CSE 341, Autumn 2006, Assignment 4
Miranda Part I
Due: Wed October 25, 10:00pm

15 points total (2 points each Questions 1-4, 7 points Question 5)

You can use up to 3 late days for this assignment. (The reason for this is so that we can post sample solutions within a reasonable time — if we wait for 6 days that's getting long.)

The first four questions are similar to those in the Scheme warmup, but they do also probe aspects of Miranda that are different from Scheme (such as static type declarations and infinite data structures).

For each top-level Miranda function you define, include a type declaration. For example, your `cone_volume` function for Question 1 should start with:

```
cone_volume:: num->num->num
```

You don't need to include types for helper functions — just the top-level functions. If you get stuck figuring out the correct type, you can write the function without a type declaration, let Miranda infer it, and then add the information to your script later. However, it will be good practice to try figuring it out yourself.

You should also include a set of easy-to-run test cases. For example, for Question 1, one of your tests could be:

```
cone_test1 = "cone_volume 10.0 2.0 = " ++ show (cone_volume 10.0 2.0)
```

Then define a `all_tests` function that returns a nicely-formatted string consisting of the results of running all the tests:

```
all_tests = lay [cone_test1, repeat_test1, repeat_test2]
```

1. Write a function `cone_volume` that takes two numbers representing the height and radius of the base of a cone, and that returns the volume of the cone. (To be precise, since in Miranda all functions are curried, `cone_volume` doesn't actually take two parameters; instead, it takes one parameter and returns a function that takes another parameter, which finally returns the volume.) Remember to include the type declaration.

2. Write a recursive function `cubes` that takes a list of numbers, and returns a list of the cubes of those numbers. For example, `cubes [1,3,10]` should evaluate to `[1,27,1000]`, while `cubes []` should evaluate to `[]`. Also try your function on an infinite list, for example `cubes [1..]` or `cubes [1,3..]`.

3. Write a function `repeat_n` that takes any value x and an integer n, and returns a list containing n occurrences of x. For example, `repeat_n "clam" 4` should return `["clam","clam","clam","clam"]`, and `repeat_n "clam" 0` should return `[]`. If n is negative, also return an empty list. (The Miranda library already includes a function named `repeat`, hence the different name. To answer a question that might come up, you can use any of the library functions in the definition of `repeat_n`, although you don't need to.)

4. Write a function to test whether a list of items is in strict ascending order. For example, `ascending [1,2,3]` should return `True`, while `ascending [2,3,1]` and `ascending [2,2]` should both return `False`. But wait! You should also be able to run this on a list of characters, strings, tuples, and

so forth. For example, `ascending ["squid", "octopus"]` should return false. You should handle the empty list, and a list of one item. What happens when you try `ascending` on an infinite list, for example `[1,3..]` or `[10,9..]`? What about a list of functions, for example `[sin,cos]`? Put your answer to the infinite list and list of functions questions in a comment. (You're still remembering to include the type declaration in your answer, right?)

5. Write a symbolic differentiation script in Miranda. You should have a function `deriv` that takes an expression and a variable, and returns the derivative of the expression with respect to the variable. You should handle all of the cases that the Scheme symbolic differentiation program handles, including the extensions you added in HW 2 (the difference operator, sin and cos, and raising an expression to an integer power). Support all of the simplifications that the Scheme program does.

   Miranda doesn't have the same program/data equivalence that Scheme does, so you'll need to decide how to represent the symbolic expressions. You can enter your test cases and print out the results using this representation (in other words, you don't need to write a parser).

   Hint: define a new user-defined type for symbolic expressions. Then, using this type, you can write expressions such as

   ```
   deriv (Sum (Var "x") (Const 5)) (Var "x")
   ```

   (In other words, what is the derivative of $x + 5$ with respect to x?)

   Develop your script in stages. First define the new type, and test it by typing in expressions such as

   ```
   Var "x"
   ```

   and

   ```
   Sum (Var "x") (Const 5)
   ```

   Once your sure this is working, go on to define the `deriv` function.

Extra Credit. (2 points max) Write a function `printstring` that takes a symbolic expression (as you defined for Question 5), and returns a string with that expression in standard infix notation. Include parentheses as needed to account for operator precedence, but don't put them in if they aren't necessary. For example, here would be some correct answers from `printstring`:

```
x
42
3*x+10
x*(y+10)
cos(2*x+0.5)
```

Turnin: Turn in your Miranda script, including your test cases. As usual, your program should be tastefully commented, and have well-chosen tests.