# CS 137: Assignment #5

Due on Friday, Nov 1, 2024, at 11:59PM

Submit all programs using the Marmoset Submission and Testing Server located at

https://marmoset.student.cs.uwaterloo.ca/

*Victoria Sakhnini*

Fall 2024

## Notes:

- Use the examples to guide your formatting for your output. Remember to terminate your output with a newline character.
- For this assignment, you may use any content covered until the end of Module 8.
- You should use `double` when appropriate; do not use `float` type.

## Problem 1

C is a strongly typed language. This means that all variables must be assigned a type explicitly. Other languages, such as JavaScript and Python, do not require you to assign a type to each variable. We will emulate a more flexible type system in C for this question. Consider the following structure definition:

```
typedef struct FlexData {
      bool isInt;
      int value_int;
      double value_float;
}FlexData;
```

Create a function `FlexData flexDivide(FlexData a, FlexData b)` that will divide `a/b` according to the indicated data type (`int` or `double`). If either `a` or `b` is a `double`, treat the whole operation as a real division. Otherwise, treat it as an integer division. The function returns the result. **Assume b is not zero**.

Note: You are to submit `flexdata.c` containing only your implemented function and the structure definition (you must delete the test cases portion which is the `main` function). However, **you must keep the required included libraries.**

The sample main function for testing is below.

```c
int main(void){
    FlexData n1 = {true, 17, 0};
    FlexData n2 = {true, 5, 0};
    FlexData d1 = {false, 0, 14.3};
    FlexData d2 = {false, 0, 2.4};

    FlexData r1 = flexDivide(n1,n2);
    FlexData r2 = flexDivide (d1,d2);
    FlexData r3 = flexDivide (n1,d2);
    assert(r1.isInt);
    assert(!r2.isInt);
    assert(!r3.isInt);
    assert(3==r1.value_int);
    printf("%lf\n", r2.value_double);
    printf("%lf\n", r3.value_double);

    return 0;
}
```

The expected output:

```
5.958333
7.083333
```

## Problem 2

You are given a dataset of employees' information, which includes their ids, ages, and salaries. The dataset is represented using an array of structures named Employee. Your task is to implement a C program that performs the following operations (some of them are already implemented for you in the provided `company.c` file):

Input: Read data for n employees from the user (0<n ≤ 100).

Display: Display the information of all employees.

Average Salary: Calculate and return the average salary of all employees.

Youngest Employee: Find and display the details of the youngest employee. Assume there is only one.

Salary Range: Return the number of employees whose salaries fall within a given range (inclusive).

Salary Raise: update the salary of all employees with a given raise.

Complete `company.c` program that accomplishes these tasks using the provided array of structures and appropriate functions.

Note: You are to submit `company.c` containing only your implemented functions along with the structure definition (that is, you must delete the test cases portion and the `main` function). However, **you must keep the required included libraries.**

Here is an example of the execution of the provided main function:

The bold part is the user input

Enter the number of employees (up to 100): **3**
Enter the details of 3 employees:
Employee 1:
ID: **1111**
Age: **45**
Salary: **120500**
Employee 2:
ID: **22222**
Age: **33**
Salary: **145900**
Employee 3:
ID: **3333**
Age: **55**
Salary: **345889**
Employee Details:
Employee 1:
ID: 1111
Age: 45
Salary: 120500.00
Employee 2:
ID: 22222
Age: 33

Salary: 145900.00
Employee 3:
ID: 3333
Age: 55
Salary: 345889.00
Average Salary: 204096.33
Youngest Employee:
ID: 22222
Age: 33
Salary: 145900.00
Enter the salary range:
Minimum Salary: **200000**
Maximum Salary: **300000**
Number of employees in the given salary range: 0
Salaries after 0.05 raise
Employee Details:
Employee 1:
ID: 1111
Age: 45
Salary: 126525.00
Employee 2:
ID: 22222
Age: 33
Salary: 153195.00
Employee 3:
ID: 3333
Age: 55
Salary: 363183.45

