CSE 3302/5307 Lab Assignment 3

Due August 5, 2013

Goal:

Understanding of Pascal, compiler/interpreter concepts, and subscripting.

Requirements:

1. Modify the solution to lab 1 from Spring 2013 to add simple integer arrays to PL/0:

   a. A variable name (in a `var` list at any scope) may be followed by `{` *lowBound* `:` *upperBound* `}` to declare an integer array. A bound may be either a number or a previously declared constant. *lowBound* may not be larger than *upperBound* and should be checked at compile-time.
   b. The stack-based PL/0 interpreter will need new instructions for loading from or storing to an array. Bound checks are to be included as run-time checks.
   c. A subscripting expression *arrayVariableName* `{` *expression* `}` may appear as the l-value for an assignment or as an r-value for an expression. Nested subscripts are legal.

2. Email your program to `yeqing.li@mavs.uta.edu` by 12:45 p.m. on August 5, 2013

Getting Started:

1. The solution for lab 1 is available at `http://ranger.uta.edu/~weems/NOTES3302/LAB1SPR13/plzero.io.pas`

   That assignment appears as the second page of this handout.

2. You may assume your input is a syntactically-correct PL/0 program, but some error-handling will reduce your debugging time.

3. A few details ordered by their relevance to the PL/0 environment source code:

   a. `symbol` must be extended for dealing with `{`, `}`, and `:` as tokens.
   b. `table` must accomodate a `kind` for integer arrays.
   c. `getsym` must be modified for `:` as a token.
   d. `enter` must be modified for saving details of an array.
   e. `vardeclaration` must be modified to parse and check an array declaration, along with saving semantic information in `table`.
   f. `factor` will parse and compile a subscripting expression as an r-value.
   g. `statement` will parse and compile a subscripting expression as an l-value for an assignment statement. For `a{` *expr1* `}` `:=` *expr2*, *expr1* should be processed before *expr2* (which is also natural for a recusive-descent compiler). Thus, at run-time, the result of the subscript calculation *expr1* will be on the stack while *expr2* is evaluated.
   h. Implementing simple array bound checks needs the bounds, but `table` is transient and not available at run-time. For a procedure creating arrays within its scope, it is convenient to save the two bounds just before the allocated array in the stack. The `lit` and `sto` instructions are useful for generating this code in `block`.
   i. `interpret` will implement new instructions for accessing arrays. Be sure to understand `lod` and `sto` first.

4. Sample PL/0 programs are on the webpage.

CSE 3302 Lab Assignment 1

Due XXXXXXXXXXXXXXX

Goal:

Understanding of Pascal and elementary compiler/interpreter concepts.

Requirements:

1. Add simple I/O to PL/0 (`plzero.pas`) as an input stream (`in`) and an output stream (`out`).
   a. An integer may be read from the input stream by using `in` as an r-value.
   b. An integer may be written to the output stream by using `out` as an l-value.
   c. The actual input stream will be as integers, one per line. `-999999` will end the stream and will be supplied to the PL/0 program once. Attempting to access the input stream after `-999999` will abort. "`?`" should be used as a prompt.
   d. The output stream will be integers written one per line. Each output line should begin with "`!`".
   e. The output stream does not terminate.
   f. "`in`" and "`out`" are identifiers for the streams. These are not reserved words and may be "masked" by other declarations.
2. Email your program

Getting Started:

1. It is convenient to provide a source program and input to your compiled PL/0 interpreter by:

```
cat add.pl0 - | plzero.io
    0 var                                   start pl/0
    1   x,y;                               ? 1
    1                                      ? 1
    1 begin                                ! 2
    2   x:=in;                             ? 5
    4   while x#-999999 do                 ? 6
    9   begin                              ! 11
    9     y:=in;                           ? 11111
   11     if y#-999999 then                ? 12345
   15     begin                            ! 23456
   16       out:=x+y;                      ? -123
   20       x:=in                          ? 123
   21     end;                             ! 0
   22     if y=-999999 then                ? -999999
   26       x:=-999999                      end pl/0
   28   end
   30 end.
```

2. A few small corrections have been made to Wirth's PL/0 code, including using `longints`. The source is at:

   `http://ranger.uta.edu/~weems/NOTES3302/NEWNOTES/NOTES02/plzero.pas`

3. You may reuse error numbers. For example, error 12 indicates "Assignment to constant or procedure is not allowed" and would also apply to "`in := ...`".