CISS240: Introduction to Programming
Assignment 6

Objectives

- Use binary operators.
- Write if statements.
- Write if-else statements.

Q1. The following is taken from wikipedia.org:

The Gregorian calendar, the current standard calendar in most of the world, adds a 29th day to February in all years evenly divisible by 4, except for centennial years (those ending in 00), which receive the extra day only if they are evenly divisible by 400. Thus 1600, 2000 and 2400 are leap years but 1700, 1800, 1900 and 2100 are not.

Write a program that prompts the user for a year and prints either `leap year` or `not leap year` according to the above rule.

Test 1.

```
1991
not leap year
```

Test 2.

```
2004
leap year
```

Test 3.

```
1200
leap year
```

Q2. Write a program that tells a user if the date entered was a valid date or not. Specifically, the program prompts the user for a date (an integer) in the format of yyyymmdd and prints `correct` if the date is correct, or `incorrect` if the date is incorrect. For instance, if the user enters `19910202` for February 2 of 1991, your program should print `correct`.

The only thing you can assume about the user input is that the user enters a positive integer with 8 digits.

Test 1.

```
19910229
incorrect
```

Test 2.

```
19910132
incorrect
```

Test 3.

```
19910105
correct
```

Test 4.

```
20040229
correct
```

Test 5.

```
20000229
correct
```

Test 6.

```
19000229
incorrect
```

Test 7.

```
20060631
incorrect
```

Q3. Let's redo the problem on finding roots of the quadratic equation, this time including complex roots. (No, you don't need to know algebra to solve this problem. It's a matter of following the instructions.) Recall that the solutions to the quadratic equation

$$ax^2 + bx + c = 0$$

are given by the following "recipe":

- CASE 1: If $b^2 - 4ac < 0$, the solutions are
$$\frac{-b}{2a} - \frac{\sqrt{4ac - b^2}}{2a} i$$
and
$$\frac{-b}{2a} + \frac{\sqrt{4ac - b^2}}{2a} i$$
- CASE 2: If $b^2 - 4ac \geq 0$, the solutions are
$$\frac{-b - \sqrt{b^2 - 4ac}}{2a}$$
and
$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

For the first case above, you don't have to know anything about that symbol i. You just have to follow the instructions which include printing the symbol i as `i` whenever you're in the first case. (See test cases below.)

Write a program that prompts the user for a, b, and c (as doubles) and prints the solutions according to the above "recipe". Refer to the tests for the output format. **The output must be in fixed point format with 5 decimal places.**

Test 1.

```
1 2 1
-1.00000, -1.00000
```

Test 2.

```
1 2 3
-1.00000 - 1.41421i, -1.00000 + 1.41421i
```

Test 3.

```
1 -2 2
1.00000 - 1.00000i, 1.00000 + 1.00000i
```

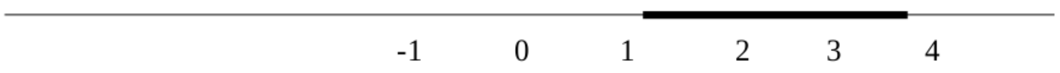
Test 4.

```
1.0 -4.2 4.0
1.45969, 2.74031
```

Test 5.

```
2 1 2
-0.25000 - 0.96825i, -0.25000 + 0.96825i
```

Q4. An interval is a collection of all real numbers between two given numbers. For instance the closed interval $[1.2, 3.7]$ includes all the numbers between 1.2 and 3.7, including 1.2 and 3.7.

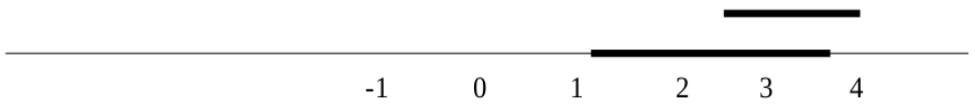


Write a program that prompts the user for four numbers where the first two are the end points of a closed interval and the next two numbers are the end points of another interval. The program prints `1` if the two intervals overlap, and `0` if the two intervals do not overlap.

