COSC 341/342 Programming Project #2

Interpreter for variable definitions, function definitions, and addition operations

Distributed: February 18, 2010 **Due**: March 16, 2010

Synopsis

Write an interpreter that can implement the following BNF for a very simple language called M.

BNF for M

NT = { <statement>, <funDef>, <varDef>, <expression>, <id>, <operand>, <addop>, <integer>}

T = { **;** , **val** , **=** , **+** , **fun** }

S = <statement>

P =

```
<statement>   ::=   <funDef> ;          |     <varDef> ;   |
                    <expression>;       |

<varDef>      ::=   val <id> = <expression>

<expression>  ::=   <operand> <addop> <operand>          |
                    <operand>                            |
                    <id> ( <operand> )

<operand>     ::=   <integer>     |     <id>

<addop>       ::=   +

<funDef>      ::=   fun <id>  ( <id> ) = <expression>
```

<id> is any legal identifier.

Symbol table:

To simplify your life by simplifying the interpreter's environment, there is exactly one environment. Any redeclaration of variable or function will replace its value in the symbol table.

An identifier can be both a function definition and a variable definition (at the same time, … two entries in the symbol table). Which one you use will be determined by the statement being interpreted.

The symbol table should include at least the following attributes:
`name, type (function or variable), value`

The value of a variable is obvious.

The value of a function is the string representation of the function definition. You may format the representation in any way you wish. When the user inputs a statement that is a function call, then your interpreter will have to access the function "value" and execute it.

The data structure for the symbol table is your choice: array of objects, linked list of objects, hash table, … The use of parallel arrays is extremely inelegant.

Extension to Symbol Table (optional):
You can extend the symbol table to something closer to ML by treating it as a stack for inserting definitions. Each new variable definition or function definition causes a new entry to the top of the stack. When a variable or function is referenced, then search the symbol table from the top until you find what you want.

Execution:

The interpreter outputs a prompt.
The user enters input on a single line followed by carriage return.
The interpreter executes the statement.

If there is an error, output an error statement. Some extra
To check the correctness of the interpreter, you should include the option to output the value of the symbol table after each statement is executed. This can be done via run-time option, or via user input when starting the interpreter.

Test suite
Write down everything your interpreter should do correctly and give a set of statements that test that.
You must have at least 10 test cases. Here are a couple to get you started:
(1) variable definition correctly entered and retrieved from symbol table
val x = 3;
x;
val x = 5;
x;

(2) two variable definitions correctly entered and retrieved from symbol table
val x = 3;

val y = 5;
y;
x;

Check your program (an example test run)

```
> val x = 3;
3
> val  y = 3 + 2;
5
> x;
3
>x+y;
8
>fun inc(x) = x+1;
"fun inc(x) = x+1"
>inc(3);
4
>inc(y);
6
```

Turn in:
Hardcopy
Sample runs with symbol table output that cover the functionality
Also, schedule a code demo and walk through.

Grade based on:

Meets specs:	70%
Coding style:	10%
Elegance:	10%
Test cases:	10%