CPSC 411 Compiler construction I
Author: Robin Cockett
Date: 9 Jan. 2002

Assignment #1: Compiler for Minusculus (see link for grammar with left recursions removed)
Due date: 29th Jan 2002 (modified from 28 January 2002 to equalize labs.)

Implement a compiler to "stack machine code" (described below) for the language Minusculus whose syntax is defined by the grammar below.
In this assignment you may use LEX (in fact you MUST use LEX) but please write a recursive descent parser (i.e. you cannot use YACC yet!). It is important to attend the labs as they will be discussing this assignment and will also introduce LEX.

Minisculus grammar:

```
prog -> stmt.
stmt -> IF expr THEN stmt ELSE stmt
            | WHILE expr DO stmt
            | DO stmt UNTIL expr | READ ID
            | ID ASSIGN expr
            | PRINT expr
            | BEGIN stmtlist END.
stmtlist -> stmtlist stmt SEMICOLON
            |.
expr -> expr addop term
            | term.
addop -> ADD
            | SUB.
term -> term mulop factor
            | factor.
mulop -> MUL
            | DIV.
factor -> LPAR expr RPAR
            | ID
            | NUM
            | SUB NUM.
```

This grammar has left recursive nonterminals. The grammar with the left recursions removed automatically is here . Notice this file calculates the first sets and follow sets of the above grammar and then transforms the grammar and introduces new nonterminals in this process.
Where the tokens above stand for:

```
"if" => IF
"then" => THEN
"while" => WHILE
"do" => DO
"until" => UNTIL
"read" => READ
"else" => ELSE
"begin" => BEGIN
"end" => END
"print" => PRINT
{alpha}[{digit}{alpha}]* => ID (identifier)
{digit}+ => NUM (positive integer)
"+" => ADD
"-" => SUB
"*" => MUL
"/" => DIV
```

```
  "(" => LPAR
  ")" => RPAR
  ";"=> SEMICOLON
```

```
Minisculus comments:
Minusculus has two types of comments:
  * multi-line comments:        /* comment */
  * and one line comments:      % comment
```

EXAMPLES:

Minsiculus program:
Here is an example program:

```
  /* This program calculates the factorial of the number input */
  begin % input a number ..

  read x;
   y:= 1;
   while x do
  begin

  y:= y * x;
  x:= x - 1;

  end;
  print y;

  end
```

Code generation:
Typical minusculus programs fragments are:

```
begin

  y:= 23;
  x: 13 + y;
  print x;

end

begin

  if y then x:= 10
         else x:= 1;
  z:= z * x

end
```

(where we assume here that the variables x, y, and z must have been initialized earlier: Note that for conditionals and while statements zero is false anything else is true). These fragments are translated into a stack machine code:

```
  cPUSH 23
  LOAD y
  cPUSH 13
  rPUSH y
  OP2 +
  LOAD x
  rPUSH x
```

```
  PRINT

  rPUSH y
  cJUMP   L1
  cPUSH 10
  LOAD x
  JUMP   L2

L1:

  cPUSH 1
  LOAD x

L2:

  rPUSH z
  rPUSH x
  OP2 *
  LOAD z
```

where

* cPUSH k --- push constant k onto stack
* rPUSH r --- push contents of register r onto stack
* sPUSH --- replaces the top element of the stack by the element it indexes in the stack
* LOAD r --- pop the top of the stack and put the value in register r
* OPn?? --- perform the operation on the top n values of the stack replacing them by the result
* cJUMP L --- conditional goto L (a label) pops top of stack and if it is zero (false) it jumps to label
* JUMP L --- unconditional jump to label
* PRINT --- pops and prints the top element of the stack
* READ r --- reads a value into register r (actually it reads a line and uses the first value on the line ...)

Here is an implementation of this stack machine in the shell: source this then source the file your Minusculus compiler produces!

```
  # ... aliases for Robin's stack machine.
  # This file should be "sourced" prior to executing
  # stack machine files.
  set stack = ""
  alias cPUSH            'set stack = (\!:1 $stack)'
  alias rPUSH            'set stack = ($\!:1 $stack)'
  alias sPUSH            '@ stack[1] = $stack[1] + 1 ; set stack[1] =
  $stack[$stack[1]]'
  alias LOAD            'eval "set \!:1 = \$stack[1] ; shift stack"'
  alias OP2               'eval "@ stack[2] = \$stack[2] \!:1
  \$stack[1]"; shift stack'
  alias cJUMP           'set tos = $stack[1]; shift stack; if ($tos ==
  0) goto \!:1'
  alias JUMP             goto
  alias PRINT            'echo $stack[1]; shift stack'
  alias READ            'eval "set \!:1 = $< " '
```
