

## CMPT125, Fall 2025

### Homework Assignment 2

Due date: Friday, October 10, 2025, 23:59

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

Solve all 5 problems in this assignment, one function for each problem.

**Grading:** The assignment will be graded automatically.

Make sure that your code compiles without warnings/errors, and returns the required output.

**Compilation:** 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.

**Warnings:** Warnings during compilation will reduce points.

More importantly, they indicate that something is probably wrong with the code.

**Dynamically allocated arrays:** Do not use variable length arrays! Never!

If you need an array of unknown length, you need to use malloc.

**Memory leaks:** Memory leaks during execution of your code will reduce points.

Make sure all memory used for intermediate calculations are freed properly.

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

Write helper functions if that makes the code more readable.

**Testing:** 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 all the functions in ***assignment2.c***.
2. You should not add `main()` to `assignment2.c`, because it will interfere with `main()` in the test file.
3. Submit only the ***assignment2.c*** file to CourSys.

### Problem 1 [40 points]

In this question you will implement a database of usernames and passwords. You will store your dictionary in a file. Write the following two functions, maintaining a file with the database.

```
int add_user_password(const char* file_name,
                     const char* username, const char* password);
```

The function gets the user and a password.

- If the user is not in the file, the function adds the pair to the file and returns 1.
- If the user 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.

```
int check_user_password(const char* file_name,
                       const char* username, const char* password);
```

The function gets the filename and the username/password, and searches the file for them.

- If the username is found and the password matches, the function returns 1.
- If the file does not exist, the function returns -1.
- If the username is not found, the function returns -2.
- If the username is found but the password doesn't match, the function returns -3.

### 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](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 pair is added using `add_user_password`, then `check_password` 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, 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 usernames are always *alpha-numeric*. The password may contain *non-alpha-numeric* symbols, e.g. spaces, underscores, or special symbols. Any symbol (other than '\0') is allowed to be in the password, even '\n'.
5. You should not assume that the lengths of the strings are bounded. Some usernames and passwords might be very long, say, longer than 1000 chars.
6. Don't forget to close the file at the end of each function.

## Problem 2 [20 points]

Define the following variant of the Fibonacci sequence called Fib3:

- $\text{fib3}(0) = 0$
- $\text{fib3}(1) = 0$
- $\text{fib3}(2) = 1$
- $\text{fib3}(n) = \text{fib3}(n-1) + \text{fib3}(n-2) + \text{fib3}(n-3)$  for all  $n \geq 3$

That is, the sequence is 0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149...

```
int64_t fib3(unsigned int n);
```

## Problem 3 [40 points]

Write 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 " ", and a delimiter ' ': the list of tokens is []
- 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. [20 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.
- `count_tokens("abcaa", '*')` needs to return 1.

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, ending with the NULL pointer.

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

For example:

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

The length of the returned array should be the number of tokens + 1 (+1 is for the NULL indicating the end of the array).

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