

CMPT125, Fall 2020

Homework Assignment 2

Due date: Wednesday, October 21, 2020, 23:59

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

Solve all 4 problems in the assignment.

The assignment will be graded both **automatically** and by **reading your code**.

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

Readability: Your code should be readable. Add comments wherever is necessary. If needed, write helper functions to break the code into small, readable chunks.

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 it compiles

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

Testing: 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 assignment1.c file,
but you cannot add add main() to
Submit only the **assignment2.c** file to CourSys.

Question 1 [20 points]

Consider the following recursive function on positive integers:

```
foo(0) = 0
foo(1) = 1
foo(2) = 2
foo(n) = foo(n-3) + foo(n-2) - foo(n-1) - 1 for n > 2
```

Write a function that gets an integer n and computes $foo(n)$.

```
long foo(int n)
```

Your function should return the correct answer in reasonable time on inputs up to 1000.

Question 2 [30 points (15 points each item)]

Consider the struct

```
typedef struct {
    char* name;
    int id;
} person;
(see .h file for the definition of person)
```

A. Implement the recursive version of the linear search algorithm.

The function gets an (unsorted) array of length n with elements of type `person`, and an `ID`. The output of the function is the index of the item in the array that contains the person with the given ID. If A does not contain the item, the function returns -1.

```
int linear_search_rec(const person* A, int n, int id)
```

B. Implement the recursive version of the binary search algorithm.

The function gets an array **sorted by ID** of length n with elements of type `person`, and an `ID`. The output of the function is the index of the item in the array that contains the person with the given ID. If A does not contain the item, the function returns -1.

```
int binary_search_rec(const person* A, int n, int id)
```

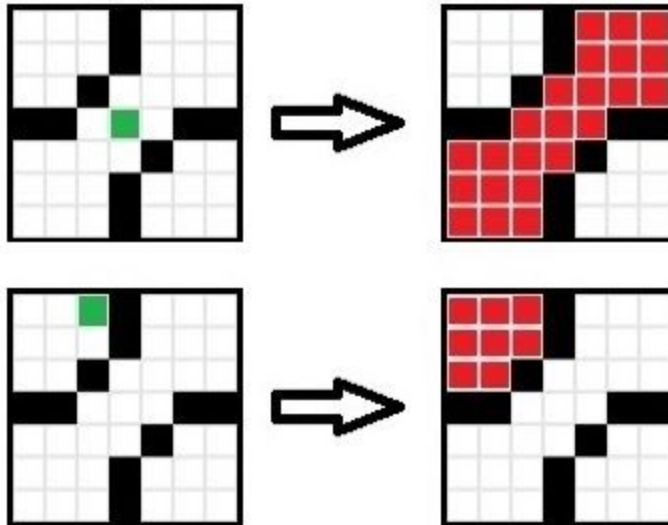
Question 3 [35 points]

Write a function that gets an NxN matrix of colors, and a starting point, and colors the WHITE area connected to the starting point with RED.

See examples below:

The *green point* on the left represents the starting point.

The *red points* are the area containing the starting point.



Specifically, the colors are represented as ints:

WHITE corresponds to the number 0, RED corresponds to 1.

All other numbers are treated as other colors.

The arguments of the function are the dimensions of the array and the starting point (see .h file for the definition of the type `point`)

```
void flood_fill(int N, int ar[N][N], point start)
```

The function colors the array as in flood fill algorithm.

Your function should work in reasonable time on arrays of size up to 50x50.

[Hint: use recursion]

[Hint2: the code should not be longer than 15 lines]

Question 4 [40 points (20 points each)]

Write the following two functions maintaining a database of entries of type `person` in a file.
(see `.h` file for the definition of `person`)

```
person* find_person(const char* file_name, int ID)
```

The function gets a name of a file and searches the file for a person with the given ID. It returns a pointer to the struct `person` (on heap) with the name and ID of the person. If the file does not exist or if the person with the required ID is not in the file, the function returns the `NULL` pointer.

The file should not be changed by this function.

```
int add_person(const char* file_name, person p)
```

The function gets a name of a file, and a person `p`.

If the person with the same ID is already in the file, the function does nothing.

If the person is not in the file, the function adds `p` to the file.

If the file does not exist, a new file is created and the person is added to the file.

The function returns 0 if the person is added to the file.

The function returns 1 if the person has already been in the file.

The function returns -1 if there was an error (e.g. error opening a file).

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 person is added using `add_person`, then `find_person` will be able to find it.
You should decide carefully on the format for storing the data of each person.
- 3) When storing the name, remember that you need to store the name and not the pointer to it. Also, it may be convenient to store the length of the name in the file.
- 4) Don't forget to close the file at the end of each function.