CMPT125, Fall 2021

Homework Assignment 2
Due date: Wednesday, October 15, 2021, 23:59

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

Solve all 4 problems in the assignment.

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.

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.
You are strongly encouraged to write more tests to check your solution is correct, but you don't have to submit them.

You need to implement the functions in ***assignment2.c****.*
If necessary, you may add helper functions to the assignment2.c file,
but you should not add main() to the file.
Submit only the ***assignment2.c*** file to Canvas.

Question 1 [30 points].
Write the following two functions maintaining a database of entries of type song in a file. (see .h file for the definition of the struct song)

```
int add_song(const char* file_name, const song s);
```

The function gets the name of a file, and a song. If the song is not in the file, the function adds it to the file and returns 1. Otherwise, the function does not modify the file and returns 0.

```
song* find_song(const char* file_name, const char* title);
```

The function gets the name of a file and a title of a song. It searches the file for the song with the given title. If a song is found, it returns a pointer to the song with all the details. If not found, the function returns NULL. You may assume the song titles are unique.

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 song is added using add_song, then find_song will be able to find it. You should decide carefully on the format for storing the data of each song.
3. When storing the title and artists, remember that you need to store the actual string and not the pointer to it. Also, it may be convenient to store the length of the string in the file.
4. 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) = fib(n-1)+fib(n-2)+fib(n-3) for all n>=3*

That is, the sequence is 0,1,2,3,6,11,20,37,68,125...

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

Your function should work correctly for inputs up to n=60, and run under a second.

Question 3 [20 points].
Implement ***two*** *recursive versions of the linear search that gets an array of songs and a title and searches for a song with the given title in the array.*

1. *[10 points] In this version the function returns the* ***first*** *index of the array containing the number. If the number is not in the array, the function returns -1.*

   ```
   int linear_search_rec_first(int* ar, int length, int number);
   ```

2. *[10 points] In this version the function returns the* ***last*** *index of the array containing the number. If the number is not in the array, the function returns -1.*

   ```
   int linear_search_rec_last(int* ar, int length, int number);
   ```

Question 4 [30 points].
Implement the following two functions that allow breaking a string into non-empty tokens using a given delimiter. For example,
- *For a string "abc-EFG-hi", and a delimiter '-' : the list of tokens is ["abc", "EFG", "hi"]*
- *For a string "abc-EFG---hi-", and a delimiter '-' : the list of tokens is ["abc", "EFG", "hi"]*
- *For a string "abc", and a delimiter ' ' : the list of tokens is ["abc"]*
- *For a string "++abc++", and a delimiter '+' : the list of tokens is ["abc"]*

That is, we break the string using the given delimiter, and the tokens are only the non-empty substrings.

1. *[10 points] The function* `count_tokens` *gets a string* `str`*, and a char* `delim`*, and returns the number of tokens in the string separated by delim.*

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

   For example
   - `count_tokens("abc-EFG--", '-')` *needs to return 2.*
   - `count_tokens("++a+b+c", '+')` *needs to return 3.*
   - `count_tokens("***", '*')` *needs to return 0.*

2. *[20 points] The function* `get_tokens` *gets a string* `str`*, and a char* `delim`*, and returns the array with the tokens 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, char delim);
   ```

   For example:
   - `get_tokens("abc-EFG--", '-')` *needs to return* `["abc","EFG"]`
   - `get_tokens("++a+b+c", '+')` *needs to return* `["a","b","c"]`.
   - `get_tokens("***", '*')` *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.