Assignment 3 (15% of Course Total)

Due date: 11:59pm, Mar 19, 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.

Question 1 [16 marks] (submit question1.c)

Recall the Quicksort algorithm where items are sorted by picking a "pivot" and rearranged by comparing their values with the pivot.

Implement the Quicksort algorithm by writing a function that takes in a C string array, its size, and a bool variable indicating whether it is sorted according to length or alphabetical order. Use this function header:

void quicksort_cstrings(char stringArray[], int size, bool byLength)*

Hint: You are likely going to implement a few helper functions like swap and partition to make the code more manageable. For the implementation of the partition function, choose the last item as the pivot. You can also use functions from string.h which is already included for you in question1.h.

You can assume that each word contains no space and has 1 to 63 characters. The user will input a non-negative number for number of words and if it is 0 the program will print "No words to be sorted.".

For example, here is the output generated by the provided test1.c file when the user inputs 0:

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

As another example, if the user inputs 5 words: of, cute, A, lot, birds, then the order sorted by length (byLength variable is true) is: A, of, lot, cute, birds; whereas the order sorted by alphabetical order (byLength variable is false) is: A, birds, cute, lot, of:

```
How many words are you going to input?
5
Input word #1: of
Input word #2: cute
Input word #3: A
Input word #4: lot
Input word #5: birds
User words printed by length:
A
of
lot
cute
birds
User words printed by alphabetical order:
A
birds
cute
lot
of
```

As shown in the test1.c file, since the number of words is unknown, malloc is used to create enough memory for the inputs. If malloc fails, the program terminates by calling the function: exit(0)

Only include the function definition(s) in your answer and name the source file containing it as **question1.c**. You must implement the sorting algorithm without using predefined sorting functions.

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

Recall in Assignment 2 Question 3 you were asked to define the enum Suit, struct Card, and struct Deck, along with the NUM_OF_CARDS_IN_DECK token defined as 52:

- 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

First, **write a function** that initializes all the fields in a Deck variable by taking in the address of the variable and a C string as the brand name. For each Card variable, it has a suit (of type Suit) and a value (of type Char) that are ‘A’, ‘2’, ‘3’, …, ‘9’, ‘T’, ‘J’, ‘Q’, ‘K’. Use this function header:

void initializeDeck(Deck theDeck, char* brandName);*

Next, **write another function** that shuffles the cards in the cards field. This shuffling can be achieved by the Fisher-Yates shuffle algorithm that can be found here (refer to the answer by John Leehey):

https://stackoverflow.com/questions/6127503/shuffle-array-in-c

Use this function header:

void shuffleDeck(Deck theDeck);*

Note that in the provide test2.c file there is a call to the "srand(0)" function, which starts the random number generator properly. Do not modify or call this "srand(0)" in your answer as this will make the numbers generated unpredictable and thus difficult to mark. You will only need to call "rand()" everytime you want a random number generated for you.

Lastly, **write a function** that prints the cards of a Deck 13 cards per row. Use this function header:

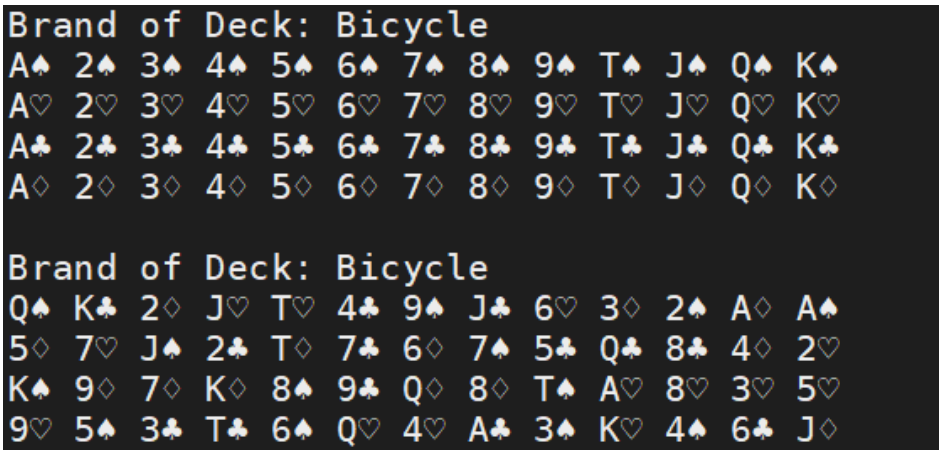
void printDeck(const Deck theDeck);*

Read carefully at the sample output. In particular, there is no space in the front and the back of each row, but a space between each card. To print the suits, refer to the following sample printf calls:

