2lexical-analysis-lab

CST 334 (Operating Systems)
Dr. Glenn Bruns

Lab: Lexical analysis

The purpose of this lab is to give you experience in using a lexical analyser. See hints at the end only if you get stuck.

Copy the file:

```
/home/CLASSES/brunsglenn/cst334/labs/lex-lab/lab.tar
```

to a directory of your own, and extract the files.

1. Look at file main.c. The main program initializes the lexical analyzer, and then calls the function parse(), which is defined in parser.c

2. Look at the file parser1.c. The function parse() calls the lexical analyzer to get the lookahead symbol, then calls function assign_stmt(), which will parse a simple assignment statement.

3. Look at the function assign_stmt(). The comment at the top of the function shows the syntax of a very simple assignment statement:

   assign_stmt ::= ID "=" NUM

   The string "x = 3" can be derived from this grammar. Make sure you understand what assign_stmt() is doing.

4. Look at the Makefile. Make sure the value of VERSION is parser1.c. Look at what the target 'test1' does. Then, "make test1" at the command line. Does the test work?

   (Note: if your machine complains about the <<< in the Makefile, create a file for the input text and use regular input redirection with <)

5. Next try the targets 'test2' and 'test3' of the Makefile.

6. Try running the parser from the command line. Enter ./parser, hit return, enter your input, then enter ctrl-D to indicate end of file. Like this:

```
$ ./parser
x = 10
parsed
$ ./parser
x =       10
parsed
$ ./parser
x
= 10
parsed
$
```

7. Look at the function match() in parser1.c. It checks to make sure the token passed as a parameter is equal to the lookahead token. If so, it gets the next token. Otherwise, it reports a syntax error.

8. Copy parser1.c to parser2.c Modify assign_stmt() so that it uses function match(). This will make the code of assign_stmt() much simpler.

9. Change the value VERSION in the Makefile to parser2.c Try the three tests again.

10. Copy parser2.c to parser3.c Modify assign_stmt() so that the variable name and number are saved during parsing, and then the assignment statement is printed at the end of the function. You will need

the function lexer_id_val() to get the value of the ID token, and the function lexer_num_val() to get the value of the NUM token.

Note: lexer_num_val()

When you print the statement at the end of the function, you can use a print statement like this:

```
    printf("%s = %d\n", id, i);
```

11. Change the value VERSION in the Makefile to parser3.c. Try the three tests again. Please note carefully that the "pretty printed" output will have one space on either side of the equals sign, even if the input doesn't.

12. If you still have time, copy parser3.c to parser4.c, and change assign_stmt() so that an assignment symbol has the syntax

    assign_stmt ::= ID "=" NUM + NUM

Also, modify your code to print the right thing at the end.

13. If you still have time, modify your code so that an assignment statement must have a semicolon at the end.

14. If you still have time, write the BNF for a simple function call of the form f(x), where 'f' and 'x' are both ID tokens, and add a function fun_call() to parser.c. Test your function.

Hints

1. Try hard to remember how to do this. Hint: 'x' stands for extract.

tar xf lab.tar

8. Here is the revised assign_stmt():

```
void assign_stmt() {
   match(ID);
   match('=');
   match(NUM);
}
```

10. Here is the revised assign_stmt():

```
void assign_stmt() {
   char *id;
   int i;
   if (lookahead == ID) {
          id = lexer_id_val();
   }
   match(ID);
   match('=');
   if (lookahead == NUM) {
          i = lexer_num_val();
   }
   match(NUM);
   printf("%s = %d\n", id, i);
}
```