Wanner HernandezR

CMSC430 Week 2 Project 1

Professor Jarc, Duane

March 29, 2021

**CMSC 430 Project 1**

The first project involves modifying the attached lexical analyzer and the compilation listing generator code. You need to make the following modifications to the lexical analyzer, scanner.l:

1. A new token ARROW should be added for the two character punctuation symbol =>.
2. The following reserved words should be added:

case, else, endcase, endif, if, others, real, then, when

Each reserved words should be a separate token. The token name should be the same as the lexeme, but in all upper case.

1. Two additional logical operators should be added. The lexeme for the first should be or and its token should be OROP. The second logical operator added should be not and its token should be NOTOP.
2. Five relational operators should be added. They are =, /=, >, >= and <=. All of the lexemes should be represented by the single token RELOP.
3. One additional lexeme should be added for the ADDOP token. It is binary -.
4. One additional lexeme should be added for the MULOP token. It is/.
5. A new token REMOP should be added for the remainder operator. Its lexeme should be

rem.

1. A new token EXPOP should be added for the exponentiation operator. Its lexeme should

be \*\*.

1. A second type of comment should be added that begins with // and ends with the end of

line. As with the existing comment, no token should be returned.

1. The definition for the identifiers should be modified so that underscores can be included,

however, consecutive underscores, leading and trailing underscores should not be

permitted.

1. A real literal token should be added. It should begin with a sequence of one or more

digits following by a decimal point followed by zero or more additional digits. It may optionally end with an exponent. If present, the exponent should begin with an e or E, followed by an optional plus or minus sign followed by one or more digits. The token should be named REAL\_LITERAL.

1. A Boolean literal token should be added. It should have two lexemes, which are true and false. The token should be named BOOL\_LITERAL.

You must also modify the header file tokens.h to include each the new tokens mentioned above.

The compilation listing generator code should be modified as follows:

1. The lastLine function should be modified to compute the total number of errors. If any errors occurred the number of lexical, syntactic and semantic errors should be displayed.

If no errors occurred, it should display Compiled Successfully. It should return the

total number of errors.

1. The appendError function should be modified to count the number of lexical, syntactic

and semantic errors. The error message passed to it should be added to a queue of

messages that occurred on that line.

1. The displayErrors function should be modified to display all the error messages that

have occurred on the previous line and then clear the queue of messages.

An example of the output of a program with no lexical errors is shown below:

1 (\* Program with no errors \*)  
2  
3 function test1 returns boolean;  
4 begin  
5 7 + 2 > 6 and 8 = 5 \* (7 - 4); 6 end;

Compiled Successfully

Here is the required output for a program that contains more than one lexical error on the same line:

1 -- Function with two lexical errors

2

3 function test2 returns integer;

4 begin

5 7 $ 2 ^ (2 + 4);  
Lexical Error, Invalid Character $ Lexical Error, Invalid Character ^

6 end;

Lexical Errors 2

Syntax Errors 0

Semantic Errors 0

**Course file:**

* listing.cc
* listing.h
* scanner.l
* tokens.h
* makerfile

**After Compile:**

* scanner.c
* scanner.o
* listing.o
* compile

**Approach:**

* I started making changes with one file at a time because I did not want to get that many confusions if I did encounter some error. I started with listing.cc.
* I added three more Errors commands into the code which they were ; lexErrors, semErrors, synErrors.

Ex.

static int lexErrors = 0;

static int semErrors = 0;

static int synErrors = 0;

* I then procced to adding the functions fiction and commands to make the error execute correctly when I compile my code.
* I added line where if the compile was not successful than the error would display.

