Assignment 2 (15% of Course Total)

Due date: 11:59pm, Feb 12, 2021

The assignment will be graded automatically. Make sure that your code compiles without warnings/errors and produces the required output. Also use the file names and structures indicated as requested. Deviation from that might result in 0 mark.

Your code MUST compile and run in the CSIL machines with the Makefile provided. It is possible that even with warnings your code would compile. But this still indicates there is something wrong with you code and you have to fix them, or marks will be deducted.

Your code MUST be readable and have reasonable documentation (comments) explaining what it does. Use your own judgement, for example, no need to explain i += 2 is increasing i by 2, but explain how variables are used to achieve something.

If you use any malloc in your answer, make sure you free the memory when you don't need it anymore.

Description

There is a total of 4 questions in this assignment. For each question, write your answer in files as required and include your student information in all of them. Unless otherwise specified, do not include any libraries. You can however write your own helper functions. Also do not print anything unless the question asks you to do so. None of these files should contain the main function, except test4.c in Question 4.

Question 1 [8 marks] (submit question1.c)

Write a recursive function that expands a positive int by adding 0's between the digits and returns the result as a positive int. If the input int has only one digit, return the input directly. Use this function header:

unsigned int expand_int(unsigned int number)

For example:

expand_int(1) should return 1

expand_int(24) should return 204

expand_ints(876) should return 80706

Essentially, the interlace_ints function from Assignment 1 can now be written like this:

```
unsigned int interlace_ints(unsigned int firstNum, unsigned int secondNum) {
        return expand_int(firstNum)*10 + expand_int(secondNum);
}
```

You can assume that the input int will not cause an overflow with the expansion (i.e., the returned value will be under the maximum possible value of unsigned int). Only include the function definition in your answer and name the source file containing it as **question1.c**. You must use recursion.

Question 2 [20 marks] (submit question2.c)

Consider the following recursive function on non-negative integers:

$foo(0) = 1$

$foo(1) = 2$

$foo(n) = 2 * foo(n-2) + foo(n-1) + n + 2, \quad for\ n \geq 2$

First **write a recursive function** that takes in one unsigned long as its input and returns the value by applying foo. Use this function header:

unsigned long foo_recursive(unsigned long n)

For example:

foo_recursive(0) should return 1

foo_recursive(10) should return 2724

foo_recursive(30) should return 2863311514

You will notice when n becomes larger, for example, 30, the function will take a long time to finish. This is because whenever n is >= 2 the function has to recursively call itself again two times, which continues until it arrives at one of the base cases.

As a comment after the description of the file, **write the number of times the foo_recursive function is called when n is 30**. For example, when n is 0, the function is called 1 time, when n is 2, the function is called 3 times (hint: there is a special kind of variable taught in the class that lets you find that).

Next, **write a non-recursive function** that does exactly the same thing but returns the result much faster. For example, it is expected that even when n is 1000 the function will take less than a second to finish. Use this function header:

unsigned long foo_fast(unsigned long n)

Only include the function definitions in your answer and name the source file containing them as **question2.c**. Remember to write how many times foo_recursive is called when n is 30 in your comments.

Question 3 [8 marks] (submit question3.h)

Define the enum Suit, struct Card, and struct Deck as described (use typedef so you can later declare variables as Suit, Card, and Deck):

- The enum Suit has 4 values: Spades, Hearts, Clubs, Diamonds
- The struct Card has 2 fields: Suit called suit, char called value
- The struct Deck has 2 fields: char* pointing to a C string called brand, Card array of size 52 called cards

Use the #define preprocessor to define a token called NUM_OF_CARDS_IN_DECK to the integer 52 and use it in your struct Deck definition.

Only include the enum and struct definitions in your answer and put them in the provided **question3.h**.

