Assignment 2 (5% of Course Total)

Due date: 11:59pm, Oct 13, 2023

Part of 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, what a certain loop is doing, and the purpose of each #include.

Description

There is a total of 3 questions in this assignment. For each question, write your answer in a single file that contains your student information as comments at the top. **Unless otherwise specified, do not include any pre-existing libraries in your answers**. You can however write your own helper functions. Also, do not print anything unless the question asks you to. **None of these files should contain the main function**. (except for Question 3).

Question 1 [4 marks]

Write **a recursive solution** that does exactly the same as what the function in Question 1 of Assignment 1 does. That is, "shuffles" the digits in an unsigned int number and returns the result as an unsigned int with this rule: swap the first digit with the last digit, swap the second digit with the second last digit, …etc. Use this function header:

unsigned int shuffleDigitRecursive(unsigned int number)

For example:

shuffleDigitsRecursive(1) should return 1
shuffleDigitsRecursive(123) should return 321
shuffleDigitsRecursive(30400) should return 403 (leading zeros will not be part of the resulting number)

You can assume the number does not have leading zeros, and we will not test shuffleDigitsRecursive(0).

This question aims to let you practice recursion. One way to do so is to look at your answer for Assignment 1, separate the recursive case (most likely where you use a loop) from the base case (most likely the last step you do before returning the result). You should be able to see a significant simplification to your code. **Do not use any global or static variables, or pass-by-reference (if you do so you get 0 for this question)**.

This question is tricker than you might think, especially if you want to do it with arithmetic (+-*/%)! Hint: solve it by adding a recursive helper function where shuffleDigitRecursive simply calls it for the recursion.

Only include the function definition (and your helper functions, if any) in the source file and name it as **a2_question1.c**. Aim at defining everything in at most 15 lines (excluding comments).

Question 2 [6 marks]

In the header file for this question, a struct called Talk is defined like this:

```
typedef struct {
   short hours;
   short minutes;
   short seconds;
   char* title;
   char* overview;
} Talk;
```

Write a function that takes in 5 parameters: three shorts representing hours, minutes, and seconds respectively, a char* array representing a title, and a char* array representing an overview; and returns the address of a dynamically (i.e., uses malloc) created Talk struct variable storing those parameters.

Use this function header:

Talk createTalk(short hours, short minutes, short seconds, const char* title, const char* overview)*

For example, given the code (*title* and *overview* are Cstrings storing the proper information):

```
Talk* talk = createTalk(1, 27, 10, title, overview);
printf("%dh%dm%ds\n%s\n%s\n", talk->hours, talk->minutes, talk->seconds,
     talk->title, talk->overview);
```

will result in an output like this:

```
1h27m10s
AI in Retail: Personalizing the Shopping Experience
Explore how AI is reshaping the retail industry, focusing on personalized recommendations, inventory
management, and customer engagement. Discuss the potential to enhance customer satisfaction and
optimize operations using AI-powered solutions.
```

You can assume all the hours/minutes/seconds are valid (e.g., minutes will not be negative or over 59), and all the Cstrings are properly formed (\0 terminated). Member variables *title* and *overview* in the struct **must be created dynamically and are copies of the parameters**, instead of simply pointing to the parameters' addresses. This is called "deep-copy".

Next, write another function that takes in 1 parameter: the address of a Talk struct variable; and releases (i.e., uses free) the memory created for the 2 member variables *title* and *overview*.

Use this function header:

void clearTalk(Talk talk)*

Note that the talk parameter can be NULL, and if so the function should do nothing. Also, this function does not release the memory used for the Talk struct variable, but only those used by *title* and *overview*. To release all the memory dynamically allocated for the struct variable, you should call the free() function with the address of this struct variable right after the function returns. For details read Question 3.

Only include the function definitions (and your helper functions, if any) in the source file and name it as **a2_question2.c**. Do not use recursion in your answer.

