Assignment 2 (5% of Course Total)

Due date: 11:59pm, Oct 11, 2024

This is an individual assignment. 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 using the steps covered in class (see the **Before You Submit** section for details). 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 along with the required information stated in the question. **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 [6 marks]

Write **a recursive solution** that does exactly the same as what the function in Question 1 of Assignment 1 does. That is, replaces a specific digit in an int number with another digit and returns the result as an int. Return the same number if the target/replacement character is not a digit. Use this function header:

int replaceDigitsRecursive(int number, char target, char replacement)

For example (for more examples refer to the description in Assignment 1):

replaceDigitsRecursive(1, ‘1’, ‘2’) should return 2
replaceDigitsRecursive(-232, ‘3’, ‘0’) should return -202
replaceDigitsRecursive(998, ‘9’, ‘0’) should return 8 (leading zeros will not be part of the resulting number)

You can assume the number does not have leading zeros (e.g., we will not call *replaceDigitsRecursive(01, ‘1’, ‘2’)*), except when the number is actually zero (i.e., we might call *replaceDigitsRecursive(0, ‘1’, ‘1’)*).

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)**.

Depends on how you solve this, you might have to create a recursive helper function that passes an extra parameter preserving the partial answer. IOnly include the a2_question1.h header file and the function definition (and your helper functions, if any) in the source file and name it as **a2_question1.c**.

Question 2 [4 marks]

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

```
typedef struct {
   char* name;
   short feetInHeight;
   short inchesInHeight;
   char* superpower;
   char* traits;
} Superhero;
```

First, write a function that takes in 5 parameters: a pointer to a char array representing a name, two shorts representing feet in height and inches in height respectively, a pointer to a char array representing a superpower, and a pointer to a char array representing the traits; and returns the address of a dynamically (i.e., uses malloc) created Superhero struct variable storing those parameters.

Use this function header:

Superhero createSuperhero(const char* name, short feetInHeight, short inchesInHeight, const char* superpower, const char* traits)*

For example, given the code (*name, superpower* and *traits* are Cstrings storing the proper information):

```
Superhero* superhero = createSuperhero(name, 6, 0, superpower, traits);
printf("%s\n%d\'%d\"\n%s: %s\n", superhero->name, superhero->feetInHeight,
     superhero->inchesInHeight, superhero->superpower, superhero->traits);
```

will result in an output like this:

Thunderstrike
6'0"
Weather Control: Harnessing storms, Thunderstrike commands thunder and lightning, using nature's fury to protect the innocent and fight against evil.

You can assume all the height measurements are valid (i.e., feet & inches will not be negative), and all the Cstrings are properly formed (\0 terminated). Field variables *name*, *superpower*, and *traits* 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 Superhero struct variable; and releases (i.e., uses free) the memory created for the 3 field variables *name*, *superpower*, and *traits*.

Use this function header:

void clearSuperhero(Superhero superhero)*

Note that the parameter can be NULL (if so the function should do nothing). Also, this function does not release the memory used for the struct variable, but only those used by the variable's fields. 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 a2_question2.h header file and 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 (so finish it first). You are also going to write **your own main function** so the program runs.

Create a program that reads all the superhero information from the provided superheros.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 (createSuperhero and clearSuperhero) to complete this question. Do not redefine those functions.
- Use a **dynamic array of Superhero struct pointers** to store the superhero 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 will use a tool called **valgrind** to test the program compiled from your code which reports if there is any memory leaks (refer to Canvas for details).
- 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 Superhero #. 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 Superhero #):

```
==================================================
============ Superheros Lookup System ============
====================== Victor ======================
=================== 012345678 ====================
================ no-reply@sfu.ca =================
==================================================
Superhero #1
Thunderstrike
6'0"
Weather Control: Harnessing storms, Thunderstrike commands thunder and lightning, using
 nature's fury to protect the innocent and fight against evil.
==================================================
Superhero #2
Shadow Weaver
5'8"
Shadow Manipulation: Shadow Weaver bends darkness to her will, creating illusions and c
loaking herself, striking fear into the hearts of villains.
==================================================
Superhero #3
Iron Guardian
6'4"
Enhanced Strength: With unbreakable armor, Iron Guardian defends the weak, his immense
strength serving as a bulwark against tyranny.
==================================================
Superhero #4
Mystic Echo
5'6"
```

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 dynamic 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 superhero information (in this example each item is an int, in your code each item should be **a pointer** to a Superhero struct variable).

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

Write 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, stdbool).

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.

Before You Submit

Before submitting your assignment, make sure you have test-run your code in our CSIL machines using the steps covered in class: **open your assignment folder in VS Code, use the gcc command with options in the terminal to compile your code, and run the executable**. We will use these steps when marking

your submission and will deduct marks if your code does not compile/run – to maintain fairness we do not accept reasons like "but it worked on my computer" or "but it ran just fine in my IDE".

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 (a2_question1.h, a2_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.
- Sharing your work with or making your work available on any platform for anyone who would benefit from it (e.g., other students in the course).
- 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. You can, however, ask general questions like "how do I write a for-loop?", "explain mergesort to me" and use the answers to help you come up with your own answers – make sure you have cited it at the top of your code/essay as comments by stating what tool you used what questions you asked, and how you used the answers. Type everything yourself.

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.

