number takes up 4 bytes (32 bits) in the binary file, regardless of how many digits it has. Whereas, a 32-bit integer in a plain text file takes up a number of bytes equal to the number of digit characters needed to represent it as a string.

As a result, binary files that store numeric data generally take up less space than storing the same data in a text file.

For now, however, we are only going to concern ourselves with reading and writing **text files** in C. We might come back to this topic and work with binary files on a future assignment.

1.2 Basic File Operations

In C, the basic operations on files are:

- Opening a file:
  - Creating a new file (usually for writing).
  - Opening an existing file (usually for reading).
- Reading data from a file.
- Writing data to a file.
- Closing a file.

1.2.1 Opening a File

Before C can work with a file, the file has to be *opened*. When a file is opened, it must be specified whether it is to be opened for reading or writing. A file opened for reading cannot be written to, and a file open for writing cannot be read from.

To open a file in C, the function `fopen()` must be called. `fopen()` is declared in `stdio.h`, so we must include this header. The function prototype for `fopen()` is:

```
FILE* fopen(char* path, char* mode);
```

It's behaviour is that `fopen()` opens the file name specified by the string `path`, and whether it is opened for reading or writing is determined by the string `mode`. Once the file is opened, `fopen()` returns a pointer to something with data type `FILE`. The exact nature of type `FILE` is not important right now[1], but this pointer that we get to the data entity of type `FILE` is how we refer to an open file. We will pass this pointer to the functions that read from, write to, and close files so that those functions know which open file we want to perform the operation on, since we could potentially have more than one open file at a time! Type `FILE` is also defined in `stdio.h`.

So to open a file in C, the code would look something like this:

```
FILE *fp;
fp = fopen ("file_name", "mode");
```

You can see that we first declare a variable to hold the pointer to the `FILE`. The variable `fp` is a pointer to a data entity of type `FILE`. Such a variable is called a *file stream* or sometimes a *file handle*. Think of it as a name for the open file. You can also see that we assign the result of `fopen()` to `fp`. If the file `"file_name"` was opened successfully, `fp` will now be a valid file stream. If it was not opened successfully, then `fopen()` will return `NULL` (the "null" pointer). It is always good practice to make sure that `fopen()` does not return `NULL` because any operation performed on a file descriptor whose value is `NULL` will cause a program crash.

Finally, the `"mode"` determines whether the file is opened for reading or writing. The possible values for `"mode"` are summarized in Table 1. These file modes are for opening text files. Other modes exist that allow reading and writing of binary files, and also for opening files for both reading **and** writing at the same time, but we omit these details for now. The table also describes how `fopen()` behaves for each file mode, and the conditions under which opening will fail, causing `fopen()` to return `NULL`.

[1]Type FILE is actually a C Structure. We discuss C structures in topic 10.

Mode	**Purpose**
`"r"`	Open a file for reading. When a file opened for reading its contents cannot be modified. If the given path specifies a file that does not exist, the opening fails and `fopen()` returns `NULL`.
`"w"`	Open a file for writing. If the given path specifies a file that does not exist, then a new file with the name given is created. If the given path names a file that already exists, the file is deleted and a new empty file is created. If an existing file cannot be erased in this manner, or a new file cannot be created (for example, due to file permission settings) then the opening fails, and `fopen()` returns `NULL`.
`"a"`	Open a file in append mode. If the given path specifies an existing file, it is opened with its existing contents intact, and any future writes to the file are added to the end of the existing file. If the given path specifies a file that does not exist, a new file is created. If a file cannot be opened or created (for example, due to file permission settings) then the opening fails and `fopen()` returns `NULL`.

Table 1: Mode specifiers that can be passed to `fopen()`.

