Robert D. Carswell

6 February 2024

project 2 report

CMSC 430 – COMPILER THEORY AND DESIGN

Contents

[Executive Summary 2](#_Toc158046104)

[Testing 3](#_Toc158046105)

[Test Case Table 3](#_Toc158046106)

[Test Case Screenshots 7](#_Toc158046107)

[Test Case 0 7](#_Toc158046108)

[Test Case 1 7](#_Toc158046109)

[Test Case 2 8](#_Toc158046110)

[Test Case 3 8](#_Toc158046111)

[Test Case 4 9](#_Toc158046112)

[Test Case 5 10](#_Toc158046113)

[Test Case 6 10](#_Toc158046114)

[Test Case 7 11](#_Toc158046115)

[Test Case 8 11](#_Toc158046116)

[Test Case 9 11](#_Toc158046117)

[Test Case 10 12](#_Toc158046118)

[Test Case 11 12](#_Toc158046119)

[Test Case 12 13](#_Toc158046120)

[Test Case 13 13](#_Toc158046121)

[Test Case 14 13](#_Toc158046122)

[Test Case 15 14](#_Toc158046123)

[Test Case 16 14](#_Toc158046124)

[Test Case 17 14](#_Toc158046125)

[Test Case 18 15](#_Toc158046126)

[Test Case 19 15](#_Toc158046127)

[Test Case 20 16](#_Toc158046128)

[Approach 16](#_Toc158046129)

[Lessons Learned 17](#_Toc158046130)

# Executive Summary

This project involves enhancing the syntactic analyzer for a given compiler by extending its grammar by modifying and adding the existing grammar. This included new terminals and non-terminals to eliminate the EBNF brace and bracket meta-symbols. At the same time, it introduces the meanings of parentheses, operator precedence, and associativity rules for arithmetic, logical, and relational operators. While adding error productions to find and fix syntax errors using the semicolon as the synchronization token. To ensure the correct parsing of a syntactically valid program is done without errors. To achieve these goals, attention was given to operator precedence, associativity, and the introduction of error productions. In turn, it creates an effective syntactic analyzer capable of handling additional syntax errors and producing correct parse results.

# Testing

## Test Case Table

| Test Cases | Description | Expected Output | Actual Output | Pass/Fail |
| --- | --- | --- | --- | --- |
| Test Case 0 | Compile program | Compiles with no shift/reduce conflicts | Test Case 1 | Read test file syntax1.txt |
| Test Case 1 | Read test file syntax1.txt | File contents with:  Msg (Under line 5):  syntax Error,  unexpected INT\_LITERAL,  Expecting ‘;’  Msg (Bottom of file):  Lexical Errors: 0  Syntax Errors: 1  Semantic Errors: 0 | File contents with:  Msg (Under line 5):  Syntax Error,  Unexpected INT\_LITERAL,  Expecting ‘;’  Msg (Bottom of file): Lexical Errors: 0  Syntax Errors: 1  Semantic Errors: 0 | Pass |
| Test Case 2 | Read test file syntax2.txt | File contents with:  Msg (Under line 3):  syntax error,  unexpected INTEGER, expecting ':'  Msg (Bottom of file): Lexical Errors: 0  Syntax Errors: 1  Semantic Errors: 0 | File contents with:  Msg (Under line 3):  syntax error,  unexpected INTEGER, expecting ':'  Msg (Bottom of file): Lexical Errors: 0  Syntax Errors: 1  Semantic Errors: 0 | Pass |
| Test Case 3 | Read test file Syntax3.txt | File contents with:  Msg (Under line 4):  syntax error,  unexpected INTEGER, expecting ':'  Msg (Bottom of file): Lexical Errors: 0  Syntax Errors: 1  Semantic Errors: 0 | File contents with:  Msg (Under line 4):  syntax error,  unexpected INTEGER, expecting ':'  Msg (Bottom of file): Lexical Errors: 0  Syntax Errors: 1  Semantic Errors: 0 | Pass |
| Test Case 4 | Read test file syntax4.txt | File contents with:  Msg (Under line 6):  syntax error,  unexpected INT\_LITERAL, expecting ';'  Msg (Under line 8):  syntax error, unexpected ENDSWITCH, expecting CASE or OTHERS'  Msg (Bottom of file): Lexical Errors: 0  Syntax Errors: 2  Semantic Errors: 0 | File contents with:  Msg (Under line 6):  syntax error,  unexpected INT\_LITERAL, expecting ';'  Msg (Under line 8):  syntax error, unexpected ENDSWITCH, expecting CASE or OTHERS'  Msg (Bottom of file): Lexical Errors: 0  Syntax Errors: 2  Semantic Errors: 0 | Pass |
| Test Case 5 | Read test file syntax5.txt | File contents with:  Msg (Under line 3):  syntax error,  unexpected INTEGER, expecting ':'  Msg (Under line 4):  syntax error,  unexpected MULOP  Msg (Under line 8):  syntax error,  unexpected MULOP  Msg (Under line 11):  syntax error,  unexpected ARROW, expecting INT\_LITERAL  Msg (Under line 13):  syntax error,  unexpected ENDSWITCH, expecting CASE or OTHERS  Msg (Bottom of file): Lexical Errors: 0  Syntax Errors: 5  Semantic Errors: 0 | File contents with:  Msg (Under line 3):  syntax error,  unexpected INTEGER, expecting ':'  Msg (Under line 4):  syntax error,  unexpected MULOP  Msg (Under line 8):  syntax error,  unexpected MULOP  Msg (Under line 11):  syntax error,  unexpected ARROW, expecting INT\_LITERAL  Msg (Under line 13):  syntax error,  unexpected ENDSWITCH, expecting CASE or OTHERS  Msg (Bottom of file): Lexical Errors: 0  Syntax Errors: 5  Semantic Errors: 0 | Pass |
| Test Case 6 | Read test file test1.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 7 | Read test file test2.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 8 | Read test file test3.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 9 | Read test file test4.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 10 | Read test file test5.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 11 | Read test file test6.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 12 | Read test file test7.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 13 | Read test file test8.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 14 | Read test file test9.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 15 | Read test file test10.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 16 | Read test file test11.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 17 | Read test file test12.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 18 | Read test file test13.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 19 | Read test file test14.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |
| Test Case 20 | Read test file test15.txt | File contents with :  Msg (Bottom of file):  Compilation Successful | File contents with :  Msg (Bottom of file):  Compilation Successful | Pass |