Question 3 [6 marks]

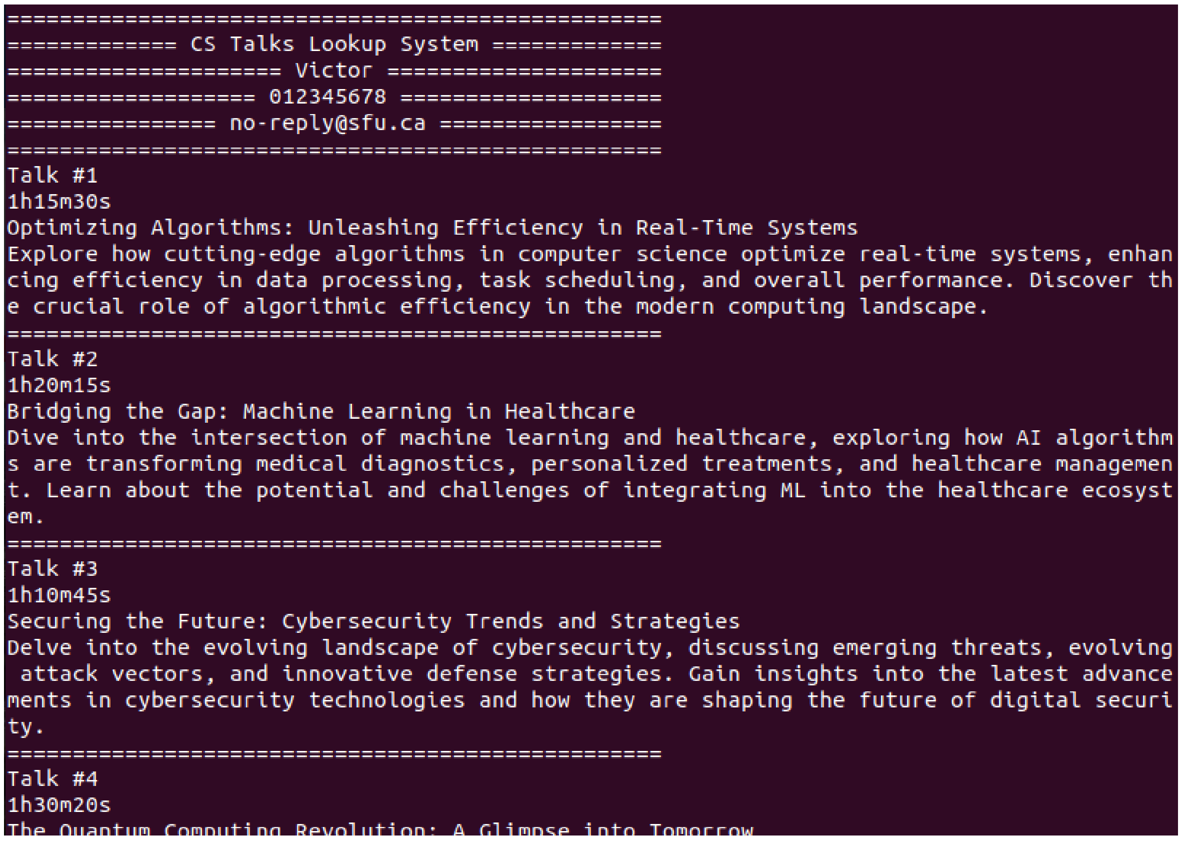
In this question you are going to make use of your answer for Question 2 to create a working program. You are also going to write **your own main function** so the program runs.

Create a program that reads all the talk information from the provided CSTalksList.txt file and prints the information to the screen. Your program must meet these requirements:

- Include the header file from Question 2 and use the functions defined there (createTalk and clearTalk) to complete this question. Do not redefine those functions.
- Use a dynamic array of Talk struct pointers to store the talk information. This is the recommended approach (instead of a static array with a fixed size) as we might change the number of entries in the provided file, and dynamic arrays can accommodate that variation.
- There must be no memory leaks (e.g., your program requests some memory but does not release them all before it terminates). We have set up the Makefile so that if that happens, after running your program it will tell you some memory leaks are detected (no messages if none is leaked).
- Start with a fancy banner. There is no specific requirement besides it must include your name, 9-digit SFU ID, and your SFU email address. Let your creativity shine, just nothing offensive.
- Print all the entries from the first to the last, along with a talk #. Do not reorder the entries*.

*The entries are randomly generated using ChatGPT with a specific format and then lightly modified. Hence, the content might not make sense and have very similar wording patterns – no need to worry.

