COMP 112 - Lab 3

Due: 12:00pm (Noon), Monday November 10

Handin: place your `lab3.py` file in the folder `~/comp112/handin/lab3/`. Remember to issue the permissions command after you hand in the file:

```
setfacl -R -m u:rthorndy:rwx comp112/handin
setfacl -R -m m:rwx comp112/handin
```

For this lab you must provide a single python program that can perform all the calculations specified below, similar to the structure used in Lab 2. Your program will take a single command-line argument that tells the program which question is to be calculated, and then uses functions to produce the output.

For example, to execute Question 2, you would type:

```
$python lab3.py q2
```

You can use I/O redirection as well to make it easier to create test inputs, as was done in the previous two labs. For example, for Question 2, you need to provide 2 lists as input. This is tedious, so you can put the two lists in a file called "`q2input1.txt`", and send it to your program as input:

```
$python lab3.py q2 < q2input1.txt
```

Question 1 (execution code "`q1`")

Write an *iterative* version of the Fibonacci problem. Given an input value, `n`, calculate the n^{th} Fibonacci number, where:

```
fib(1) = 1
fib(2) = 1
fib(n) = fib(n-1) + fib(n-2), for n > 2
```

NOTE: this problem must be solved using loops, you can't use recursion!

Your internal function must take an integer as an input parameter and produce an integer as a return value; you are not to do any input or output within the function itself (as in Lab 2).

You can test your solution by comparing to the list published here:

http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fibonacci/fibtable.html

Question 2 (execution code "`q2`")

Implement the merge() function that is part of the MERGESORT algorithm. Recall that this function takes two (sorted) lists as input, and merges them into a single sorted list.

In the MERGESORT algorithm you always operate on a segment of the larger input list. For this implementation, though, you can just accept two separate lists as input, and produce a new list as a return value. Your function must not do any input or output, as in Lab 2.

Your function must run in `O(n)` time, where `n` is the combined size of the two input lists.

Note that you can get user input of lists by using the `input()` function, as in:

```
L1 = input()
L2 = input()
```

Then, the user types in a python-looking list. Similarly for output, you can just print the resulting list, and python will format it for you. Here is a sample run of the program:

```
$ python lab3.py q2
[5, 9, 11, 18, 45, 52]
[1, 3, 19, 62]
[1, 3, 5, 9, 11, 18, 19, 45, 52, 62]
```

Question 3 (execution code "`q3`")

Write an iterative function that outputs a triangle of asterisk characters whose height is given (as a positive integer) by the user. This triangle should be a right-angled triangle with the right angle at the bottom-left.

For example, if the user entered a height of 6, your function would output the following:

```
*
**
***
****
*****
******
```

NOTE: your solution must use iterative loops, not recursion! Also, you may ***not*** use "sequence multiplication"; that is, you can't print a line of asterisks with a command similar to:

```
print "*" * i
```

Unlike the Lab 2 specifications, this time your internal function *does* do the output, and consequently has no return value!! It must take the input (the height of the triangle) as a parameter, however, and not do the input itself.

Question 4 (execution code "q4")

Repeat Question 3, only this time you must use recursion, not iterative loops. (You are still prohibited from using sequence multiplication.)

HINT: ask yourself: "If I had a function that could print a right-angled triangle as described, but it had to be smaller than the one I have to draw now, how could I use that function to draw *my* triangle?"

Question 5 (execution code "q5")

Repeat Question 3, only this time the right angle needs to be at the bottom-right. For example, if the user entered a height of 6, your function would output the following:

```
     *
    **
   ***
  ****
 *****
******
```

Use space characters before the required number of asterisks on a given line to achieve this effect.

As in Question 3, your solution must use iterative loops and not recursion; your function produces the output and has no return value, and takes the height of the tree as an input parameter and performs no user input itself.

Question 6 (execution code "q6")

Repeat Question 5, but use recursion instead of iterative loops.

Note that this is a little more difficult that the left-leaning triangle from Question 4! In order for your recursion to work, you have to have a way of telling the recursive function how many space characters it needs to print before the asterisks.

Question 7 (execution code "q7")

Use Euclid's method for calculating the Greatest Common Divisor (GCD) of two positive integers.

The general algorithm looks like this:

- Input the two positive integers, X and Y
- Let R be the remainder when you divide X/Y (the "mod" function)
- Repeat until R == 0:
    - Set X = Y
    - Set Y = R
    - Recalculate R as the remainder when you divide X/Y
- Output the value of Y as the final answer

NOTE: you must use iterative loops to solve this problem; you can't use recursion.

Your function should take the two values as parameters, and return a single value that is the GCD of the two input values. Your function should have no console input or output.

Question 8 (execution code "q8")

Repeat Question 7, only use recursion instead of iterative loops. We can redefine GCD recursively as follows:

GCD(X, Y) is:

- Y, if X is a multiple of Y
- GCD(Y, X mod Y) otherwise