211hw2

Homework 2 (Due Friday 1/29/10 at 10:00 am)

Submit via OWLSPACE in a single file that is runnable in DrScheme (so all expository answers must be enclosed in comment blocks or commented out using semi-colons.s into one file and submit this one compressed file.

The type "natural number" (**N**) in this assignment means the *natural-number* type defined in the text in Section 11.1. Your file should include a data definition for **N**. Unless the problem statement stipulates otherwise, the *only* built-in operations you may use with values of this type or variants (such as `natural>=1`) are the constructors, accessors, and recognizers for the type and the `equal?` (or =}) operation. For **N**, the constructor is `add1` ; the accessor is `sub1`, and the recognizers are `zero?` and `positive?`.
For variants, the constructor is typically `add1` (unless the variant consists of multiples of `m>1` [such as the even numbers]), the accessor is typically `sub1`, and the recognizers are typically `(equal? ... k)`, where `k` is the base number, and `(> ... k)`.

Problems from the book (HTDP) with some customization

- 11.2.4 (20 pts)
  - Copy the definition of `deep-list` from the text. Be sure to provide your own function template for `deep-list` and to write template instantiations for `depth` and `make-deep`.
- 11.4.7 (20 pts)
  - Include a data definition (following the text) of `natural>=1`. In addition to the constructors, accessor, recognizers, and `equal?`, you may use the library functions `remainder` and `*` . Hint: define an auxiliary function `is-divisible-by` of two inputs `p` and `q` (using the `remainder` library function) that determines if `p` is divisible by `q` (*i.e.*, p/q is a whole number).
  - Note that the problem as stated in the book has TWO parts; the second, writing `prime?` is easy after doing the first. Do not worry about optimizing the search for a divisor for `n` by bounding the search to numbers less or equal to `(integer-sqrt n)`; for simplicity, the `integer-sqrt` and `sqrt` library functions are forbidden in this exercise.
- 12.2.2 (20 pts)
- 12.4.2 (30 pts)
  - Do this problem followed by developing the function `arrangements` that returns a list containing all of the arrangements (permutations) of the input word. This function is described in detail in the text and the code for it is **given to you** in problem 12.4.1, but present this answer in your program file as if you developed it, including supporting test data.
  - The **Hint** for this problem should also state that

```
(insert-everywhere/in-all-words 'd (list (list 'e 'r) (list 'r 'e))) =
  (list (list 'd 'e 'r) (list 'e 'd 'r) (list 'e 'r 'd) (list 'd 'r 'e) (list 'r 'd 'e) (list 'r
'e 'd))
```

  - Notes: the function `arrangements` computes all of the **permutations** of the input word. Permutation is an important concept in basic probability theory. For some reason, the authors of the book chose to avoid using the relevant mathematical terminology. This problem includes writing the `arrangements` function because it is cool and developing `insert-everywhere/in-all-words` is the bulk of the work involved in developing `arrangements` .
- 13.0.5 (part 4 only) (5 pts)
- 13.0.8 (part 2 only) (5 pts)