Semester Project – Part 2 – CSE244 – Spring 2001

Due: Thursday, April 12, 2001, 12:30pm

April 1, 2001

The second part of the project for the Spring 2001 Semester will focus on developing a lexical analyzer/parser using lex/yacc (or pclex/pcyacc or flex/bison), and is divided into three major tasks:

1. Design the remainder of the lex (flex) specification for GUI.

2. Redesign the yacc (bison) specification for GUI.

3. Integrate and Test your combined lex/yacc (flex/bison) specification.

Work under the assumptions of project part 1. Note that your code implementation should be minimal!

Now, let's review what's required for each task of the project. First, the web site contains a set of four files, namely: sample GUI program (guiho.pgm.txt), a partial lex specification (guiho.1), a "complete" yacc specification (guiho.y), and the debugging output from the execution of the GUI program (guiho.sam.txt). The key issue regarding all of these files is that you CANNOT change the syntax of the sample program!! Any changes that you make to be able to correctly analyze and parse the sample program must occur in the lex and/or yacc specifications!! Note that the man command can be used to check the manual pages for: lex(flex) or yacc(bison), or you can consult the lex/yacc textbook for the course.

For the first task, you must complete the lex specification. Clearly a number of tokens are missing. The tokens that you need are given in the yacc specification in the %token command at the top of the file. (Note: These token commands in yacc automatically generate #defines.) The regular definitions (ws, word, dqs, comment) in the lex file are not necessarily correct, in that they may not be able to recognize the appropriate lexemes in guiho.pgm.txt. To check you lex specification for errors and to generate the lex.yy.c file on Unix, simply type: lex guiho.1. When using pclex, the command is issued from an MS DOS window as: pclex -c guiho.1, and the compile generates the file: yylex.c.

For the second task, you must debug and correct the yacc specification given in guiho.y. There are a number of explicit and subtle errors in the specification! The major problem involves the three bind blocks for binding buttons, functions, and panel items. The grammar rules for these three blocks allow for possible epsilon transitions (e.g., for cases when one of them is not given in a GUI). However, for some reason, not all of the combinations work. That is, if only a bind buttons section is given, the grammar is OK. If both bind buttons and functions are given, the grammar works. Also, if all three are present in the sample program, the grammar works. But, if one of the earlier ones is missing, then the parse goes into an error state. So, you must investigate this problem and attempt to correct it by redesigning the grammar. In addition to this "major" glitch,
there are also many minor grammar problems that must be found and corrected. I want to stress again that for both this task and the first task, you CANNOT change the sample input program given in guiho.pgm.txt!!! Make changes to the lex and yacc specification. To "compile" your yacc specification on Unix type: yacc guiho.y. The compile generates the file: y.tab.c. When using pcyacc, the command is issued from an MS DOS window as: pcyacc -c guiho.y, and the compile generates the file: yytab.c.

To assist you in understanding where shift/reduce or reduce/reduce conflicts occur, and for tracing your derivations, the following yacc switch is helpful:

```
yacc -v guiho.y    // Generates the file y.output with all of
                   // the LALR(1) states for the GUI language.
pcyacc -v -c guiho.y // Generates the file yy.ltr which also
                   // contains the LALR(1) states.
```

Note that every time you yacc the grammar specification a new y.tab.c or yytab.c is generated! Thus, for debugging output (as given in guiho.sam.txt), you must re-edit the file and add in the initialization before compiling. To compile you lexical analyzer/parser, use the Unix C compiler as follows: cc y.tab.c -l. Notice that the end of the yacc specification includes the lex.yy.c file to represent the entire compiler. The main program simply calls yyparse. Once compiled, execution occurs by: a.out < guiho.pgm.txt. Of course, you can also redirect your output to a file!

The final task is to thoroughly integrate and test your GUI compiler. In addition to making sure that you compiler runs on the sample (guiho.pgm.txt), you must also conduct tests using your own sample and other programs. While I have done all of the previous examples using lex and yacc, don't forget that flex and bison are available. Also, for the third project where you will actually provide code for the different grammar rules in yacc (bison), if you plan on using C++, you must use flex and bison.

For the second part of the project, document your efforts as follows:

- Overview of your completed project and its documentation (1/2 to 1 page).
- For the lex specification, indicate the final regular definitions that you used for: ws, word, dqs, comment.
- For the yacc specification, keep a log of all grammar changes that corrected your problems. For each change:
  1. the problem that was identified and corrected,
  2. the original grammar subset, and
  3. the modified grammar subset.
- Testing plan and approach, test cases and test results.
- Lex (guiho.l) and yacc (guiho.y) specifications.
- Instructions for compiling and running your GUI LA/Parser.

DO NOT, I repeat, DO NOT, hand in lex.yy.c/yylex.c or y.tab.c/yytab.c!!! The documentation must be word-processed! Lastly, also submit an electronic version of your files (zipped) by the start of class, with the file named: “lastnameproj2.zip”, e.g., “demurjianproj2.zip”.