## Problem 3

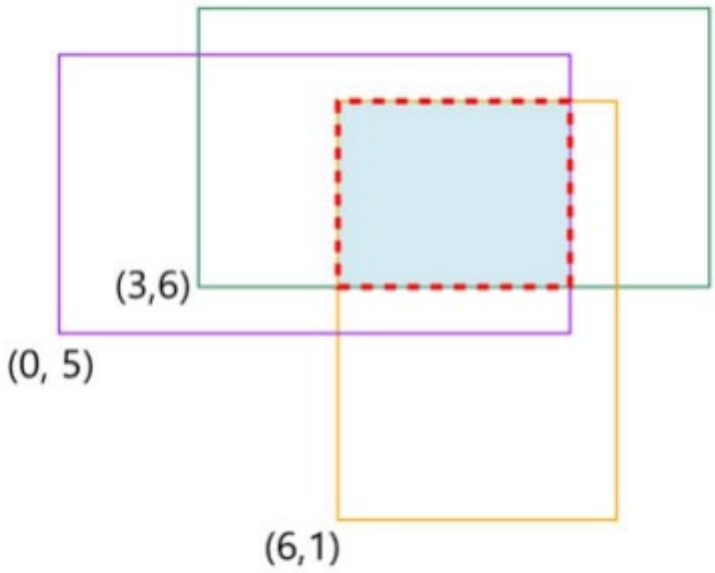
Consider the following struct.

```
typedef struct Point{
      int x;
      int y;
} point;
typedef struct Rectangle {
   point bottomLeft; // represents the bottom left corner of the rectangle
   int width;
   int height;
} rectangle;
```

Write the following function:

```
rectangle intersection(rectangle rects[], int n).  // n>1
```

which takes in an array or `rectangle` structs of length `n`, computes the rectangle we get from the intersection of these rectangles in the array, and returns it. For instance, in the following example, for the three input rectangles (green, purple, orange), the intersection resulting from the intersection of the three rectangles is the blue rectangle with the red dotted boundary.




If all the rectangles do not intersect (i.e. an empty intersection) then return `rectangle` with the values `{{0,0},0,0}`. Otherwise return the result (the intersection) as of type `rectangle`.

Complete and Submit the provided `rec.c` file containing only your implemented function (that is, you must delete the test cases portion/the main function). However, **you must keep the required included libraries.**

Sample Test program (You need to add more test cases)

```
1. #include <assert.h>
2.
3. int main(){
4.
5.    rectangle result;
6.    rectangle r = {{2,6},3,4};
7.    rectangle s = {{0,7},7,1};
8.    rectangle t = {{3,5},1,6};
9.    rectangle u = {{5,6},3,4};
10.
11.   // Test 1
12.   rectangle rects1[2] = {r, s};
13.   result = intersection(rects1, 2);
14.   assert(result.bottomLeft.x == 2);
15.   assert(result.bottomLeft.y == 7);
16.   assert(result.width == 3);
17.   assert(result.height == 1);
18.
19.   // Test 2
20.   rectangle rects2[3] = {r, s, t};
21.   result = intersection(rects2, 3);
22.   assert(result.bottomLeft.x == 3);
23.   assert(result.bottomLeft.y == 7);
24.   assert(result.width == 1);
25.   assert(result.height == 1);
26.
27.   // Test 3
28.   rectangle rects3[4] = {r, s, t, u};
29.   result = intersection(rects3, 4);
30.   assert(result.bottomLeft.x == 0);
31.   assert(result.bottomLeft.y == 0);
32.   assert(result.width == 0);
33.   assert(result.height == 0);
34.
35.   return 0;
36. }
```

## Problem 4

Complete implementing the Sequence ADT in `sequence.c`. Read carefully through the global constant and function declarations in `sequence.h` to get a better understanding of what each function should do and how it should behave.

You are also provided with a sample program for testing called `test_sequence.c`

Make sure you add more tests to test your implementations.

Note: You are to submit `sequence.zip` containing only `sequence.h` and `sequence.c`

If you implement the functions correctly, the `test_sequence.c` file must print the following:

```
[1]
[empty]
[1]
[1,1]
[empty]
[1]
[1,1]
[2,1,1]
[3,2,1,1]
[4,3,2,1,1]
[4,3,2,5,1,1]
[4,3,2,5,1,1,4,3,2,5,1,1]
[4,3,2,5,1]
```