CS 301 Homework 2 – due 3:30pm, Wednesday, February 3, 1999

Lexical Analysis

Write a lexical analyzer for the full **PCAT** language, and a driver to test it. The lexical structure is described in Section 2 of the **PCAT** *Programming Language Reference Manual.*

The lexical analyzer will consume source text from standard input and produce tokens (with attributes where appropriate), one token per call to the analyzer routine. It will also keep track of the current line number. If any portion of the input text cannot be converted to a legal token, an appropriate informative error message should be written to standard error, and the analyzer should halt.

The driver will repeatedly invoke the analyzer, obtaining one token at a time, and echo the results to standard output, one line per token. Each line should have the following contents: the input line number followed by a colon, the token name (see below), and the token attribute value (if any). There should be a tab between the colon and the token name, and another between the token name and the attribute value (if it occurs). Token names are as follows:

- `ID` for identifiers.
- `INTEGER` for integers.
- `REAL` for reals.
- `STRING` for strings.
- Keywords token names are simply the keywords themselves.
- Single-character token names are simply the ASCII codes for the characters themselves.
- `ASGN` for `:=`
- `LEQ` for `<=`
- `GEQ` for `>=`
- `NEQ` for `<>`
- `LBRANG` for `[<`
- `RBRANG` for `>]`

The following token types have an associated attribute: `ID` (the identifier), `INTEGER` (the integer value), `REAL` (the lexeme matching the real pattern), and `STRING` (the string, in double-quotes (`""`)).

A working solution to the assignment is in `/u/cs301acc/2/lexer`. Your program should generate the same output as this one, except that errors (sent to standard error) may be different in format (though not in substance).

For example, the input stream

```
WRITE (4, "= 2+2 =", (* newline here! *)
                  4.00);
```

should produce the output

```
1:      WRITE
1:      (
1:      INTEGER 4
1:      ,
1:      STRING  " = 2 + 2 = "
1:      ,
2:      REAL    4.00
2:      )
2:      ;
```

Implementation and Assignment Submission

You should structure your program into two parts: the lexical analyzer itself and the driver program. The lexical analyzer should export a function that, when called, returns a single token name, perhaps after storing an associated attribute into a global variable. It will also be convenient to store the current line number in a global variable. The driver module should repeatedly invoke the analyzer and print the names of the resulting tokens (and attributes if appropriate) on standard output.

You are encouraged (but not required) to use the lexical analyzer generator tool `flex` (or it's older brother `lex`) to produce the analyzer. You can obtain access to `flex` on the CS Solaris systems by running `addpkg gnu`. Full documentation for `flex` can be obtained via `man flex` and `man flexdoc`.

If you use `flex` or `lex`, it is natural to use the names `yylex`, `yylval` and `yylineno` for your function, attribute variable, and line number variable, respectively. To make it easier to integrate your analyzer into subsequent assignments, you should use the token names defined above as symbolic enumeration values within the `flex` actions and in your driver. As usual, single character tokens can be represented by the character's ASCII code. You'll need to treat `BEGIN` separately, because it already has a specialized meaning within `flex`.

Prepare a `makefile` that builds your lexical analyzer and driver given your source files and leaves the executable program in `lexer`. Submit your program by `mail`ing a `shar` "bundle" containing the relevant files to `cs301acc`, as described in the "Handing In Assignments" handout.