CS 101, Spring 2013 — May 2nd — Exam 4

Note: Make sure your programs produce the output in **exactly** the format described, including capitalization and punctuation. You may not receive credit for programs that produce incorrectly formatted output.

Getting started: Start **Cygwin Terminal** and **Notepad++** closing any open tabs in **Notepad++**. (Note: Do *not* open **ANY** other programs.) Your instructor will give you the name of a zip file. In Cygwin Terminal, run the following commands:

```
cd h:
mkdir -p CS101
cd CS101
curl -O http://faculty.ycp.edu/~dhovemey/spring2013/cs101/assign/zipfile
unzip zipfile
cd CS101_Exam4
```

Substitute the name of the zip file for *zipfile*.

Editing code: Use Notepad++ to open the source file (e.g., `question1.cpp`) referred to in the question. **Do not open any files other than the ones for the exam.**

Compiling: To compile the program for Question 1, run the following command in Cygwin Terminal:

```
make question1.exe
```

Change the number as appropriate for the other questions (e.g., `question2.exe`).

Running: To run the program for Question 1, run the following command in Cygwin Terminal:

```
./question1.exe
```

Change the number as appropriate for the other questions (e.g., `question2.exe`).

To submit: In Cygwin Terminal, run the command

```
make submit
```

Enter your Marmoset username and password when prompted.

Good luck!

Question 1. [15 points] In the program `question1.cpp`, the `struct Student` data type represents information about a student account:

```
struct Student {
    int id;              // student id
    int num_credits;     // how many credits student is registered for
    int commuter;        // 1 if student is a commuter, 0 if not
    double amount_due;   // amount due for tuition
};
```

For this question you should only add function calls in `main`. The function definitions for `init_Student()`, which initializes the fields of a `struct Student` variable, and `compute_Tuition()`, which computes the tuition amount, are provided and should **NOT** be modified. You will also not need to add or modify any code for input or output.

(1) As indicated by the `TODO` comment in `main`, write a call to the `init_Student()` function so that the variable called `the_student` is initialized with the user input values read by the provided `scanf` statements. Once you do this, when you run the program it will echo back the input values (except amount per credit) as follows (user input in **bold**):

```
ID: 9025555555
Number of credits: 15
Commuter (1=yes, 0=no): 1
Amount per credit: 400.0
Initialization function called
id=9025555555
num_credits=15
commuter=1
amount_due=0.00
```

(2) As indicated by the `TODO` comment in `main`, write a call to the `compute_Tuition()` function passing `the_student` and the amount per credit user input value read by the provided `scanf` statements. The provided function will compute the tuition and store it in the `amount_due` field of the structure. Example using the previous example values (user input in **bold**):

```
ID: 9025555555
Number of credits: 15
Commuter (1=yes, 0=no): 1
Amount per credit: 400.0
Initialization function called
id=9025555555
num_credits=15
commuter=1
Compute tuition function called
amount_due=6000.00
```

Question 2. [30 points] In the program `question2.cpp`, the `struct Transcript` data type represents the number of credits earned and the course grade for some number of classes:

```
struct Transcript {
    int num_classes;       // how many classes were taken
    int credits[MAX];      // number of credits for each class
    double grades[MAX];    // grade for each class
};
```

Your task is to define a function called `compute_GPA` that takes a *pointer* to a `struct Transcript` and returns the grade point average for that transcript. The grade point average is the sum of the quality points for all classes, divided by the total number of credits taken. The number of quality points for a single class is the number of credits times the grade earned.

The program's `main` function declares a `struct Transcript` and prompts the user to enter the number of classes and number of credits/grade for each class. Code is also provided to print out the GPA computed from the entered transcript information.

You will need to:

- Add a prototype for the `compute_GPA` function
- Add a definition for the `compute_GPA` function
- Add a call to the `compute_GPA` function to the `main` function, assigning the computed GPA to the variable called `gpa`

Example session (user input in **bold**):

```
How many classes? 3
Credits: 3
Grade: 3.5
Credits: 4
Grade: 3.0
Credits: 3
Grade: 4.0
GPA=3.450000
```

The GPA is 3.45 because

$$\frac{(3 \cdot 3.5) + (4 \cdot 3.0) + (3 \cdot 4.0)}{3 + 4 + 3} = 3.45$$

Do not change any code except as indicated by the `TODO` comments.

Make sure that your function is called `compute_GPA`, and that it takes a *pointer* to a `struct Transcript` as its only parameter.

Question 3. [20 points] In `question3.cpp` implement the function called `find_occurrences` - **DO NOT MODIFY** `main`.

