CMPT125, Spring 2022

Homework Assignment 2

Due date: Friday, February 18, 2022, 23:59

You need to implement the functions in ***assignment2.c****.*
Submit only the ***assignment2.c*** file to CourSys.

Solve all 3 problems in the assignment. This means you need to implement 5 functions total.

The assignment will be graded automatically.
Make sure that your code compiles without warnings/errors, and returns the required output.

Your code MUST compile in CSIL with the Makefile provided.
If the code does not compile in CSIL, the grade on the assignment is 0 (zero).
Even if you can't solve a problem, make sure the file compiles properly.

Warning during compilation will reduce points.
More importantly, they indicate that something is probably wrong with the code.

Memory leak during execution of your code will reduce points.
Check that all memory used for intermediate calculations are freed properly.

Your code must be readable, and have reasonable documentation, but not too much.
No need to explain i+=2 with // increase i by 2.

An example of a test file is included.
Your code will be tested using the provided tests as well as additional tests.
Do not hard-code any results produced by the functions as we will have additional tests.
You are strongly encouraged to write more tests to check your solution is correct, but you don't have to submit them.

1. You need to implement the functions in ***assignment2.c****.*
2. If necessary, you may add helper functions to the assignment2.c file.
3. You should not add main() to assignment2.c, because it will interfere with main() in the test file.
4. Submit only the ***assignment2.c*** file to CourSys.

Question 1 [40 points].

In this question you will need to implement a database for a dictionary. Each entry in a dictionary consists of a term and its meaning/definition. You will store your dictionary in a file. Write the following two functions, maintaining a file holding a database for a dictionary.

```
    int add_term(const char* file_name,
                 const char* term, const char* definition);
```

The function gets the name of a file, a term and its definition.
- If the term is not in the file, the function adds the pair term/definition to the file and returns 1.
- If the term is already in the file, the function does not modify the file and returns 0.
- If the file does not exist, the function creates a new file with the given name, adds the pair to the file, and returns 1.

```
    char* find_term(const char* file_name, const char* term);
```

The function gets the name of a file and a term. It searches the file for the given term.

- *If the term is found, the function returns a string containing the definition.*
- *If the file does not exist or the term is not in the file, the function returns NULL.*

Additional instructions and hints:

1. For the instructions on how 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.
2. There are no specific instructions about how you should store the information in the file. The only requirement is that the *two functions are compatible with each other*. That is, if a term is added using add_term, then find_term will be able to find it. You should decide carefully on the format for storing the data of each entry.
3. When storing the pairs term/definition, remember that you need to store the actual strings and not their pointers. It may be convenient to store the length of the string in the file.
4. The terms might consist of several words.
5. You should not assume that the lengths of the strings are bounded. Some terms or definitions might be very long, say, longer than 1000 chars.
6. The characters in the definition might be non-alpha-numeric.
7. Don't forget to close the file at the end of each function.

Question 2 [20 points].
Define the following variant of the Fibonacci sequence called Fib3:

- *fib3(0) = 0*
- *fib3(1) = 1*
- *fib3(2) = 2*
- *fib3(n) = fib3(n-1) + 2*fib3(n-2) + 3*fib3(n-3) for all n>=3*

That is, the sequence is 0, 1, 2, 4, 11, 25, 59, 142, 335, 796, 1892, 4489...

```
uint64_t fib3(unsigned int n);
```

Your function needs to compute and return the correct value for all n<50 under 1 second.

Question 3 [40 points].
Implement the following two functions that get a string, and compute an array of non-empty tokens of the string containing only lower-case letters. For example:

- For a string "*abc EFaG hi*", the list of tokens with only lower-case letters is *[*"*abc*", "*hi*"*].*
- For a string "*ab 12 ef hi* ", the list of such tokens is *[*"*ab*","*ef*","*hi*"*].*
- For a string "*abc 12EFG hi*   ", the list of such tokens is *[*"*ab*","*hi*"*].*
- For a string "   *abc*   ", the list of such tokens is *[*"*abc*"*].*
- For a string "+*abc!! B" the list of such tokens is *empty.*

That is, we break the string using the spaces as delimiters (ascii value 32), and look only at the tokens with lower-case letters only .

1. *[15 points] The function* `count_tokens` *gets a string* `str`*, and returns the number of such tokens.*

   ```
   int count_tokens(const char* str);
   ```

   For example
   - `count_tokens("`*abc EFaG hi*`")` *needs to return 2.*
   - `count_tokens("`*ab 12 ef hi*`")` *needs to return 3.*
   - `count_tokens("`*ab12ef+*`")` *needs to return 0.*

2. *[25 points] The function* `get_tokens` *gets a string* `str`*, and returns the array with the tokens containing only lower-case letters in the correct order. The length of the array should be the number of tokens, computed in* `count_tokens`*.*

   ```
   char** get_tokens(const char* str);
   ```

   For example:
   - `get_tokens("`*abc EFaG hi*`")` *needs to return* `["`*abc*`","`*hi*`"]`
   - `get_tokens("++a+ b + c")` *needs to return* `["`*b*`","`*c*`"]`.
   - `get_tokens("`*ab12ef+*`")` *needs to return either NULL or an empty array.*

Remark: Note that the returned array and the strings in it must all be dynamically allocated.