Wanner HernandezR

CMSC430 Week 2 Project 2

Professor Jarc, Duane

April 13, 2021

**CMSC 430 Project 2**

The second project involves modifying the syntactic analyzer for the attached compiler by adding to the existing grammar. The full grammar of the language is shown below. The highlighted portions of the grammar show what you must either modify or add to the existing grammar.

function:

function\_header {variable} body

function\_header:  
FUNCTION IDENTIFIER [parameters] RETURNS type ;

variable:

IDENTIFIER : type IS statement

:

type:

INTEGER | REAL | BOOLEAN

body:

BEGIN statement END ;

statement:

expression ; |

REDUCE operator {statement} ENDREDUCE ; |

operator:

ADDOP | MULOP

case:

WHEN INT\_LITERAL ARROW statement

expression:  
( expression ) |  
expression binary\_operator expression |  
NOT expression |  
INT\_LITERAL | REAL\_LITERAL | BOOL\_LITERAL | IDENTIFIER

|

page1image62771904

parameters

page1image62770752

parameter {, parameter}

page1image62774400

parameter:

page1image62771520

IDENTIFIER : type

page1image62740288

IF expression THEN statement ELSE statement ENDIF ;

page1image62734528

CASE expression IS {case} OTHERS ARROW statement ; ENDCASE ;

binary\_operator: ADDOP | MULOP | REMOP | EXPOP | RELOP | ANDOP | OROP

In the above grammar, the red symbols are nonterminals, the blue symbols are terminals and the black punctuation are EBNF metasymbols. The braces denote repetition 0 or more times and the brackets denote optional.

You must rewrite the grammar to eliminate the EBNF brace and bracket metasymbols and to incorporate the significance of parentheses, operator precedence and associativity for all operators. Among arithmetic operators the exponentiation operator has highest precedence following by the multiplying operators and then the adding operators. All relational operators have the same precedence. Among the binary logical operators, and has higher precedence than or. Of the categories of operators, the unary logical operator has highest precedence, the arithmetic operators have next highest precedence, followed by the relational operators and finally the binary logical operators. All operators except the exponentiation operator are left associative. The directives to specify precedence and associativity, such as %prec and %left, may not be used

Your parser should be able to correctly parse any syntactically correct program without any problem.

You must modify the syntactic analyzer to detect and recover from additional syntax errors using the semicolon as the synchronization token. To accomplish detecting additional errors an error production must be added to the function header and another to the variable declaration.

Your bison input file should not produce any shift/reduce or reduce/reduce errors. Eliminating them can be difficult so the best strategy is not introduce any. That is best achieved by making small incremental additions to the grammar and ensuring that no addition introduces any such errors.

An example of compilation listing output containing syntax errors is shown below:

1 -- Multiple errors

2

3 function main a integer returns real;

Syntax Error, Unexpected INTEGER, expecting ':' 4 b: integer is \* 2;

Syntax Error, Unexpected MULOP

5 c: real is 6.0;

6 begin

7 if a > c then

8 b 3.0;  
Syntax Error, Unexpected REAL\_LITERAL, expecting ';'

else  
b = 4.;

endif;  
Syntax Error, Unexpected ';', expecting END

9 10 11

Lexical Errors 0

Syntax Errors 4

Semantic Errors 0

**Course file:**

* listing.cc
* listing.h
* scanner.l
* parser.y
* makerfile

**After Compile:**

* tokens.h
* parser.c
* parser.output
* parser.o
* scanner.o
* listing.o
* compile

**Approach:**

* Throughout this week Project 2 I had to make big changes to my listing.cc to make it easier for me to read and added a few extra line so that myself and other people that read my code could have a clear view of what is going on.
* New input that were added are; static string mulError = ""; static int semanticErrors = 0; static int errorLine = 0;
* I chose to use the If method because the If, Else method was confusing me when I was making the code and came across of small mistake, by changing my code a bit I made it easier for me to track my work and learn different ways to make my code more eligible and easier to track.
* After using this method I was able to track my work better and learn new skills and experience a different way to approach the code.
* For the second project I took a different approach I started with my scanner.l after changing a lot of my listing.cc to make it look cleaner and added a few more commands.
* For my scanner.l file I had to changes a few things like getting rid of a few lines due to giving me duplicate error file and not needed to be added under scanner.l file because it was implement into the new source file named parser.y.

Ex.:

int main()

{

firstLine();

FILE \*file = fopen("lexemes.txt", "wa");

int token = yylex();

while (token)

{

fprintf(file, "%d %s\n", token, yytext);

token = yylex();

}

lastLine();

fclose(file);

return 0;

}

* The next step that I made was get started with my parser.y file enabling to add new error and different new expression and compile output.
* I started by creating the character error message with the code.;

nt yylex();

void yyerror(const char\* message);

* After that I took the next step to identified my tokens : IDENTIFIER, INT\_LITERAL REAL\_LITERAL, BOOL\_LITERAL, ADDOP, MULOP, RELOP, ANDOP, EXPOP, OROP, NOTOP, REMOP, ARROW, THEN, WHEN, BEGIN\_ BOOLEAN, END, ENDREDUCE, FUNCTION, INTEGER, IS, REDUCE, RETURNS, CASE, ELSE, ENDCASE, ENDIF, IF, OTHERS, REAL.
* I then made the expression to make my code work properly as far I know I believe I did not miss anything and felt confident about this week code with zero mistake.
* I ended the code with making the yyerror characteristic message and identifying firstline() and Lastline() so it would run with the perser file and not with the scanner, this made my code more organize and less complicated to read.