Its prototype is

```
int find_occurrences(int arr[], int length, int val, int occur[]);
```

The function should scan through the elements of `arr`. Each time an array element equal to `val` is found, it should place the *index* of that element in an element of `occur`. The index of the first occurrence should be stored in `occur[0]`, the index of the second occurrence should be stored in `occur[1]`, etc. The parameter `length` indicates the number of values stored in `arr`, and you can assume that `occur` will have at least that many elements.

The total number of occurrences found should be returned as the return value of the function.

For example, if the `arr` array has the values {1, 2, 1, 3, 4, 1}, and `val` is 1, then the values {0, 2, 5} should be placed in the first three elements of `occur`, because those are the indices of the first three occurrences of the value 1 in `arr`. The function would return 3 in this case, since there were 3 occurrences of 1 in `arr`.

In your code to implement the `find_occurrences` function, *first* print out the values of the array containg the values separated by a single space, then each time an occurrence is found, print a single-line message of the form:

`Occurrence at` *N*

where *N* is the index of the occurrence.

Example run (user input in **bold**):

```
How many values? 6
Enter values: 1 2 1 3 4 1
Search for? 1
1 2 1 3 4 1
Occurrence at 0
Occurrence at 2
Occurrence at 5
Found 3 occurrence(s): 0 2 5
```

Hints:

- Use a counter variable to keep track of how many occurrences have been found; this will be useful in deciding where in `occur` to store the index of each occurrence

Question 4. [20 points] In the program `question4.cpp`, the `struct Point` data type represents a point with integer x/y coordinates, and the `struct Rect` data type represents a rectangle with specified minimum x/y values, a width, and a height:

```
struct Point {
    int x, y;
};
```

```
struct Rect {
    struct Point minxy;   // minimum x/y coordinates
    int width;            // width of the rectangle
    int height;           // height of the rectangle
};
```

The `count_inside` function's prototype is

```
int count_inside(struct Point *p, struct Rect rectlist[], int nrects);
```

Given a pointer to a `struct Point` and an array of `struct Rect` elements, the function returns a count of how many of the rectangles the point was inside. The `nrects` parameter specifies how many elements the `rectlist` array has.

A point is considered to be inside of a rectangle if $x_{\min} \leq x \leq x_{\max}$ and $y_{\min} \leq y \leq y_{\max}$, where x and y are the coordinates of the point, $x_{\min}$ and $y_{\min}$ are the minimum x and y coordinates of the rectangle, and $x_{\max}$ and $y_{\max}$ are the maximum x and y coordinates of the rectangle. The values $x_{\max}$ and $y_{\max}$ can be found by adding the rectangle's width and height (respectively) to a its minimum x and y coordinates.

Each time the `count_inside` function determines that a point is inside a rectangle, it should print a message on a single line reading `Point is inside rectangle` N, where N is the index of the rectangle in the `rectlist` array.

A `main` function - **DO NOT MODIFY** - is provided to read in the coordinates of a point and a series of rectangles, call the `count_inside` function, and print out how many rectangles the point was inside. Example session (user input in **bold**):

```
Point x/y: 10 10
Number of rects: 3
Min x/y: 0 0
Width/height: 20 20
Min x/y: 7 8
Width/height: 6 4
Min x/y: 11 5
Width/height: 20 10
Point is inside rectangle 0
Point is inside rectangle 1
Point is inside 2 rectangles
```

Important: Do not modify the program except to add code to the body of the `count_inside` function.

Question 5. [15 points] Write the prototype and definition for a function called `overlap` that takes four parameters: two `struct Circle` parameters, one `double` pointer parameter for the distance between the centers, and one `double` pointer parameter for the sum of the radii.

The function should return a bool value, indicating whether or not the circles overlap (`true` if they do, `false` if they do not). It should also store the distance between the centers and the sum of the radii in the third and fourth parameter pointer variables.

Two circles overlap if the distance between their centers is less than the sum of their radii. The distance between two points $(x1, y1)$ and $(x2, y2)$ is given by

$$\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

The `struct Point` structure contains two double fields `x` and `y`.

The `struct Circle` structure contains a `struct Point` field `c` representing the center location and a double field `r` representing the radius of the circle.

The `main` function - **DO NOT MODIFY** - reads the two circles' values as input, stores them in the structs, calls the `overlap` function, and then prints the result of the overlap check. Do not change any of the code in `main`.

Example 1 (user input in **bold**):

```
x1/y1/r1: 1 2 3
x2/y2/r2: 4 3 4
3.162 7.000 overlap
```

Another example (user input in **bold**):

```
x1/y1/r1: -2 3 2
x2/y2/r2: 6 -4 1.5
10.630 3.500 non-overlap
```

Hints:

- Note that the first two parameters of `overlap` should have the data type `struct Circle`: they do *not* take pointers
- Don't forget to store the distance between the centers and the sum of the radii in the variables pointed to by the third and fourth parameters, each of which is a pointer to a `double` variable