Example 1
This code fragment illustrates how to open a file for writing.

```
#include <stdio.h>

int main() {
   FILE *fp;
   fp = fopen("file_name.txt", "w");
   if(fp == NULL) {
      // File open failed.
      printf("file_name.txt␣could␣not␣be␣opened␣for␣writing.\n");
      return 1;
   }

   // File opened successfully.

   return 0;
}
```

Since the file mode was `"w"` *in this example, the file* `file_name.txt` *is created in the current working directory at the time of program execution (any existing file of that name in that directory is first deleted). If desired, you can specify the full relative or absolute path to where you want to create your file. Opening a file for reading is identical, except that the file mode passed to* `fopen()` *would be* `"r"` *instead of* `"w"`*.*

1.2.2 Reading and Writing to/from a Text File

For reading and writing to a text file, we use the functions `fprintf()` and `fscanf()`. These functions are identical to `printf()` and `scanf()` except that `fprint()` and `fscanf()` take an additional argument of type `FILE*` that specifies the file stream of the file that should be read from or written to. Their prototypes are:

```
fprintf(FILE *stream, char *format, ...);
fscanf(FILE *stream, char *format, ...);
```

The first argument is the file stream to read/write, the second argument is the format string (identical to `printf()`, `scanf()`), and any remaining arguments are variables that are matched with format specifiers in the format string.

Indeed, `printf()` is identical to calling `fprintf()` with the file stream `stdout` (standard output — the console), and `scanf()` is identical to calling `fscanf()` with the file stream `stdin` (standard input — the console). `stdin` and `stdout` are pre-defined variables of type `FILE*` defined in `stdio.h`. `fprintf()` and `fscanf()` otherwise follow the same behaviour as `printf()` and `scanf()`.

Example 2
This example shows how to write a number entered by the user to a text file called `file_name.txt`*.*

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int num;
   FILE *fp;

   fp = fopen("file_name.txt","w");
   // Caution: we should make sure fp is not NULL here!
   // Read a number from the console.
   printf("Enter␣num:␣");
   scanf("%d",&num);

   // Write the same number to the file file_name.txt
   fprintf(fp,"%d",num);
   fclose(fp);

   return 0;
}
```

Example 3
Read from the text file created from the previous example and prints it onto the screen.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int num;
   FILE *fp;

   fp = fopen("file_name.txt","r");
   fscanf(fp,"%d", &num);

   printf("Value␣of␣n=%d", num);
   fclose(fp);

   return 0;
}
```

1.2.3 Limitations of Reading Strings with `fscanf()`

It is important to understand that `fscanf()` has the same limitations when reading strings as `scanf()`. `fscanf()` will stop reading a string as soon as any whitespace is encountered, including a newline. Thus, `fscanf()` cannot read strings containing spaces.

1.2.4 Closing a file

In order to ensure that no data is lost, especially when **writing** files, a file **must** be closed when we are done using it and before the program terminates. The C standard library provides the `fclose()` function to perform file closing operation. It takes only one argument, the file stream (of type `FILE*`) of the file to be closed. The function prototype is:

```
int fclose (FILE *stream);
```

This causes the file stream `stream` to be closed. After closing, no further access to the file is possible until it is re-opened. `fclose()` returns 0 if closing was successful, and non-zero if it wasn't (this should really never happen but it's theoretically possible).

Example 4

```
FILE *fp;
```

```
fp  = fopen ("file_name.txt", "r");

// ... Read from the file ...

fclose (fp);  // Close the file
```

You can also see in the examples in the previous section that we have closed the file streams when we were done with them. It is **vitally important** to close streams that are open for writing, or data that has been written to the file might not actually be written. Writes to files are “buffered” by the operating system, which, in practice, means that the operating system actually delays writing of data to the disk until there is enough data to make it efficient to do so, because writing to the disk is a very expensive operation. If you allow a program to terminate without closing a file open for writing, any buffered data will be lost and not written to disk.

2 Assignment Problems

Question 1 (6 points):

Purpose: To practice text file output with `fprintf()`

Princess Adora, the renowned adventurer, wants to keep a record of the value of the stolen treasure they find while adventuring so they can return it to its owners.

Write a program that continually prompts the user to enter positive integers (treasure values) from the console until a non-positive integer value is read. Each positive treasure value read should be **appended** to the file `treasure_journal.txt`, which should be created if it does not already exist. The file should be written so that there is one treasure value per line.

Hints and Trivia:

- 0 is neither a positive nor a negative number.
- You may assume that all values entered are valid integers.
- You may remember from CMPT 141 that a file that has one value per line is called a *list file*. In a later assignment we hope to return to the topic of file I/O and work with *tabular files*, which have many values per line, and also files that contain general text and not just numbers.
- We strongly suggest **NOT** using CLion for this question, as it may not write the output file where you expect. You can use an absolute path for the output file if you want, but it will be easier if you do not use CLion and run the program from the command line. In that case, the output file will simply appear in the directory from which you run the program.