Ex.:

void yyerror(const char\* message)

{

appendError(SYNTAX, message);

}

int main(int argc, char \*argv[])

{

firstLine();

yyparse();

lastLine();

return 0;

}

**Compiled Error Come Across-**

* For this week project I only come across one error which it was that I was getting a duplicate with the firstLine and LastLine because I did not realize until late that I have to erase the scanner.l code until I finish with my parser.y file after reading the error information and experimenting with my code I was able to figure out the source of the error which it was.

Ex.:

int main()

{

firstLine();

FILE \*file = fopen("lexemes.txt", "wa");

int token = yylex();

while (token)

{

fprintf(file, "%d %s\n", token, yytext);

token = yylex();

}

lastLine();

fclose(file);

return 0;

}

* This gave me a better understanding on reading error output and how to fix this mistake just by reading and trying out different options.

**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\_Project2(File name)
* Prompt command ‘make’ (ls to check file, make to compile the files)
* Compile: scanner.o parser.o listing.o

g++ -o compile scanner.o parser.o listing.o

* Scanner.c: scanner.l

flex scanner.l

mv lex.yy.c scanner.c

* Parser.o: parser.c listing.h

g++ -c parser.c

* Parser.c tokens.h: parser.y

bison -d -v parser.y

mv parser.tab.c parser.c

mv parser.tab.h tokens.h

* Listing.o: listing.cc listing.h

g++ -c lisitng.cc

* 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, text, email

Description automatically generated**

**Test 1:** Arithmetic operators

Graphical user interface, text, application

Description automatically generated with medium confidence

**Test 2:** Relational and logical operators

Graphical user interface, application, Word

Description automatically generated

**Test 3:** Conditional expression

Graphical user interface, application, Word

Description automatically generated

**Test 4:** Variable initialization

Graphical user interface, application, Word

Description automatically generated

**Test 5:** Parameter initialization

Graphical user interface, application, Word

Description automatically generated

**Test 6:** Parameter initialization

Graphical user interface, application, Word

Description automatically generated

**Test 7:** function main a: integer, b: integer returns integer;

Graphical user interface, application, Word

Description automatically generated

**Test 8:** Function with a Reduction

Graphical user interface, application, Word

Description automatically generated

**Test 9:** Test nested case

Graphical user interface, application, Word

Description automatically generated

**Test 10:** Two Parameter initialization

Graphical user interface, text, application, Word

Description automatically generated

**Teste 11:** Function with a ReductionGraphical user interface, application, Word

Description automatically generated

**Test 12:** Function with a Nested Reduction

Graphical user interface, application, Word

Description automatically generated

**Test case error in all productions:**

**TestError1:** Testing case statement with errors

Graphical user interface, application, Word

Description automatically generated

**TestError2:** Testing case statement with errors

Graphical user interface, application, Word

Description automatically generated

**TestError3:** Testing case statement with errors

Graphical user interface, application, Word

Description automatically generated

**TestError4:** Testing case statement with errors

Graphical user interface, text, application, Word

Description automatically generated

**TestError5:** Testing case statement with all added tokensGraphical user interface, text, application, Word

Description automatically generated

**TestError6:** Testing case statement with all added tokens

Graphical user interface, application, Word

Description automatically generated

**Test case with multiple errors:**

**ErrorMultipleTest1:** Testing case statement with all added tokens**Graphical user interface, application, Word

Description automatically generated**

**ErrorMultipleTest2:** Testing case statement with errors

Graphical user interface, application, Word

Description automatically generated

**ErrorMultipleTest3:** Multiple errors

Graphical user interface, text, application, Word

Description automatically generated

**Lessons learn:**

This week project 2 I was able to pick up new skills like how to read and resolved compile error and how to approach the error and take the necessary steps to fix this error. For example, I got the error of duplicate in my compile output at first did not understand how to resolve or what necessary steps I need to take but after trials and error I was able to figure out where the problem was. I did come through a big problem on Project 1 because I came across multiple errors, so it made me confused on what necessary steps I needed to take and how to fix these problems. In this week I got to learn a little about parser and what are they use for as well as what steps I need to take to not come up with a lot of errors and come across big mistakes. This week was easier than last week because I did not come across the same mistake I made last week and I made sure to double check and make secondary resources took a look the code for any small mistake that I might have miss and could not see my-self. I still got a long way to make sure I fix this small mistake my-self, but I do feel a bit confident on my coding skills and enjoy the challenges that these projects put me through.