Lexical Analyzer Report

Ahmed Khaled Saad Seif 07

Ahmed Reda Amin 09

Hisham Osama Ghazy --

Problem Description :

It is required to develop a suitable Syntax Directed Translation Scheme to convert Java code to Java bytecode, performing necessary lexical, syntax and static semantic analysis (such as type checking and Expressions Evaluation).
Generated bytecode must follow the standard bytecode instructions defined in Java Virtual Machine Specification.
Phase 1 :

- Your task in this phase of the assignment is to design and implement a lexical analyzer generator tool.
- The lexical analyzer generator is required to automatically construct a lexical analyzer from a regular expression description of a set of tokens.
- The tool is required to construct a nondeterministic finite automata (NFA) for the given regular expressions, combine these NFAs together with a new starting state, convert the resulting NFA to a DFA, minimize it and emit the transition table for the reduced DFA together with a lexical analyzer program that simulates the resulting DFA machine.
- The generated lexical analyzer has to read its input one character at a time, until it finds the longest prefix of the input, which matches one of the given regular expressions. It should create a symbol table and insert each identifier in the table.
- If more than one regular expression matches some longest prefix of the input, the lexical analyzer should break the tie in favor of the regular expression listed first in the regular specifications.
- If a match exists, the lexical analyzer should produce the token class and the attribute value.
- If none of the regular expressions matches any input prefix, an error recovery routine is to be called to print an error message and to continue looking for tokens.

Data Structures Used:

- Vectors : Vectors were used heavily during the implementation of this phase , the most prominent instance of its usage is the representation of our main adjacency list , Which is a vector of vectors of pairs of Char(input) and DFANode(A custom data Structure we built) .

- DFANode : Special types of data structures that carry important data about nodes in the DFA graph , such as whether it is an acceptance state or not and several other data connected to the transformation between DFA and NFA .
- State : Special type of data structures , it represents an NFA node , it includes an Id , whether it is an acceptance state or not and also all edges outgoing from this Node, a DFANode actually carries a vector of these NFA nodes in its new DFA form Representing NFA states that have been merged into this Node.
- NFA : The NFA graph is also represented as a vector of States.

Algorithms Used:

- NFA Algorithm :
  The NFA is formed from parsing the grammar , we then build an NFA graph using Simple concatination methods corresponding to our grammar , adding epsilon when Needed .
- DFA Algorithm :
  The DFA Algorithm is formed by eliminating epsilon transition states , part by part via a union find technique (here we are using a parent vector of parent indices),
  The parents correspond to the wrapper states that these NFA states will merge into , We then traverse the graph building a new one on its trace and ignoring edges between states of the same parent.
- Myhill-Nerode Algorithm [DFA Minimization] :
  The Myhill-Nerode Algorithm is used to convert a non-minimal DFA to a minimal DFA using boolean matrices , many additions were made to the algorithm to convert its Outcome to a meaningful representation , such as a continuous operation of joining Nodes , Converting between several representations and incorporating meta data About the graph and its nodes and edges.

Bonus Section:

→ ***Required Steps*** :

1- For most linux users lex will be available by default as it is a UNIX utility , but for Windows users , they might want to download it first from this location , Note : flex is an open source version of lex , both serve the same functionality.

2-Create a .l (lex) file , which contains your Grammar rules , its format should be :

```
%{
//any header file #include and any #define you want to make in C
%}

%%
//Your actual grammar goes here on the format
// {expected token                action to be performed in C ; }
//The token format follows general regex rules
// for ex

[a-zA-Z][_a-zA-Z0-9]*          printf( "IDENTIFIER" );
[1-9][0-9]*                    printf( "INTEGER" );

%%

// here we create the int yywrap(void) function , it has a trivial implementation that
// returns a non-zero value if no further processing is needed , or a zero value if a
//following input file is on its way and further processing is needed.

//we can also create a main() as an entry point for lex/flex , but most lex/flex
//implementations have a default main .
```