Ex.

printf("Compiled Succesfully");

}

else

{

printf("Lexical Errors: %4d\n", lexErrors);

printf("Syntax Errors: %4d\n", synErrors);

printf("Semantic Errors: %4d\n", semErrors);

}

return totalErrors;

* I made message that will prompt when the compiled output an error.
* After that I move to my next file scanner.l , where I need it to make a few modifications to the lexical analyzer. Where I added the recommended changes, I need to make to the code I would mention them but they are quite a lot so I will give a few examples;
* REAL\_LITERAL {digit}+\.{digit}\*([Ee][+-]?{digit}+)?
* rem {ECHO; return(REMOP); }
* begin { ECHO; return(BEGIN\_); }
* boolean { ECHO; return(BOOLEAN); }
* true { ECHO; return(BOOL\_LITERAL); }
* false { ECHO; return(BOOL\_LITERAL); }
* or { ECHO; return(OROP); }
* not { ECHO; return(NOTOP);}
* The lines could go on.
* The Next changes were made into the tokens.h, where I Identify all the functions I added on file scanner.l. Making sure all other token/functions were define so that the code could read and compiled all the files with no error.

**Compiled Error Come Across-**

* I came across only with a few problems, The first one was the I could not Compile my code because I did have private access with Terminal.
* Another problem I came across it was the mistake of using // for my headers in scanner.l and tokens.h files which it was giving me error and I did not understand why was I getting that problem.
* Last problem was related to the function BOOL\_LITTERAL true/false, The reason why this compile error gave me a lot of trouble is because the mistake was really small and I was already really stress because my code kept coming with small details error. The problem was that I did not define BOOL\_LITTERAL on tokens.h correctly and I had it under the name BOOLEAN\_LITERAL which it was a mistake that cost me a problem with the compile of the files.
* This error taught me a lot and I was able to realize that even the smallest details can mess with my entire code.

**Instruction to Compile code/open iOS:**

* Open Terminal
* Make sure to have private access on terminal before proceeding
* Prompt command cd desktop (to go to the require files)
* Prompt command cd CMSC430\_Project1(File name)
* Prompt command ‘make’ (ls to check file, make to compile the files)
* After doing that just start testing your files
* Prompt command ./compile <test1.txt(name of the test file)
* The code will work successfully

**Test compile code:**

**Graphical user interface, application, Word

Description automatically generated**

**Test 1:** Function with arithmetic expression

**Graphical user interface, application, Word

Description automatically generated**

**Test 2:** Program containing the comment, modified identifier and new literals

Graphical user interface, text, application, Word

Description automatically generated

**Test 3:** Punctuation symbols

Graphical user interface, text, application, Word

Description automatically generated

**Test 4:** Program containing arrow symbol and new reserved words

Graphical user interface, application, Word

Description automatically generated

**Test 5:** Program containing the new operators

Graphical user interface, application, Word

Description automatically generated

**Test 6:** Test comment

Graphical user interface, text, application, Word

Description automatically generated

**Test 7:** Function with arithmetic expression

Graphical user interface, application, Word

Description automatically generated

**Test 8:** Punctuation symbols

Graphical user interface, text, application, Word

Description automatically generated

**Test 9:** Program containing the new operators

Graphical user interface, application, Word

Description automatically generated

**Test 10:** Program containing arrow symbol and new reserved words

**Graphical user interface, application, Word

Description automatically generated**

**Testerror1:** Function with two lexical errors

Graphical user interface, application, Word

Description automatically generated

**Testerror2:** Testing various wrong characters

Graphical user interface, application, Word

Description automatically generated

**Testerror3:** Test comments

Graphical user interface, application, Word

Description automatically generated

**Testerror4:** Function with two lexical errors

Graphical user interface, application, Word

Description automatically generated

**Testerror5:** Punctuation symbols

**Graphical user interface, text, application, Word

Description automatically generated**

**Lessons learn:**

This week project I was able to pick up new skills like how to compile a multiple source files by using ‘makefile’ to me this was very difficult because I have never done or lack the experience in C programing so it was an interesting experience to learn this unique skill. I have able to pick up new stuff when I was going through modifications on the lexical analyzer. One thing that I learn and feel that will help me is how to approach with the changes on tokens.h and scanner.l file because it was something new that ever done so I was able to pick up the skills really fast and make my code work correctly. Like I mention on the approach I was far off from perfect when I was making this code because I encounter a lot of error that I feel it will help me on my futures codes. I am thankful that I was able to learn all these new things about C program and I am looking forward to keeping learning more and getting more experience. The lessons taught me a lot of things specially how to compile because I did not have the skill to compile multiple file sources.