# Test Case Screenshots

### Test Case 0

A computer screen shot of a program

Description automatically generated

### Test Case 1

A screen shot of a computer

Description automatically generated

### Test Case 2

A computer screen shot of a black screen

Description automatically generated

### Test Case 3

A screenshot of a computer program

Description automatically generated

### Test Case 4

A screenshot of a computer program

Description automatically generated

### Test Case 5

A screenshot of a computer program

Description automatically generated

### Test Case 6

A black screen with white text

Description automatically generated

### Test Case 7

A computer screen with white text

Description automatically generated

### Test Case 8

A screenshot of a computer

Description automatically generated

### Test Case 9

A computer screen with white text

Description automatically generated

### Test Case 10

A screen shot of a computer

Description automatically generated

### Test Case 11

A screenshot of a computer program

Description automatically generated

### Test Case 12

A computer screen shot of a black screen

Description automatically generated

### Test Case 13

A computer screen with white text

Description automatically generated

### Test Case 14

A screen shot of a computer

Description automatically generated

### Test Case 15

A screen shot of a computer

Description automatically generated

### Test Case 16

A computer screen with white text

Description automatically generated

### Test Case 17

A screenshot of a computer

Description automatically generated

### Test Case 18

A screenshot of a computer program

Description automatically generated

### Test Case 19

A screenshot of a computer program

Description automatically generated

### Test Case 20

A screenshot of a computer program

Description automatically generated

# Approach

It was said that slow and steady wins the race; well, attention to detail doesn’t hurt, and both were needed on Project 2. I started by analyzing the requirements and the make file. Taking note of what was being generated and how to get a firm grasp on the flow of information. I started by moving my scanner.l, listing.cc, listing.h, and updating the parser.y to handle the new tokens since it produced the token.h. However, I had trouble with my scanner.l, so I used the skelleton and added my old one as I progressed. I slowly worked my way down the new grammar, adding and modifying the terminals and non-terminals while testing the current and all previous passable tests. At the same time, I took note of the original structure, using it to eliminate the EBNF braces and brackets, add the error productions, and account for the operator precedence and associativity rules.

# Lessons Learned

I learned that being attentive, slow, and steady does win the race. In the early stages of the project, I started over three times. I was having problems with my scanner.l file, so I opted to cut and paste my old one in as I went along since the skeleton was the baseline. Well, I came to find out the main problem was with the ‘%’ token. In project 1, it was called REMOP, and in project 2, it was called MODOP. Everything moved along nicely until the end, when I ran syntax5.txt. It was getting an error in the header, which was checked in a previous test. The error was the comment double dash, so I went through the parser with a fine toothcomb. It wasn’t until I noticed some of the tests in the approach were slightly different that I ran the tests off each page. The approach led me to test 5, which is my test 20. It used the integer literals and was failing when it shouldn’t, but it led me to the scanner.l literals that I forgot to cut and paste from Project 1. This revelation helped me identify the source of both issues and fix them accordingly.