CSE 305 Programming Languages Spring, 2010
Homework 10
Maximum Points: 15
Due 10:30 AM, Friday, April 16, 2010

Professor Shapiro

April 9, 2010

Write the answers in a file named `hw10.txt`. Put your name and user name at the top of the file, and submit the file, using the UNIX command, `submit_cse305 hw10.txt`. You will be directed to submit 2 more files below.

Note: When you are asked to write a program, you are never to trivialize the task by using some library program or other predefined program which does the job for you. It is acceptable to research the problem on the Web, or in other resources, but if you do, you must cite the sources that you used. It is never acceptable to claim credit for work that is not your own.

1. (5) "Interval arithmetic is the arithmetic of quantities that lie within specified ranges (i.e., intervals) instead of having definite known values. Interval arithmetic can be especially useful when working with data that is subject to measurement errors or uncertainties." [Weisstein, Eric W. "Interval Arithmetic." From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/IntervalArithmetic.html`] An interval can be represented by a tuple of two numbers, `(a,b)`, which represents some number that can be as low as `a` and as high as `b`. If `(a,b)` represents the possible weight of one object, and `(c,d)` the weight of another, if you weighed them together, the total weight might be anything within the inteveral `(a+c, b+d)`. Similarly if a loaded truck weighs somewhere in the interval `(a,b)`, and empty it weighs somewhere in (c,d), it's load might have weighed anywhere in the interval `(a-d, b-c)`.

   (a) (3) Write two Python functions, `intSum(x,y)` and `intDiff(x,y)`, each of which takes two intervals as arguments, and returns the interval sum and the interval difference, respectively. Use assignment statements with multiple variables in the Left Hand Side to unpack the tuples, and multiple-value return statements to return the intervals.

   Put your two functions in a file called `intervals.py`, print your file here, and submit it.

   (b) (2) You can run a Python interactive shell by entering the Unix command `python`. Then, you can load your functions into the shell with the statement `import intervals`. (Assuming you are running in the same directory as the one containing the file.) If you edit your file, you can reload it with the function call `reload(intervals)`. After loading your file, you call your functions with the expressions `intervals.intSum(x,y)` and `intervals.intDiff(x,y)`. You can exit the Python interactive shell by entering `C-d`. Put a transcript of an interactive Python run in which you load and demonstrate your functions here.

Continued on next page.

2. (10) Copy the Ruby program `/projects/shapiro/CSE305/ProgramsForHomeworks/match.rb`. It contains one function, `variable?(symb)`, that takes one symbol as argument and returns `true` if that symbol is to be interpreted as a variable, and `false` otherwise.

   (a) (8) Add to your copy of `match.rb` a definition of the function `match`, which should take three arguments:

   - a pattern, represented as an array that might contain variables;
   - an array that doesn't contain variables;
   - an optional substitution, represented by a hash, that defaults to the empty hash, {}.

   Your `match` function should return `false` if the pattern does not match the constant array. If the pattern does match the constant array, your program should return a substitution that shows what constant element has been matched to each variable in the pattern. The substitution should be a hash with variables as keys and their matching constant elements as their values. The following table shows example calls to `match` in the left column, and the values that `match` should return in the right column.

```
match([], [])                         =>   {}
match([], [1])                        =>   false
match([1], [])                        =>   false
match([:a,2,:c], [:a,2,:c])           =>   {}
match([:a,2], [:a,2,:c])              =>   false
match([:a,2,:c], [:a,2])              =>   false
match([:a,:x,:c], [:a,2,:c])          =>   {:x=>2}
match([:a,:x,:c,:x], [:a,2,:c,2])     =>   {:x=>2}
match([:a,:x,:c,:x], [:a,2,:c,4])     =>   false
match([:a,:x,:c,:y], [:a,2,:c,4])     =>   {:x=>2, :y=>4}
```

   For full credit, write your `match` function recursively. You may write it iteratively, but then the maximum number of points you can earn will be 6.

   Print your version of the file `match.rb` here and submit it.

   (b) (2) You can run a Ruby interactive shell by entering the Unix command `irb`. Then, you can load your functions into the shell with the function call `load 'match.rb'`. (Assuming you are running in the same directory as the one containing the file.) If you edit your file, you can reload it with the same function call. After loading your file, you call your functions with the expressions shown in the above table. You can exit the Ruby interactive shell by entering `quit`. Place here a transcript of an interactive Ruby run in which you load your `match.rb`, and test it on all the calls shown in the above table.