Question 2 (6 points):

Purpose: To practice reading data from a file with `fscanf()`

The Wizard of Wines Winery is the only wine producer in the fantastical land of Barovia. Davian, its proprietor, wants to calculate some basic statistics about his wine shipments. He has recorded the number of cases of wine he has shipped each week in a text file.

The first line of the text file contains the number weeks, n, that Davian has recorded data for, that is, the first line of the file contains the number of remaining lines in the file.

Each of the subsequent n lines of the file contains the number of cases of wine shipped in that week as a positive integer value.

Write a program that first reads n, and then reads all n of the remaining shipment sizes. Determine the number of cases in the largest shipment, the number of cases in the smallest shipment, and the average shipment size (which could be a fractional number of cases) and print these results to the console.

Hints:

- No test data is provided, so you have to create your own input files for testing and convince yourself that your program works. If you are having problems, be sure that your input file is correctly formatted. Sometimes what appears to be a program error is an unexpected formatting error in the input file!
- Your program may assume that the text file is properly formatted and contains no errors, such as a line that contains something other than an unsigned integer, or contains more than one number.

- We strongly suggest **NOT** using CLion for this question, as it will not look for input files where you expect. You can use an absolute path to the input file if you want, but it will be easier if you do not use CLion and just put the input file in the same directory as the .c and the compiled executable.

Question 3 (3 points):

Purpose: To practice the use of pointers.

Steven thinks that he has the best strawberry smoothie recipe in town, and wants to open up a smoothie stand. His recipe for *one smoothie* is as follows:

- 12 strawberries
- 0.5 tsp of sugar
- 1.5 cups of fruit juice

To run the smoothie stand, he will need to be able to make large batches of smoothies for his customers.

Your task is to help Steven calculate the amount of each ingredient he needs for *n* smoothies.

(a) Write a **single function** that will calculates the amount of each ingredient needed to make n smoothies. The function will need four parameters. The first parameter n can be a simple integer. But remember that C cannot return more than one value, so you will have to make use of pointers to allow the function to write the calculated amounts into memory. The remaining three parameters should be pointers to variables into which the required number of strawberries, sugar, and juice to make n smoothies can be stored and the function should modify those variables through the pointers so they have the appropriate values. The function should return 1 if the given value for n is less than or equal to 0, and return 0 otherwise.

(b) In your `main()` function, prompt the user for the number of smoothies, n, and do what is necessary to call your function from part (a) to obtain the amount of ingredients necessary to make n smoothies. If the function returned 0, print to the console the amounts of each ingredient as calculated by your function. If the function returned 1 (an invalid input for n), print an error message instead. You can assume that a whole number of smoothies will always be entered (no half smoothies here!), and that the number of strawberries used will always be a whole number.

Hints

- Remember that the function in part (a) only returns an integer indicating success or failure. The amount of each ingredient is not communicated back to `main()` via the return value. These vales are written into the memory pointed at by the pointers passed to the function. In this way, we can get multiple values back from a function. The arguments to the pointer parameters of your function should be the addresses of variables defined in main(). The function will then modify those variables through the pointers.

3 What to Hand In

Hand in a `.zip` file archive which contains the following files:

asn3q1.c: Your completed solution for question 1.

asn3q2.c: Your completed solution for question 2.

asn3q3.c: Your completed solution for question 3.

VERY IMPORTANT: You **must** hand in a ZIP archive containing all of the above files. You may not use any other type of archive (this means no gzip, no 7zip, etc.), and you may not submit the files individually. We regret this necessary inconvenience but failure to follow these instructions may result in your assignment not being graded. We simply do not have the resources to handle special cases when we have so many students in the class. Instructions on "how to create zip archives" ca be found here `https://canvas.usask.ca/courses/9771/pages/how-to-zip-slash-compress-your-files.`

We will not grade assignments if these submission instructions are not followed.

Grading Rubric

The grading rubric can be found on Canvas.