CSS 448 Phase 2, Parser Due Wednesday, Apr 20

From project description: "You create a Backus-Normal (Naur) Form (BNF) grammar for the language so that yacc (yet another compiler compiler) can generate a parse table for the language. The parsing phase groups together tokens into meaningful units to determine if they have the right form for the language."

<u>You will do several things in this phase:</u>
1. Take the grammar given and create a yacc file. You must take the grammar and write it in BNF form with proper yacc syntax so yacc compiles successfully. This yacc file creates a parse table (y.tab.c file) which has no conflicts. Often this means rewriting various rules in the grammar to resolve conflicts. Some conflicts are not possible to remove, but usually yacc defaults to what you want. Sometimes new non-terminals are needed.

To aid you in this phase, I took the grammar found in the Lexical Conventions and Partial Grammar for Pascal handout and did some editing so it has correct yacc syntax. You still have some work to do, but no straight editing and minimal grammar writing. This file, linked off the website, is called y. After you finish the grammar writing, you will likely get one shift/reduce error when you first yacc the file. The default way yacc handles the conflict typically works correctly.

Note that the list of tokens in the y file is the same as in tokenconsts.h. If you added any tokens, e.g., yreal, you must add it to the list in the yacc file.

2. Create a yylex() function with no parameters. The yylex function you created in phase 1 is used by yacc. It must take no parameters. If necessary, rewrite what you must so that y.tab.c can call yylex(). You probably did not pass parameters, but if you passed parameters, you will want to use global variables. In this phase, you will use cin. In MSVS, there is a way to use standard input (but I don't remember how). It is under some tab. In some box, you place the name of the input file.

3. Create a makefile to compile and link together your lexical analyzer (the yylex() function) and y.tab.c. Test your program using the pascal examples on the website. (See *"Project makefile"* linked off the Yacc page.) At this point, nothing will happen other than they will (should) successfully parse (no syntax error message produced). Comments are displayed from your lexical analyzer, but no other output. If valid Pascal programs do not parse correctly, check your grammar and lexical analyzer.

4. When all the pascal programs successfully parse, add actions to display all the identifiers. Incorporate actions in the yacc grammar file so that any identifier found by the lexical analyzer is displayed (the actual string is displayed, not "yident"). Print exactly one blank after each identifier. At this point, do not print any new lines. (This printing is done by the parser, not the lexical analyzer.)

5. Adjust your lexical analyzer so that when it encounters a new line, it displays a new line (an end of line character)**.**

6. Test your program using the pascal examples on the website. All the valid pascal programs should parse correctly. Output will be all the identifiers and comments with the new lines shown. (The lexical analyzer is displaying new lines and comments; parser displays identifiers.)