For instance, the above interval $[1.2, 3.7]$ does not overlap with $[5.0, 6.0]$. Hence this is an execution of the program:

```
1.2 3.7 5.0 6.0
0
```

However, note that $[1.2, 3.7]$ overlaps with $[2.5, 4.0]$ since 3 is in both intervals.



Test 1.

```
1.2 2.3 3.4 4.5
0
```

Test 2.

```
3.4 4.5 1.2 2.3
0
```

Test 3.

```
1.2 3.4 3.4 10
1
```

Test 4.

```
1.2 3.4 3 10
1
```

Test 5.

```
1.2 3.4 0 10
1
```

Test 6.

```
3.4 10 1.2 3.4
1
```

Test 7.

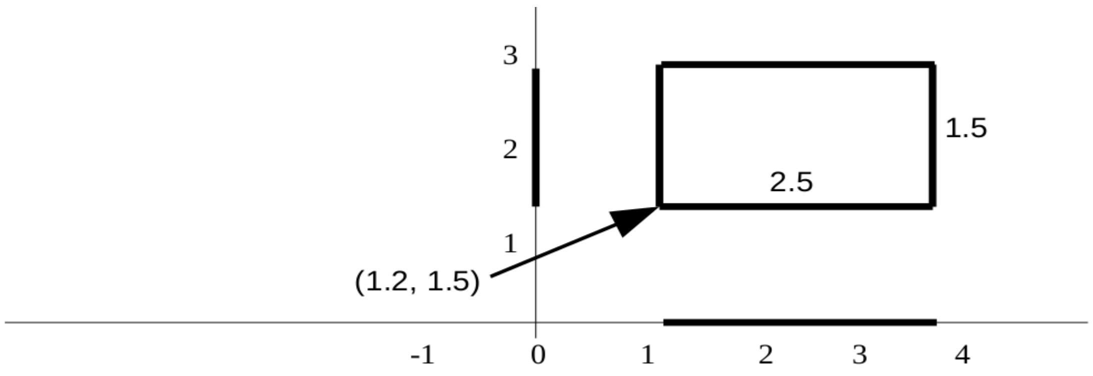
```
3 10 1.2 3.4
1
```

Test 8.

```
0 10 1.2 3.4
1
```

(This is very important for collision detection in computer games, at least for very simple games. You need a more powerful algorithm for high-end games.)

Q5. This is a continuation of the "overlapping interval problem" from the previous assignment. We now want to look at overlapping **rectangles**. A rectangle can be described by 4 numbers. For instance, look at the following rectangle:



The xy–coordinates of the lower left corner are 1.2 and 1.5. The width of the rectangle is 2.5 while the height is 1.5. We can therefore describe the above rectangle with these four numbers:

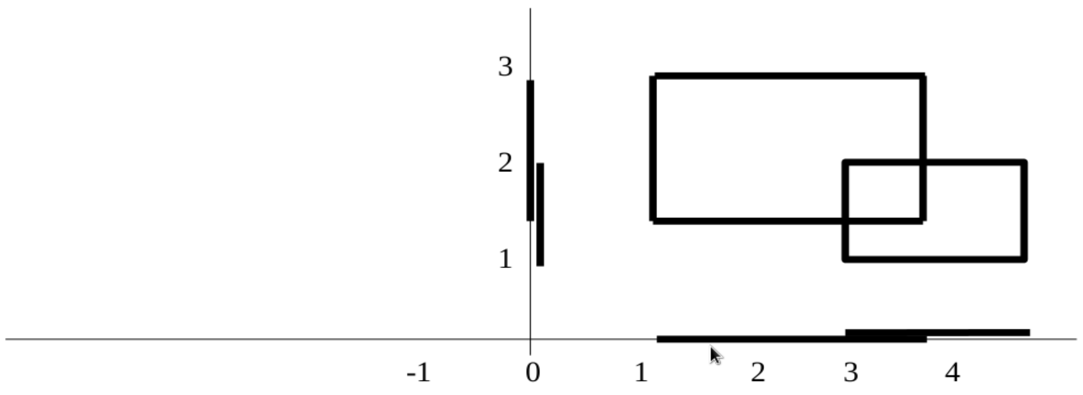
$$1.2 \quad 1.5 \quad 2.5 \quad 1.5$$

I have also drawn the "shadow" of the rectangle along the x– and y–axis. The math folks call them projections onto the axes.

