README for the parser project

Wuu Yang
National Chiao-Tung University, Taiwan, Republic of China
July 28, 2022

For the second part of the compiler project, you will need to implement a parser. The syntax of our target langauge *minipascal* is given in the file 01-minipascal-spec.pdf.

You will use the yacc (or bison) tool. You need to prepare for the input for the yacc tool. A sample input file—standard-pascal.y, which is the syntax for the *complete* Pascal language—is provided.

For the parser project, you need to turn off the output of the scanner. You can also assume there is no scanner error in the input file. The parser will produce its own output.

If the input contains no syntactic errors, the output of the parser will be a single line containing just two characters: OK. Otherwise, the parser will issue error messagesuch as

```
line 23:  error token ident
```

Your parser should attempt to do error recovery and parse as much as possible of the input.

Tools and ParserTestCases

In the parser project, you HAVE TO use yacc/bison. Do not use hand-writtten parser otherwise you will get into great trouble later.

The file "standard-pascal.y" contains the complete grammar for Standard Pascal. For the second project (parser), you need to prepare your own yacc input, say minipas.y, using the minipascal grammar given in 01-minipascal-spec.pdf. You can try to run yacc on this grammar:

```
yacc minipas.y
```

Two files should be generated: y.tab.c and y.tab.h. Then you may combine the scanner (generated by lex) with the parser (generated by yacc). See the file "Communication between Lex and Yacc.pdf" for combing lex and yacc.

Then you use a C compiler to compile y.tab.c and other files generated by yacc and lex. You will obtain an executable file. The executable file is the parser. You use that parser to compile minipascal test programs. There is a collection of minipascal test prorams located in the ParserTestCases directory. You may use them to try your parser.

Lex and flex are equivalent. You can use either of them. Yacc and bison are also equivalent. You can use either of them.

If you still have problems, please come to discuss with me. Please do not leave the project blank.

Difficult Issues

Combining your parser (y.tab.c, generated by yacc) and the scanner (lex.yy.c, generated by lex in Project 1) is a difficulty. You can read the document "Communication between Lex and Yacc.pdf" or "Lex and YACC primer_HOWTO.pdf" for combining the parser and the scanner. Specifically, the numbers representing the terminals (i.e., tokens) must be the same in lex.yy.c and in y.tab.c because the scanner returns the numbers of the terminals (also called tokens in the scanner) to the parser. The parser uses the numbers of terminals for parsing.

Handin

For the 2nd (parser) project, you need to turn in your lex files, yacc files, semantic routine files (if you have semantic routine files), test cases, and the

executable code. You need to prepare for Makefile. The TA will run your parser with

```
make run
```

You also need to write a readme.txt file for the project, telling the TA how to run your program and additional details.

Note that you usually do not need semantic routines in this parser project. It is fine if you do not have semantic routines.

Put all of the above files in a single zip file which will be named "DDDDDDD-parser.zip", where DDDDDDD is your student id. Upload the zip to the new e3 platform.

Deadline of the 2nd project is November 8, 2022, Tuesday, 23:55 pm.