Introduction to Computer Science CMSC 105

Lab 2: Design By Cases; Preconditions and Postconditions

This lab is designed to challenge you to think about interesting "corner cases" as you develop algorithms and test suites. Pay close attention to exactly what is required in each question and do your best to think of different kinds of instances of each problem.

Pre-Lab Work:

What values or errors do the following Python expressions produce? Predict, and then check your answers.

- `wombat[0]`
- `'wombat'[0]`
- `len('wombat')`
- `len(wombat)`
- `'wombat'[6]`
- `'wombat'[5]`
- `'wombat'[2:5]`
- `'wombat'[:3]`
- `'wombat'[3:]`
- `'wombat'[6:]`
- `'wombat'[-1]`
- `animal='wombat'`
  `animal[0]`
- `'folding'[3:]+'wombat'[3:]`
- 'wombat'[::-1] # you don't need to predict this; type it in and see what it does

Lab Work:

You will need continue to do work in the `computational_geometry` project from Lab 1, and obtain the files for the `basic_recursive_design` project. To get a new project, switch to CVS Repository Exploring pespective, opening the HEAD repository, right-clicking on `basic_recursive_design` and choosing Check Out (if the project isn't there, ask your lab instructor; if you need more detail, see the on-line instructions "Using Eclipse to Obtain (and Submit) Lab Work")).

1. The "Fibonacci sequence" (described in a book by Leonardo Fibonacci in the year 1202) is a sequence of numbers starting with two 1's and in which each other number in the sequence is the sum of the previous two numbers. This sequence begins with the numbers 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ... and goes on forever. It has surprising relevance for natural objects and processes, such as mathematical models for the growth of animal populations, the shapes of certain shells, or the patterns of seeds in flowers (see `http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/fibnat.html`

for details and other examples).

In the file `fibonacci.py` in the `basic_recursive_design` project, provide a test suite and specification for a function named `fib` that is given a number n and **finds** the n^{th} element of the Fibonacci Sequence, i.e., `fib(6)` should be `8` (so the function will return an integer, **not** a True or False value).

a) Create a test suite at the top of the file `fibonacci.py`. Since we are not yet concerning ourselves with the speed of our algorithms, do not try to find any elements of the sequence past the 30^{th}; some algorithms for this problem are rather slow.

b) Provide a precondition for the `fib` function (use the `precondition` function if you can).

c) Provide a postcondition for the `fib` function (as a comment, unless you can think of a precise way to test the result in Python).

2. A palindrome is a sequence of letters that read the same forwards or backwards. For example, the words "kayak", "racecar", and "sees", and the phrases "A man, a plan, a canal: Panama" and "Lived on decaf, faced no devil.", are palindromes (thanks to `http://www.palindromelist.net`). The words "bluebus" and "bolton", and the phrase "This is not a palindrome", are not.

   In the file `palindrome.py` of the `basic_recursive_design` project, provide a test suite and a specification for a function `is_palindrome` that will be given a single Python String that is the sequence of letters to be tested and return True if the sequence is a palindrome, or False if it is not. Note that the function will be given only the letters, and all will be lower case, so the second phrase above would be presented as "livedondecaffacednodevil" rather than "Lived on decaf, faced no devil." (the user interface takes care of removing non-letters and converting to lower case).

   a) Add a test suite at the top of `palindrome.py`. If the informal description of the problem (given above) is not precise enough to let you figure out the answers to all of your tests, make reasonable assumptions about what to do, but document them with comments; if it is precise enough, add a comment saying so. (If you want to change this while working on next weeks lab, that is fine, so don't worry about making your life harder later.)

   b) Add a precondition for your function. You may make this a comment instead of using the `precondition` function from `logic.py`, if you like.

   c) Add a postcondition, using the `postcondition` function if possible (hint: use `[::-1]`).

3. Design and implement an algorithm to solve the circle-rectangle problem from Lab 1 in the file `circle_rectangle.py` of the `computational_geometry` project. Note that we will be using this program for several other activities in class, such as a "code review session", so the `circle_rectangle_ovelap` is designed to select among these activities. To write your function, you'll need to change the definition of `MODE` in that function, to switch from running the sample answers (with `MODE='test samples'`) to running your own algorithm (with `MODE='mine'`), and then write your algorithm in the `my_circle_rectangle_ovelap` function.

   To plan your algorithm, follow the design steps you used for other problems in Lab 1 — spend some time thinking about an algorithm and making notes before you write the Python function. If your thinking suggests any interesting new tests, add them to your test suite.

   For this problem (and the extra credit below), you may need to both identify different cases *and* think about the mathematics needed in each case. In this problem and the upcoming labs, it is also particularly important to follow the Appendix A and style guide rules for `if` statements:

every `if` and `if/elif` sequence should end with a final `else:`, and every section indented under `if`, `elif`, or `else` should end with a `return`.
Once you have written your function, test it with your test suite. If you aren't sure why the function fails one of your tests, take a copy of the function itself and write in all the parameter values, and then "play computer" to figure out what your function actually does. Fix any problems you find; if you think of interesting new tests as you work, add them to your test suite (you can switch `MODE` back to 'sample answers' to test the samples again, if you like).
Submit your work at each interesting point and when you are done.

4. **EXTRA CREDIT:** *(Do not work on this until you have completed the work above.)* Design and implement an algorithm to solve the `segment_ovelap` problem from Lab 1 in the file `segment.py` of the `computational_geometry` project.

We will return to the Fibonacci sequence and the palindrome testing problem in next week's lab.

Remember to use "Team→Commit" as you make progress and when you are done (but let us know, via the comment that you can enter as you commit, whether you have finished).