Here is a sample output (the program terminates once the last entry is printed, note the talk #):



```
===================================================
============= CS Talks Lookup System =============
===================== Victor =====================
=================== 012345678 ====================
================ no-reply@sfu.ca =================
===================================================
Talk #1
1h15m30s
Optimizing Algorithms: Unleashing Efficiency in Real-Time Systems
Explore how cutting-edge algorithms in computer science optimize real-time systems, enhan
cing efficiency in data processing, task scheduling, and overall performance. Discover th
e crucial role of algorithmic efficiency in the modern computing landscape.
===================================================
Talk #2
1h20m15s
Bridging the Gap: Machine Learning in Healthcare
Dive into the intersection of machine learning and healthcare, exploring how AI algorithm
s are transforming medical diagnostics, personalized treatments, and healthcare managemen
t. Learn about the potential and challenges of integrating ML into the healthcare ecosyst
em.
===================================================
Talk #3
1h10m45s
Securing the Future: Cybersecurity Trends and Strategies
Delve into the evolving landscape of cybersecurity, discussing emerging threats, evolving
 attack vectors, and innovative defense strategies. Gain insights into the latest advance
ments in cybersecurity technologies and how they are shaping the future of digital securi
ty.
===================================================
Talk #4
1h30m20s
The Quantum Computing Revolution: A Glimpse into Tomorrow
```

And here are some hints for you to write your code:

- Use a loop to go through the file and call the functions from Question 2 in each iteration to build the Talk array. A while-loop is likely the best as the number of entries in the file can change.
- To learn to read and write to files see section “C Programming Files” in https://www.programiz.com/c-programming or https://www.tutorialspoint.com/cprogramming/c_file_io.htm or any other online resources.
    - In particular, look at the fscanf and fgets functions from stdlib.h (you’ll use both).
- Don’t forget to close the file after reading it.
- To create a dynamic array with unknown size, a typical strategy is to start with a small array and double its size when it is at capacity. For example:

  ```
  int capacity = 16; //initial small size
  int used = 0; //no items in use yet
  int *intArray = (int *)malloc(sizeof(int)*capacity); //create an array of size 16
  //… after filling up the array, keep increasing used each time you assign an item
  if (used == capacity) {
    capacity = capacity * 2; //double the capacity
    intArray = (int *)realloc(sizeof(int)*capacity); //request an array of size 32, keeping the items
  }
  ```

  Use this to build your dynamic array to store the talk information (in this example each item is an int, in your code each item should be **a pointer** to a Talk struct variable).

You can assume the file will always have the name **CSTalksList.txt** and will be available in the same directory as the program. You can also assume the format for a talk entry is consistent (first line starts with **Duration:**<space>, second line starts with **Talk Title:**<space>, third line starts with **Overview:**<space>, fourth line are 3 hyphens), with each line having at most 300 characters. However, there is no guarantee on how many talk entries are inside the file (the provided file is just an example), though each talk will have the exact same set of information.

Include the main function (and your helper functions, if any) in the source file and name it as **a2_question3.c**. You can include any libraries that are covered in class (i.e., stdio, stdlib, string, math).

Coding Style [4 marks]

Your program should be properly indented, have clear and meaningful variable names (e.g., no single-letter variable names except loop iterators) 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. Remember to include your information.

Keep your code concise and efficient. If your code is unnecessarily long or inefficient (e.g., hard-code for all possible cases, extraneous function calls), we might deduct marks. 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 Assignment 2):

```
$ 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”, “test3”, …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 your code 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. You can look at them but do not modify them. You also do not have to submit them.

Submission

Submit **only the 3 source files** indicated above (a2_question1.c, a2_question2.c, a2_question3.c) to CourSys. Refer to the corresponding Canvas assignment entry for details.

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.

Use of ChatGPT or Other AI Tools

As mentioned in the class, we are aware of them. I see them as helpers/tutors from which you can look for inspiration. However, these tools are not reliable and they tend to be overly confident about their answers, sometimes even incorrect. It is also very easy to grow a reliance to them and put yourself at risk of not actually learning anything and even committing academic dishonesty. Other issues include:

- When it comes to uncertainty, you won't know how to determine what is correct.
- If you need to modify or fine tune your answer, you won't know what to do.
- You will not be able to learn the materials and thus will not ne able to apply what you learn in situations where no external help is available, for example, during exams and interviews.

For introductory level courses and less sophisticated questions, it is likely that you'll get an almost perfect answer from these tools. But keep in mind if you can get the answer, everyone can also get the answer. Bottomline is, if you ask these tools to give you an answer and you use the answer as yours, you are committing academic dishonesty by claiming work that is not done by you as yours.

Note that different instructors might have different policies regarding the use of these tools. Check with them before you proceed with assignments from other courses.