Project 1 Documentation

Antonio E Ramirez

University of Maryland Global Campus

29 MAR 2020

# Approach

This project seemed quite daunting at first look. I do not have much experience in both lexical analysis as well as C/C++. These were my biggest hurdles associated with this project. The approach taken was to understand the end state first. After the objectives were understood, the given code was gone through to ensure the current functionality was understood prior to any changes were made. This verified any gaps in knowledge and allowed them to be filled immediately. Next, it gave a clearer understanding as to what needed to be done to the “skeleton” to get it to the desired end state. Once all of this was done, the rubric’s list was gone through step-by-step with testing done after each. This was done to ensure the program continued to work as intended prior to moving on. After all the steps were completed, the final grading table was used to form a test plan.

# Test Plan

To test the final program, the rubric was relied on heavily to establish final testing parameters. Going through both the Functionality and Test Cases sections made it easy to find exactly what areas of the program should be tested. My intention is to test everything using the given test cases as well as one test case developed by me. The test case developed by me will incorporate everything on the rubric. Not only will this cover all necessary areas of testing, but by putting them all in one test case it will put the most strain on the program.

Test Case 1:

// Function with arithmetic expression

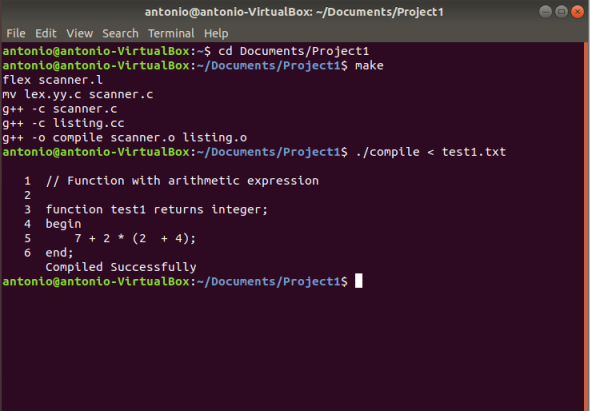
function test1 returns integer;

begin

7 + 2 \* (2 + 4);

end;

Output:



Test Case 2:

-- Function with five lexical errors

function test2 returns integer;

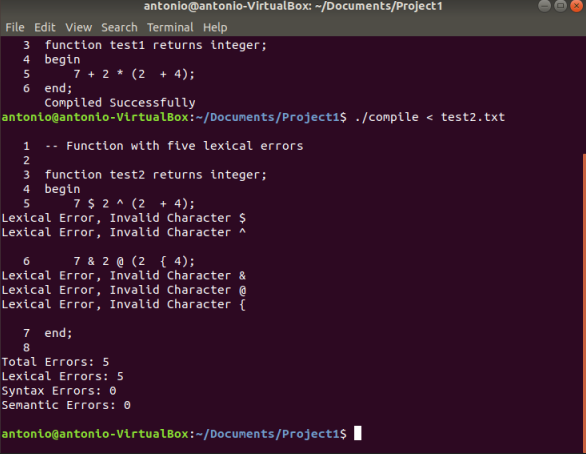
begin

7 $ 2 ^ (2 + 4);

7 & 2 @ (2 { 4);

end;

Output:



Test case 3:

-- Punctuation symbols

,;()

-- Identifier

name name123

-- Literals

123

-- Logical operators

and or not

-- Relational operators

<

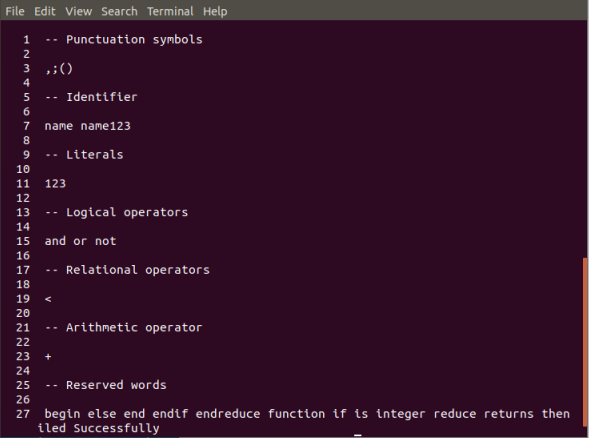
-- Arithmetic operator

+

-- Reserved words

begin else end endif endreduce function if is integer reduce returns then

Output:



Test case 4:

-- Comment test

// Another\_comment\_test

function test1 returns integer;

begin

7 + 2 \* (2 + 4);

if 7 > 4 or 3 < 5 or 5 >= 5 or 9 <= 10 then

10 / 2 /= 4;

endif

// Exponent test

7 \* \*

//Errors

6 ^ 5

3 # 4

4 $ % ^ &

// ADDOP

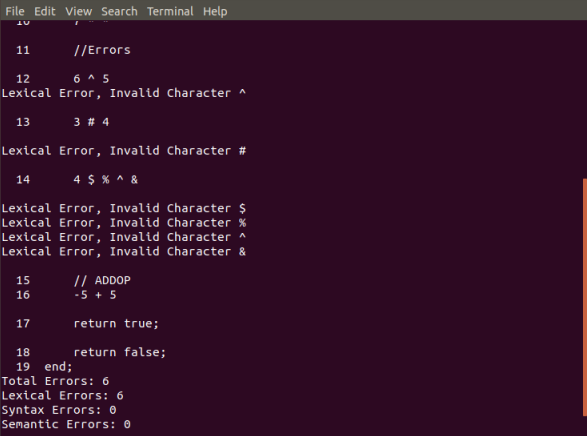
-5 + 5

return true;

return false;

end;

Output:



# Lessons Learned

I learned a lot over the course of this project. I struggled with getting my Ubuntu/VirtualBox to work as intended. After getting this figured out, the next hurdle was understanding the given code and how it functioned. This involved quite a bit of research that took up a large chunk of time. Next, was revisiting the week’s videos and readings to understand how to use the terminal/makefile to create our scanner. I had mistyped something initially that was giving me errors that I could not resolve. I hand-typed the code into my text editor as this helps me go through all of the code and makes sure I understand it. This led to the issue of the unresolved error. In the end, I copied and pasted the provided code to correct the issue. I would like to know what was giving me the error for future reference. I then struggled with getting the errors to add and display at the end. After playing around with the different functions I was eventually able to get it to work. This was a rewarding project to complete and I feel that my understanding has increased greatly.