The following diagram shows two rectangles. One rectangle is from the previous diagram. The other is described by

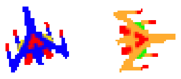
$$3 \quad 1 \quad 2 \quad 1$$

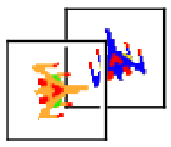
(i.e the xy–coordinates of the lower left corner are 3 and 1, the width is 2, and the height is 1):



Note that the two rectangles overlap. Of course there are other ways for two rectangles to overlap. For instance one can be "inside" another.

This is used in games for computing if the rectangular regions occupied by two images overlap. It gives an approximation of collision of two game objects:





Write a program that prompts the user for eight numbers (doubles) where the first four describe one rectangle and the next four describe another. Next the program sets a boolean variable to true if the rectangular regions represented by the numbers overlap; otherwise it's set to false. Finally your program prints the boolean variable; i.e., the output is `1` if the two intervals overlap, and `0` if the two intervals do not overlap.

Hence, this is an execution of the program for the above scenario with the two rectangles that overlap which will tell you that they overlap:

```
1.2 1.5 2.5 1.5 3 1 2 1
1
```

(You are encouraged to sketch some rectangles to better understand the problem.)

Test 1.

```
1.2 1.5 2.5 1.5 3 1 2 1
1
```

Test 2.

```
1.2 1.5 2.5 1.5 3 1 2 4
1
```

Test 3.

```
1.2 1.5 2.5 1.5 0 5 2 1
0
```

Test 4.

```
1.2 1.5 2.5 1.5 2 2 0.25 0.25
1
```

Test 5.

```
1.2 1.5 2.5 1.5 4 1 0.25 0.25
0
```

Test 6.

```
1.2 1.5 2.5 1.5 4 1 0.25 10
0
```

Test 7.

```
1.2 1.5 2.5 1.5 4 1 10 0.25
0
```

Test 8.

```
1.2 1.5 2.5 1.5 4 1 10 10
0
```

Test 9.

```
1.2 1.5 2.5 1.5 1 1 0.25 0.25
0
```

Q6. Write a program that shows the work for adding two 2–digit numbers using column addition. See the output of the test cases for the format of the output. Note in particular all the spaces and also the carryover (see Test 2 and Test 4).

Advice: Don't try to think of all the possible cases. Look at Test 1 and try to make your program work for Test 1. Then "grow"your solution to handle other cases.

Test 1.

```
12 34
  1 2
+ 3 4
-----
  4 6
-----
```

Test 2.

```
17 38
  1
  1 7
+ 3 8
-----
  5 5
-----
```

Test 3.

```
70 81
  7 0
+ 8 1
-----
1 5 1
-----
```

Test 4.

```
79 87
  1
  7 9
+ 8 7
-----
1 6 6
-----
```

After finishing this question, you should realize that you have just "taught"the computer how to do column addition – at least for two columns.

SPOILER ALERT ... HINTS ON THE NEXT PAGE ... USE THEM ONLY IF YOU NEED IT ...

Spoiler Hints

Read this only if you need to.

Remember: solve a problem in small steps. This looks like a complicated problem. So what "smaller" problem can you handle first? One possible "smaller" problem is the case where the column addition does not have any carryovers.

Using Test 1 as an example, first write a program that does this:

```
12 34
  1 2
+ 3 4
-----

-----
```

Next make it do this:

```
12 34
  1 2
+ 3 4
-----
    6
-----
```

The next step is to make it do this:

```
12 34
  1 2
+ 3 4
-----
  4 6
-----
```

The next step is to handle the carryover from the first column; see Test 2. The last step is to handle the carry from the second column.

If you look at all the numbers present in the output you will notice that there is a maximum of eight digits; see Test 4:

```
79 87
  1
  7 9
+ 8 7
-----
1 6 6
-----
```

You can do this problem if you know how to compute the values for these eight slots (use 8 variables):

```
79 87
  X
  X X
+ X X
-----
X X X
-----
```