Project 3 CIS 310 Kutztown

Points: 40
(+10 HW Points for left recursion worksheet; graded separately)
Due: TBD. No late submissions accepted.
Purpose: Scan and Parse of a program, written in Ada

Assignment: In this assignment, you will write code to process the output from Project 2, producing a program that performs a rudimentary scan and parse of a program, checking its adherence to the syntax rules set out in the Mini-Grammar after All Kleene stars are removed without application of left recursion..

Your program will obtain the name of and input a test program, from the command line if present, otherwise interactively, scan it using Project 2, which you will update to become procedure *lex*, and parse it, using the recursive descent method. The parser will analyze the lexemes produced by the scanner. See notes regarding storing the lexemes.

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

The screen output of the parse will be a series of messages indicating any errors found, and a final message indicating the result: success, or a final error message, indicating that the parser encountered one or more syntax errors. The number of errors found will also be displayed.

For example, the test file in the box should first produce an error message along the lines of:

Error: Expected Variable Name

We need to see why that is. But first, recursive descent must be explained. To implement it, simply write a function for each non-terminal. For example, the <dec part> rule is now:

```
<dec part> ---> <list var> : <type> | <list var> : <type> ; <dec part>
```

Thus, the function for <dec part> would work as in the box:

1. call function for <list var>
2. accept colon (:)
3. call function for <type>
4. if next item is semicolon
   A. call <dec part>

Now, back to the error. The problem is the semicolon before the *begin*. According to the rule for the declaration part of a program, a declaration is either <list var> : <type> or <list var> : <type> followed by a semicolon and another <dec part>. When the parser saw the semicolon, it would recursively call the dec_part function (at this point it is executing inside the dec_part function) to absorb another <list var> : <type> .It calls the list_var function, which (checking its rule) expects an identifier and instead encountered a begin symbol, which causes it to output the error.

Notes:

- Convert your second project into a package named *lex* that has Project 2's main() as its scanning function..
- The output from the *lex* subprogram will not be accessed from a file by the parser. You must use an object within another package to hold this info and make it available to the parser after *lex* fills it.

- Your program will handle several command line switches, that follow the required command line argument. They must be recognized case insensitively.
  - **/L** output lexemes to the file whose name follows the /L (separated by a blank).
  - **/S** output the symbol table to the file whose name follows the /S (separated by a blank).
  - **/E** echo the file as it is processed.

  The switches follow the required single command line argument, the input file name, and can be in any order.
- All errors are printed with line number of occurrence.
- Your symbol table will record the line number of each identifier. If an identifier is declared twice, give the line on which it first occurred when reporting the error, and don't record the 2nd occurrence.
- The errors you are required to report via a symbol table lookup upon encountering an identifier are (discovered above the *begin*) identifier declared twice, (discovered below the *begin*) identifier not declared, and wrong number of argument(s) to Read() or Write(). Also report the line numbers of BAD lexemes.
- If an error is found, try to act accordingly. For example, if it is a bad identifier in a dec section, see if a comma or right parent follows; regardless, try to recover seamlessly.
- You must have separate packages containing objects for the symbol table and for the list of lexemes, as well as the package for the scanner that was Project 2. The symbol table and holder of lexemes are like C++ classes; they are typedefs that should have associated functions.
- Your program will stop when it encounters the **end.** token. This must be true of any input file, whether or not it has anything after **end.** , even if it is the wrong format (e.g. DOS or Unix/Linux)
- Create a robust RoboDoc web area and provide the link in your readme file, to be named **readme.txt**, turned in on acad and submitted to the Project 3 Artifacts dropbox on D2L. Also, submit in the dropbox:
  - Describe:
    - Your parser's design, including major implementation decisions.
    - Your symbol table's construction and use.
    - For the non-terminals in the grammar numbered 1, 2, 3, 6, 11, and 12, provide an artifact describing how it was processed, via a flowchart (preferred) or in some other clear manner.
  - A bug report, if applicable. If your program has bugs not noted therein, the penalties will be heftier.
  - **A clear statement describing how to build and run the program**
- Submit at least two input files via turnin that were error-free, and two that had errors.
- Obvious negligence regarding testing will be severely penalized.
- A forum has been created for discussion of issues in development of Project 3. You must register for the forum; participation will be factored into the homework part of your grade, as well as borderline grading decisions. Further information will be provided under separate cover.

Deliverables:

- Worksheet – Due: 12 Noon Sat. April 4 in the Left Recursion dropbox.
- Project – Due: 10 AM Fri. April 24.
  - Turnin: Your parser, named **parse.adb** (all lower case). There will be a 3-point penalty for non-compliance. You will have at least six other files for the required packages, lex, lexemes, and the symbol table. Submit the input files used to test your program as well. Submit a textual readme.txt. DO NOT submit files produced by gnatmake, including executables.
  - D2L Project 3 Artifacts Dropbox: Full readme, design artifacts