Project 2 CIS 310 Kutztown

Points: 25
Due: TBD. No late submissions accepted.
Purpose: Lexicographical Analysis of a program, written in Ada

Assignment: In this assignment, you will scan an input file that represents a program that follows a simplistic programming language grammar, producing an output file that is an ordered list of the symbol types found in the input file.

The basis of this project, and the one that will follow it, is the small programming language grammar defined in MiniGrammar.pdf, available in class, and on the course website. We will, in class, make a list of all terminal symbols in this language.

Your program will get an input file from the command line. If the file is opened successfully, it will scan the program, classifying each symbol, and then writing that symbol to a file. The boxes at right show an input file, and the beginning of the output your program would produce from it.

```
program abc
dec
i1,i2:int;
x,y:real
begin
     Read(i2,x);
     y=i2*x+2;
     Write(y)
end.
```

```
progsym
id
decsym
id
comma
id
colon
type
semicolon
id
comma
id
colon
type
beginsym
Readsym
lParen
id
comma
...
```

To complete this project, you will need to devise strategies to identify each symbol of the language. You may read the input character by character (recommended), or line by line.

Declare an enumerated type, with one element named for each terminal symbol. Declare an instance of Enumeration_IO, and then each symbol's element name can be output once it is determined which symbol it is.

To assist you with many of the Ada concepts involved, an example has been prepared to demonstrate file I/O, enumeration types, and other important concepts. Named **ASCIIperCmdLine.adb**, it is available under the Ada examples subdirectory on acad and the web.

The example reads characters. It is recommended to build a string as identifiers are read (how do you know you are in the process of reading an identifier?), so the string can be identified as a keyword or identifier. Note that all keywords contain no digits.

Notes:

- You may assume that any input file will only contain terminal symbols.
- The file produced in this project will be the input to Project 3
- ASCIIperCmdLine is a short example for demonstration purposes only. Your submission must be broken into appropriate subprograms. For example, inputting the next token MUST occur in a subprogram that is passed the data file, and a string to hold the token. Identifying the token classification is also at least one subprogram, maybe several.
- You may assume that the last character in any test input file will be a carriage return
- Use the Ada.Command_Line package to access the command line arguments (as in the example).
- Documentation must be robust and compatible with the rules for use with Doxygen tool, which is well-documented on the instructor's links page. You are required to create a Doxygen site in your web area on acad.

<u>Turnin:</u>

Turn in your project using the turnin script. Your file is to be named **lex.adb** (all lower case). You are encouraged to submit any input files used to test your program. Also, submit a readme (named readme.txt) that contains a link to your Doxygen area. There will be a 2 point penalty for non-compliance on naming files.