Question 4 [20 marks] (submit question4.c and test4.c)

First, **write a function** that sorts an array of C strings into ascending alphabetical order. You can use any sorting algorithm you prefer (mention its name in the comments). Use this function header:

void sortAscending(char* stringArray[], unsigned int size);

For example, using the printStringArray function provided in the test file, an alphabetically sorted list of 5 C strings would look like this (the reason why Spiderman is at the top is because uppercase letters are "smaller" than lowercase letters according to the ASCII table):

```
Spiderman
batman
storm
superman
wanda
```

(if it was "spiderman" it would be at second place)

Next, **write a function** that takes in an array of C strings (it can either be sorted or unsorted) and returns the address of the "median" as defined as follows*: the median of C strings is the middle C string when all the C strings are sorted in ascending alphabetical order*. Hence, for example, billy is the median within the names that are already alphabetically sorted: alex, alice, billy, bruce, charle. If there are even number of C strings, return the address of the larger C string between the middle two. Use this function header:

char* getMedian(char* stringArray[], unsigned int size);

Finally, **complete the main function**, so that the program asks the user for the number of words that they are going to input and reads in those words into a C string array. Then it calls the appropriate functions to find and print the median C string. Here is an example of how the program runs (note the prompts):

```
How many words are you going to input?
5
Input word #1: billy
Input word #2: bruce
Input word #3: charle
Input word #4: alex
Input word #5: alice
The median string is billy
```

(note the prompts for inputs)

```
How many words are you going to input?
0
No median to be found.
```

(when user enters 0)

Since the number of words is unknown, you must use malloc to create enough memory for the inputs. If malloc fails, terminate the program by calling the function: exit(0)

You can assume that each word contains no space and has at least 1 character. The user will input a non-negative number for number of words and if it is 0 the program will print "No median to be found.". Except the last part, only include the function definitions in your answer and name the source file containing them as **question4.c**. For the last part (complete the main function), update the **test4.c** file.

Coding Style [4 marks]

Your program should be correctly indented, have clear variable names and enough white space and comments to make it easy to read. Named constants should be used where appropriate. Each line of code should not exceed 80 characters. White space should be used in a consistent manner.

To help you to get into the habit of good coding style, we will read your code and marks will be deducted if your code is not styled properly.

Using the Makefile and Other Supplied Files

The Makefile provided in this assignment is used by a command in the CSIL machines called "make" to quickly compile your code. It is especially useful if you have multiple source files. To use it, type the following command in the prompt (make sure you are in the directory with all the files of this assignment):

```
$ make test1
```

The example above illustrates how Question 1 is compiled into an executable called "test1" when using the Makefile. Replace the "test1" with "test2", …etc. for other questions. You can then run the executable by typing "./test1" to test your code for Question 1. If you make changes to your code, use the make command again. You can also use "make all" if you want to compile all files at once.

The test files (test1.c, test2.c, …etc.) are provided in this assignment for you to test your code. Each typically contains a main function along with other tester functions and/or calls. You can modify them to further test your code, but do not submit these test files (**except test4.c in this assignment**) because we will be using our test files that are similar but not identical to grade your assignment. This makes sure that your code is not written to produce hard-coded output.

The header files (question1.h, question2.h, …etc.) are there to make the compilation work. Ignore them, do not modify them, and do not submit them (**except question3.h in this assignment**).

Submission

Submit **only the files indicated above** by compressing them into a zip file (**do not** put them into a folder and zip them) and upload it to Canvas **by 11:59p Feb 12**. Name the zip file in this format: **<firstname_lastname>_<studentID>_Assignment2.zip**.

For example, John_Smith_012345678_Assignment2.zip

Assignment late penalty: 10% per calendar day (each 0 to 24 hour period past due), max 2 days late.

Academic Honesty

It is expected that within this course, the highest standards of academic integrity will be maintained, in keeping with SFU's Policy S10.01, "Code of Academic Integrity and Good Conduct." In this class, collaboration is encouraged for in-class exercises and the team components of the assignments, as well as task preparation for group discussions. However, individual work should be completed by the person who submits it. Any work that is independent work of the submitter should be clearly cited to make its source clear. All referenced work in reports and presentations must be appropriately cited, to include websites, as well as figures and graphs in presentations. If there are any questions whatsoever, feel free to contact the course instructor about any possible grey areas.

Some examples of unacceptable behavior:

- Handing in assignments/exercises that are not 100% your own work (in design, implementation, wording, etc.), without a clear/visible citation of the source.
- Using another student's work as a template or reference for completing your own work.
- Using any unpermitted resources during an exam.
- Looking at, or attempting to look at, another student's answer during an exam.
- Submitting work that has been submitted before, for any course at any institution.

All instances of academic dishonesty will be dealt with severely and according to SFU policy. This means that Student Services will be notified, and they will record the dishonesty in the student's file. Students are strongly encouraged to review SFU's Code of Academic Integrity and Good Conduct (S10.01) available online at: http://www.sfu.ca/policies/gazette/student/s10-01.html.