CS 558 Homework 1 – due 6:00pm, Wednesday, Oct. 8, 2003

Homework must be submitted by mail to cs558acc@cs.pdx.edu. All submitted files (four for this assignment) must be sent as plain-text attachments to the mail message (the contents of the message itself will be ignored). It is your responsibility to submit the homework in the proper format.

All programs mentioned can be downloaded from the course web page.

1. Getting Started (try this now!)

Consider the programs hw1.java and hw1.sml. Each writes the string “Hello World!” to standard output ten times. The Java version defines a stand-alone application (not applet) which can be compiled by typing javac hw1.java; this produces a class file Hello.class, which can be executed by typing java Hello. The ML version defines a function hi which can be loaded into the ML interactive loop by typing use "hw1.sml"; and executed by typing hi();

Extend each of these programs so that it takes two parameters, an integer c ≥ 0 and a string w, and writes Hello w! to standard output c times. Your extended Java program should be called sol1.java and should continue to define a class Hello with a main function; it should take the new parameters as arguments on the execution command line, so that one passes them by typing, e.g., java Hello 12 fred. Your extended ML program should be called sol1.sml and should continue to define a function hi; this new version of hi should take the new parameters as arguments, so that one passes them by typing, e.g., hi(12,"fred");

You may assume that your extended programs will be given only valid parameters, so there’s no need to perform error checking.

Submit (just) the files sol1.java and sol1.sml.

2. Expression Interpreter.

Consider a very simple language of expressions, whose abstract syntax is given by the following tree grammar:

```
Prog : Program → Exp
Add : Exp → Exp Exp
Sub : Exp → Exp Exp
Mul : Exp → Exp Exp
Div : Exp → Exp Exp
Int : Exp → (int)
```

To actually read and write programs in the language, we’ll use a LISP-style concrete syntax that maps directly to the abstract syntax, specified by the following grammar:
This problem asks you to modify interpreters for this language, written in Java and ML, living in files called hw1_2.java and hw1_2.sml respectively. These interpreters read a file containing a program in the LISP-style concrete syntax described above and evaluate the expression, producing an integer result. The Java version takes the filename to be read as an OS command-line argument, e.g., java Interp foo; the ML version takes it as a string argument to a top-level function in the interactive loop, e.g., Interp.interp "foo".

For example, if foo contains the following text:

```
(/ { Integer division rounds results down }
(+ 7
 (- 0 2)) { Here’s how to make a negative number }
3)
```

then both interpreters should return the answer 1. Note that programs can be broken arbitrarily across multiple lines. Any text within curly braces will be treated as a comment; comments may be nested.

Your task is to add certain features to each interpreter. Combine your solutions for parts (a) and (b) in a single Java file sol1_2.java. Combine your solutions to parts (c) and (d) in a single ML file sol1_2.sml. Submit (just) these two files.

(a) The Java version omits support for multiplication. Add it.

(b) The Java version just evaluates the expression; the ML version also prints it out in concrete syntax first, which enables us to verify that it was parsed correctly. Add an equivalent printing feature to the Java version, by implementing the method

```
public String toString()
```

for each Exp subclass (including the multiplication operator you added in part (a)). (Routines that print abstract syntax in human-readable format are usually called “pretty printers,” especially if they format their output nicely over multiple lines. The ML printing code I’ve provided does not produce particularly “pretty” output for long programs – it just dumps the whole program out on one line; your Java code should do the same.)

(c) (d)
(c) The ML version does not evaluate the program AST directly; instead it compiles it into a sequence of stack machine instructions, and then executes the resulting stack machine program. Rather bizarrely, the stack machine operates on reals rather than integers, but any stack machine program produced by the compile function should generate an integral result.

Note that the interpreter currently prints the stack machine program before executing it. Modify the ML code so that instead it traces machine instructions and resulting stack states while the program executes. The contents of the stack should be printed left-to-right, with the current top-of-stack first. For example, the execution of program foo above should produce the following output:

```
Expression: (/ (+ 7 (- 0 2)) 3)
Stack machine trace:
  CONST 7.0 : 7.0
  CONST 0.0 : 0.0 7.0
  CONST 2.0 : 2.0 0.0 7.0
  NEGATE : ~2.0 0.0 7.0
  PLUS : ~2.0 7.0
  PLUS : 5.0
  CONST 3.0 : 3.0 5.0
  PLUS : 5.0
  RECIP : 0.333333333333 5.0
  TIMES : 1.666666666667
  ROUND_DOWN : 1.0
Evaluates to: 1
```

Note that ML uses the tilde character (~) in place of the usual unary negation symbol (-), both in source programs and in the standard library routines for parsing and printing numbers. Don’t worry about this.

(Hint: The easiest approach is to print the trace directly as a side-effect of the exec function. A more elegant, but somewhat more challenging, approach is to generate the trace as an additional string result of exec and print it from within the interp function. In any case, you’ll want to write an auxiliary recursive function to dump the contents of the machine stack.)

(d) The ML version and the Java version produce different final results on some programs involving division operations. (I’m not talking about division by 0 here, nor about the use of the ~ symbol mentioned above.) Find out what the difference is, and fix the ML version to behave the same way as the Java version.

(Hint: The problem manifests itself on quite small numbers, and can be fixed by changing just one library call.)