3-From the command line run " lex {file_name}.l " , this generates a lex.yy.c file

4-You can create any C file that utilizes the produced lex.yy.c file , which contains a group of functions used for tokenizing , ex. :

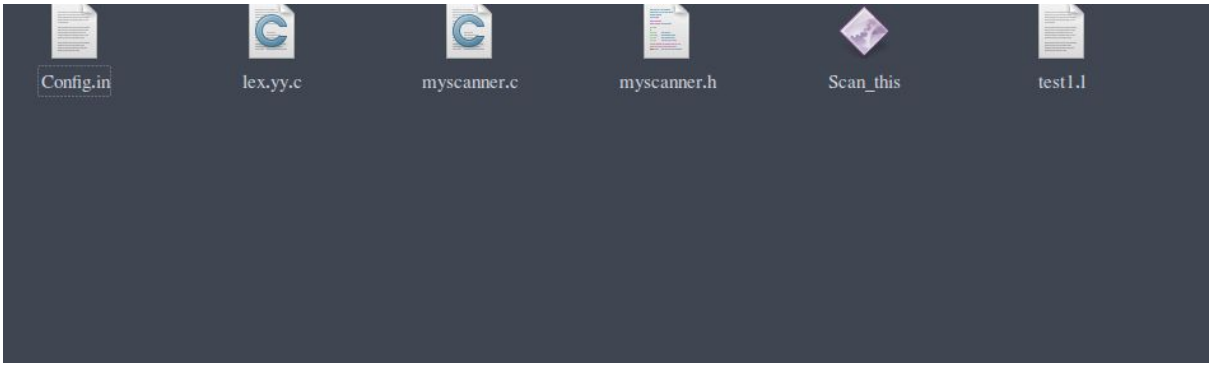
- yylex() : which is the main tokenizing function , it receives the next token and matches it to the actions specified in the grammar file .
- yylineno() : which is responsible for retrieving the line number of the sentence being tokenized in the input file .
- yytext(): which is responsible for retrieving the actual token as a string .

→ Note : Don`t forget to "extern" these functions .

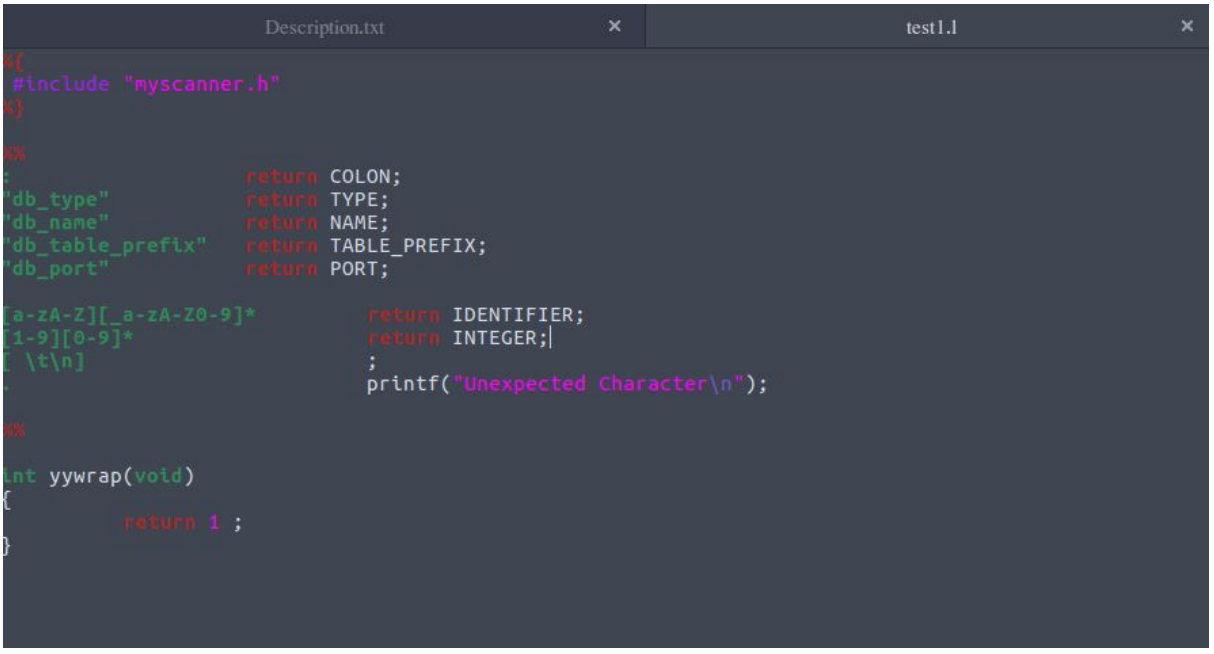
5-Compile your C file and run the input file(s) through it .