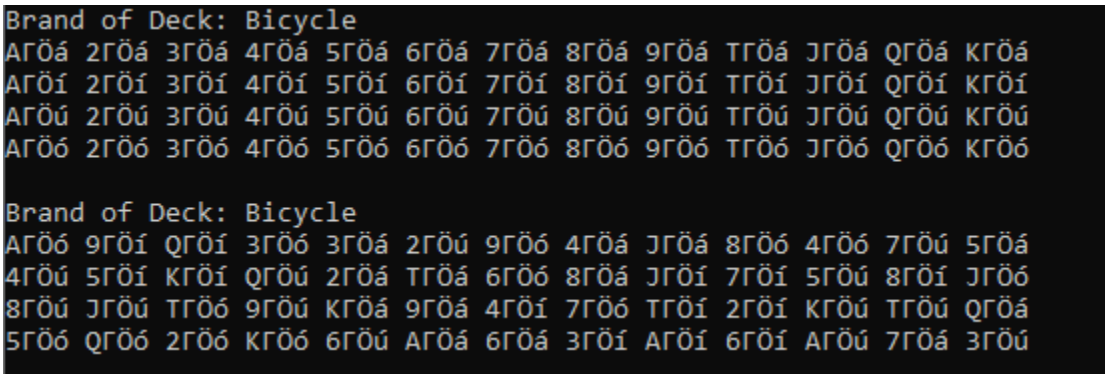
printf("\u2660\u2663"); //prints the spades symbol followed by the clubs symbol

printf("\u2661\u2662"); //prints the hearts symbol followed by the diamonds symbol

You will notice that some operating systems, for example, Windows, do not print the symbols properly. If that happens, run your code in a CSIL workstation or some online coding IDEs.



Above: suit symbols properly printed in a CSIL workstation. Second group is shuffled.



Above: suit symbols not being properly printed in a Windows 10 machine. Second group is shuffled.

Only include the function definitions in your answer and name the source file containing them as **question2.c**. Use the definitions of the structs and enum provided in question2.h for your answer.

Question 3 [12 marks] (submit question3.c)

Implement the following functions of a stack that stores C strings as its items (read the .h file for full details on how each function should behave):

cStringStack cStringStack_create();*

void cStringStack_push(cStringStack csStack, char* cStr);*

char cStringStack_pop(cStringStack* csStack);*

bool cStringStack_isEmpty(cStringStack csStack);*

void cStringStack_free(cStringStack csStack);*

void cStringStack_reverse(cStringStack csStack);*

cStringStack cStringStack_duplicate(cStringStack* csStack);*

Note that since the number of C strings to be stored is unknown, your implementation should dynamically grow/shrink by requesting/releasing memory to add/remove items. It must not use any size in advance.

The provided test3.c file implements a simple program that reads in user inputs and stores them into the stack structure. It will only be fully working if you have correctly implemented all the functions above. See the sample outputs as examples:

```
Input a command and press enter,
use "undo" to remove previous,
or "quit" to quit:
undo
Stack is empty, nothing to pop.
```

undo as the first command

```
Input a command and press enter,
use "undo" to remove previous,
or "quit" to quit:
ls

Input a command and press enter,
use "undo" to remove previous,
or "quit" to quit:
mkdir Assignment2

Input a command and press enter,
use "undo" to remove previous,
or "quit" to quit:
undo
Previous being removed: mkdir Assignment2

Input a command and press enter,
use "undo" to remove previous,
or "quit" to quit:
mkdir Assignment3

Input a command and press enter,
use "undo" to remove previous,
or "quit" to quit:
cd Assignment3
```

```
Input a command and press enter,
use "undo" to remove previous,
or "quit" to quit:
quit
---
ls
mkdir Assignment3
cd Assignment3
---
---
ls
mkdir Assignment3
cd Assignment3
---
---
cd Assignment3
mkdir Assignment3
ls
---
---
ls
mkdir Assignment3
cd Assignment3
---
```

Left: a sequence of inputs. Right: ended with the command “quit”.

Only include the function definitions in your answer and name the source file containing them as **question3.c**.

Question 4 [8 marks] (submit question4.txt)

In our lecture we mentioned the Binary Search algorithm can be implemented using recursion.

First, **write the recursive formula** of its time complexity, as well as the time complexity of the base cases (when the array size is 0 or 1).

Next**, derive the time complexity** of the recursive Binary Search algorithm. Show your steps. To show exponential terms, use ^ to denote the power, e.g., 2^3 means 2 to the power 3; to show bases, use _ to denote the subscript, e.g., log_2(8) means log of 8 base 2.

Include the top comments as in the other files along with your answer, and name the file containing them as **question4.txt**. Use a text editor to create this file (the same IDE you use for the assignment would work, just make sure it has a **.txt** extension instead of .c or .h).

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 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.

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 Mar 19**. Name the zip file in this format: **<firstname_lastname>_<studentID>_Assignment3.zip**.

For example, John_Smith_012345678_Assignment3.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.