CMPT125, Spring 2026

Homework Assignment 2
Due date: Friday, February 13, 2026, 23:59

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 provided Makefile.
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!
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.

Working with other students: You can discuss the assignment with other students, but you need to write your own code.

Using AI tools: The use of AI tools is not permitted.

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. You may add helper functions.
4. 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 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 like '\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]
Let p>2 be a parameter, and define the following variant of the Fibonacci sequence:

- *fib3_p(0) = 0, fib3_p(1) = 1, fib3_p(2) = 2*
- *fib3_p(n) = fib3_p(n-1) + fib3_p(n-2) + fib3_p(n-3) (mod p) for all n>=3*

For example, for p=17, the sequence is 0, 1, 2, 3, 6, 11, 3, 3, 0, 6, 9, 15, 13...

```
int fib3_p(unsigned int n, unsigned int p);
```

Your function needs to return the correct answer for all n<1,000,000 within 1 second.

Problem 3 [40 points]

In this problem you will implement the following pebble game played on a line.
We start at time 0 with a line of n cells, some cells being occupied by identical pebbles (marked with asterisks.
Going from time i to time i+1, each pebble moves according to the following rules:

(A) If at most one pebble is left, the next state remains the same, and the game ends.
(B) The pebble looks at the closest neighbors on the left and on the right at time i, and moves toward the closest of them.
(C) If the closest neighbors on the left and on the right are at the *same distance*, the pebble disappears.
(D) If there are at least two pebbles, the leftmost pebble jumps one step to the right.
(E) If there are at least two pebbles, the rightmost pebble jumps one step to the left.
(F) If two pebbles land at the same place, then both disappear.
(G) If the next state is the same as the previous one, the game ends. (Remember, only the occupied locations are considered ,and not the the names of the pebbles)

See examples below (the colors are for convenience only).

Example 1:
```
Step 0: -**------*--*- // apply rules (D)(B)(B)(E)
Step 1: -**-------**-- // game over by rule (G)
```

Example 2:
```
Step 0: ---*--*--*-- // apply rules (D)(C)(E)
Step 1: ----*---*--- // apply rules (D)(E)
Step 2: -----*-*---- // apply rules (D)(E) + by rule (F) ** disappear
Step 3: ------------ // game over by rule (A)
```

Example 3:
```
Step 0: ---*----*---*- // apply rules (D)(B)(E)
Step 1: ----*----*-*-- // apply rules (D)(B)(E) + by rule(F) ** disappear
Step 2: -----*-------- // game over by rule (A)
```

Example 4:
```
Step 0: -*-*-*-*------*--*--*- // * * * disappear by rule (C)
Step 1: --*---*--------*---*-- // apply rules (D)(B)(B)(E)
Step 2: ---*-*----------*-*--- // all disappear by rule (F)
Step 3: ---------------------- // game over by rule (A)
```

Example 5:
```
Step 0: ---*---------- // game over by rule (A)
```

[25 points] *Write the function* `evolve(const char* state)` *that gets a string representing a state of the game, and computes the next state of the game. The function returns the string representing the next state. For the format, see examples below:*

```
char* evolve(const char* state);
```

For example:

- *If* `state` *is* `"---"`, `evolve()` *returns* `"---"`.
- *If* `state` *is* `"--**----**-"`, `evolve()` *returns* `"--**----**-"`.
- *If* `state` *is* `"---*---*---*-"`, `evolve()` *returns* `"----*-----*--"`.
- *If* `state` *is* `"-*-*-*-*---*--*--*-"`, *it returns* `"--*---*-----*---*--"`.

[15 points] *Write the function* `last_state(const char* state)` *that gets a string representing the initial state of the game, and computes the last state of the game. The function returns a string representing the last state. For the format, see examples below:*

```
char* last_state(const char* state);
```

For example:

- *If* `state` *is* `"----"`, `last_state()` *returns* `"----"`.
- *If* `state` *is* `"--**----*--*-"`, `last_state()` *returns* `"--**-----**--"`.
- *If* `state` *is* `"---*---*---*-"`, `last_state()` *returns* `"-------------"`.
- *If* `state` *is* `"--*-*-*-*----"`, `last_state()` *returns* `"-------------"`.
- *If* `state` *is* `"--*---*-*----"`, `last_state()` *returns* `"---*---------"`.

Comments about `evolve()` *and* `last_state()`*:*

1. *You may assume that the input is always in the correct format: empty cells are represented by* `'-'` *and occupied cells by* `'*'`.
2. *You may assume strlen(state)>0.*
3. *The returned string must be allocated on the heap.*
4. *Even if the returned string is identical to the given state, the returned string must be allocated on the heap.*