→ *Screenshots:*

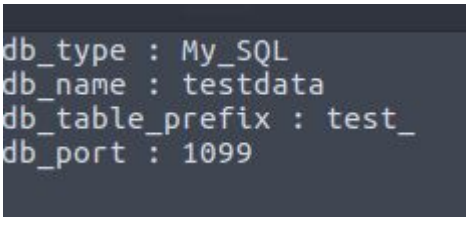
- Folder Layout :
  - Config : input
  - Lex.yy.c : compiled lex file
  - Myscanner.c : C file that uses the lex.yy.c analyser
  - Myscanner.h : some definitions
  - Scan_this : executable for the myscanner.c file
  - Test1.l : the lex file containing grammar rules



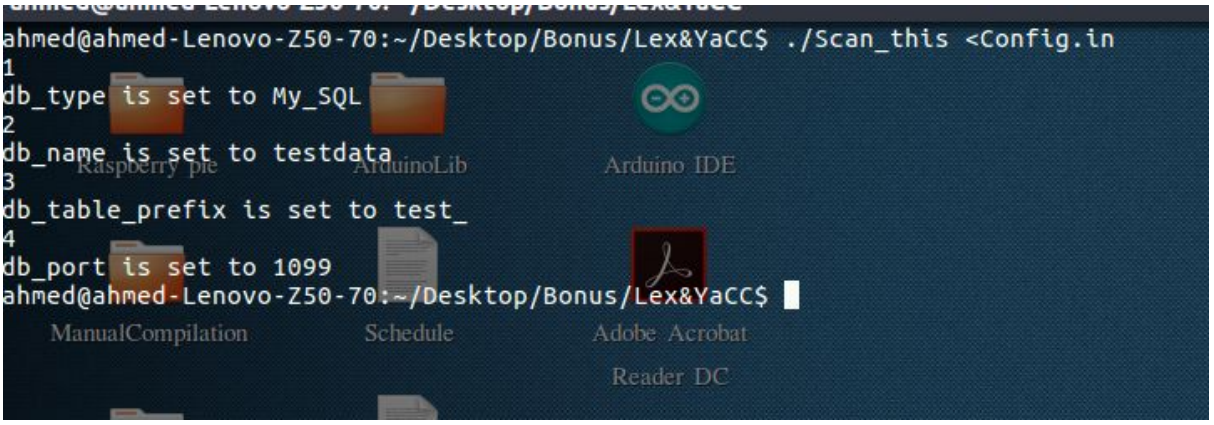
- Test1.l :



- Config.in [input file] :



- Output of the executable : The C file displays line number and converts the “ : ” to a “ is set to ” , all using the lex tokenizer .


ahmed@ahmed-Lenovo-Z50-70:~/Desktop/Bonus/Lex&YaCC$ ./Scan_this <Config.in
1
db_type is set to My_SQL
2
db_name is set to testdata
3
db_table_prefix is set to test_
4
db_port is set to 1099
ahmed@ahmed-Lenovo-Z50-70:~/Desktop/Bonus/Lex&YaCC$
Raspberry pie
ArduinoLib
Arduino IDE
ManualCompilation
Schedule
Adobe Acrobat
Reader DC