COMP 80 – Programming Languages – Spring 2003

Programming Project 3

Due date: Tue 3/4 11:00 pm

Introduction

In this project you will familiarize yourself with the ML language and interactive system, and write some functions to operate on various data types.
To start the interpreter, type sm1 at the unix prompt on any of the Sun machines in the department.

Basic Operations

Run the following lines of code in the same order as given here and copy the results into a text file, including the type. If an error occurs, write a few words on why it happened and how you can change the line to fix it.

1. 3 + 5;
2. it - 9;
3. it * (1.5);
4. val x = 8;
5. val y = 0;
6. x div y;
7. y <> x and also x = 8;
8. val y = "q"; (* what happened to the old y? *)
9. val s = "string of chars";
10. ("a", "b", 3);
11. fun area(radius) = 3.14159 * radius * radius; (* why does it say radius is a real? *)
12. area(4);
13. area(7.5);
14. if 5 > 2 then "a" else 7;

Writing ML Functions

Write and test the following functions. For each function provide the code plus a few runs (as in the class examples in starting.sml on the course web page).

1. Write a function cmpTimes that takes two arguments each denoting a time given in the format (hour, minute, AM or PM) of type (int * int * string). The function returns -1 if the first time is earlier, 1 if the second is earlier, and 0 if they are equal. Thus cmpTimes((2,15,"AM"),(1,16,"PM")) returns -1, cmpTimes((2,15,"AM"),(4,16,"AM")) returns -1, cmpTimes((5,15,"AM"),(4,16,"AM")) returns 1, and cmpTimes((5,15,"PM"),(5,15,"PM")) returns 0.
2. A function `mul` that simulates multiplication by addition. It takes a tuple `(x, i)` of type `real * int` and recursively adds `x` `i` times. Thus, `mul(3.5, 0)` evaluates to 0.0. `mul(3.5, 4)` evaluates to 14.0. You may assume that `i` is ≥ 0.

3. A function `sumIt` that takes one argument `n` of type `int` and sums the value of the polynomial `x^2 + 5x + 6` for values of `x` from 0 to `n`. Thus `sumIt(0)` returns 6 and `sumIt(3)` returns 68. You may assume that `n` is ≥ 0.

4. A function `power` that takes a tuple `(x, y, n)` of type `int * int * int` and returns `x^y mod n`. You may assume that `y` is ≥ 0. Thus, `power(3, 3, 10)` returns 7 (which is 27 mod 10) and `power(3, 4, 10)` returns 1 (which is 81 mod 10). You must make sure that you do not get an overflow during the computation. For example, `power(3, 5505, 10)` should run to completion and return 3. You should use a linear recursion, reducing the power by 1 in each recursive call.

5. A function `fastpower` with exactly the same specification as above but implemented using the repeated squaring technique. This is done as follows: if the power `y` is even, you first compute `x^y/2` and then square the result. If `y` is odd you do the same and then multiply by `x`. For example the recursive step for `power(3, 7, 10)` will compute the result as `(3^3)^2 * 3`. As before the computation is `mod n` and you must make sure that you do not get an overflow during the computation.

Experiment with `power` and `fastpower` to see for what values of arguments you start to observe a difference in performance.

**Submitting your program**

Your submission will include two files: a text file `pp3.txt` consisting of your results and answers to the first section of the project, and a ML file `pp3.ml` containing your programs for the second part.

Make sure that the file `pp3.ml` runs as it is in the system (that is, no errors should occur if we try `sml < pp3.ml`). Also make sure that your code is easily readable and well documented. The grade will be based both on correctness and on readability. Assuming your code is in the current directory on one of the local Sun machines, you should submit by typing `provide comp80 pp3 pp3.ml pp3.txt`. 

2