CS348-Assignment 4

Sweeya Reddy 200101079
Vatsal Gupta 200101105

README

2nd April 2023

ZIP FILE CONTENTS

1. Assignment 3.pdf
2. Assignment 4.pdf
3. Makefile
4. A4_40.c
5. A4_40.l
6. A4_40.y
7. A4_40.nc
8. output_A4_40.txt
9. output_A4_40_example1.txt
10. output_A4_40_example10.txt
11. output_A4_40_example2.txt
12. output_A4_40_example3.txt
13. output_A4_40_example4.txt
14. output_A4_40_example5.txt
15. output_A4_40_example6.txt
16. output_A4_40_example7.txt
17. output_A4_40_example8.txt
18. output_A4_40_example9.txt
19. README.pdf (here)

PROGRAMMING LANGUAGE USED (upon UBUNTU 20.04 LTS)

nanoC, C, lex, (parsing)Bison

Key Points/Pre-requisites

- Please make sure the terminal is run in the same directory (cwd) as the two files
- The program will create a .c file lex.yy.c from A4_40.l, lex file and subsequently the required parser files like y.tab.h and auto generate token numbers and then use them to parse the input in the A4_40.nc file.
- **The output is generated after creating & using an internal symbol table which is implicitly stored as a map in the c file. The printed output will display the tokens returned and the productions applied**

Commands to enter and expected output:

```
make
make test <or> make test_file
```

- Simply type make into the command line and then make test_file a file named output.txt is created along with other supporter files.
- **BOUNS:** You may also instead type make test which prints the output onto the terminal in a cleaner fashion where any time yylex is called and a token is returned, it is highlighted in red and anytime a production rule is applied it is highlighted in yellow.
- You can also retest the output by running make clean first which removes the optional files and then run make again which recreates them proving the sample output is indeed correct. Make clean will essentially employ the following function:

  “rm -f lex.yy.c y.tab.c y.tab.h lex.yy.o y.tab.o A4_40.o y.output a.out output_A4_40.txt”

```
make clean
```

- Note that the make file works by essentially first using flex to create lex.yy.c file
- Then, this is used in the tester A4_40.c file to combine the two and create an a.out file.
- To this a.out file, A4_40.nc file is fed and the lexer output is printed in the output file (output_A4_40.txt)

Note:

- **BOUNS:** We have shown the output for all the 10 examples as given in the assignment they can be seen in the files labelled output_A4_40_example<i>.txt for the i^{th} example

- Our test file solves the Tower of hanoi problem in nano C the following may be seen for reference (this nanoC code was self-generated) https://www.geeksforgeeks.org/c-program-for-tower-of-hanoi-2/
- The lexer follows a certain priority order as given in assignment 3, and the same lexer as submitted earlier has been used with minor to no modification.
- The string literals to be terminated by \0 will be done at parser level and for now String literals are foud solely in the c-type form of simply specifying a string within double quotes
- Escape sequences are interpreted as the matching of instances like \? Instead of just ?
- **BONUS:** Some extra punctuations are handled and you may see ids like <?,token-id,?>
- For the punctuator class, we use separate token although this may be changed quite easily by changing <QUESTION,,> to <PUNCTUATOR : QUESTION,,>. We have followed a different convention though the lexer still matches the expression as a pucntuator due to the following regex, which was explicitly defined.

```
PUNCTUATOR
"["|"]"|"("|")"|"{"|"}"|"->"|"&"|"*"|"+"|"-"|"/"|"%"|"!"|"?"|"<"|">"|"<="|
">="|"=="|"!="|"&&"|"||"|"="|":"|";"|","
```

- We have modified some grammar rules, to make the production rules correct and perform right. We have taken inspiration from several sources like the following segment of grammar on the right (A Grammar for the C-Programming Language (Version S21) March 23, 2021 1 Introduction)

1. $program \rightarrow declList$
2. $declList \rightarrow declList\ decl \mid decl$
3. $decl \rightarrow varDecl \mid funDecl$

- The changes are done for the translation_unit segment and the function_definition segment all the others are retained.

Appendix

Colored Output Screenshot:

```
TOKEN:  SQRBROPEN
TOKEN:  IDENTIFIER
primary_expression
primary_expression
TOKEN:  SUB
unary_expression
multiplicative_expression
additive_expression
TOKEN:  INTEGER_CONSTANT
primary_expression
primary_expression
TOKEN:  SQRBRCLOSE
unary_expression
multiplicative_expression
additive_expression
relational_expression
equality_expression
logical_and_expression
logical_or_expression
conditional_expression
assignment_expression
expression
postfix_expression
TOKEN:  SEMICOLON
unary_expression
multiplicative_expression
additive_expression
relational_expression
equality_expression
logical_and_expression
logical_or_expression
conditional_expression
assignment_expression
expression
jump_statement
statement
```