**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**Compiler Construction (CS F363)**

**II Semester 2019-20**

**Compiler Project (Stage-1 Submission)**

**Coding Details** 

**(February 24, 2020)**

1. IDs and Names of team members

ID: 2017A7PS0006P Name: PIKLU PAUL

ID: 2017A7PS0007P Name: RAJABABU SAIKIA

ID: 2017A7PS0090P Name: SAURAV VIRMANI

ID: 2017A7PS0111P Name: SIDDHANT KHARBANDA

ID: 2017A7PS0275P Name: SREYAS RAVICHANDRAN

1. Mention the names of the Submitted files :

1. lexer.c 7. adt.c 13. utils.c 20. t1.txt

2. lexerDef.h 8. adt.h 14. utils.h 21. t2.txt

3. lexer.h 9. hash\_table.c 15. driver.c 22. t3.txt

4. parser.c 10. hash\_table.h 16. makefile 23. t4.txt

5. parserDef.h 11. parseRules.c 17. rules.txt 24. t5.txt

6. parser.h 12. parseRules.h 18. nonterminals.txt 25. t6.txt

19. keywords.txt 26. parseTable.c 27. parseTable.h 28. terminals.txt

29. performa.docx

1. Total number of submitted files: **29** (All files should be in **ONE folder** named exactly as Group\_#, # is your group number)
2. Have you mentioned your names and IDs at the top of each file (and commented well)? (Yes/ no) **YES** [Note: Files without names will not be evaluated]
3. Have you compressed the folder as specified in the submission guidelines? (yes/no) **YES**
4. **Lexer Details:**
   * Technique used for pattern matching: **State Transition using DFA**
   * DFA implementation (State transition using switch case, graph, transition table, any other (specify): **Switch Case**
   * Keyword Handling Technique: **Stored Keywords in Hash Table**
   * Hash function description, if used for keyword handling: **Double Hashing using hash function which finds summation of power of x multiplied by each character.**
   * Have you used twin buffer? (yes/ no) **Yes**
   * Lexical error handling and reporting (yes/No): **Yes, token returned with index =-1**
   * Describe the lexical errors handled by you
5. Length of an identifier should be less than or equal to 20 characters.
6. Invalid token at start of lexeme.
7. If there is a single “.”
8. If there is a single “=”
9. If there is a single “!”
   * Data Structure Description for tokenInfo (in maximum two lines):

Token Info is implemented as a structure with fields as Line Number, Lexeme, index and Value. Value is of type union with fields as f\_val and i\_val. Index is the index of the Token used in the entire parser.

* + Interface with parse : **parseTree(char\* filename)**

1. **Parser Details:** 
   * **High Level Data Structure Description (in maximum three lines each, avoid giving C definitions used):**
     1. grammar :

rule\_rhs for storing the right hand side of a rule which stores the hash table entry pointer and next pointer

rules\_table for storing the key value pair, i.e. rule LHS as pointer to mapping table entry and RHS as rule\_rhs type linked list

rules array of type rules\_table for storing individual rules.

* + 1. parse table:

Integer type 2-D arrays with elements as the rule number corresponding to the Nonterminal and Terminal Pair. -1 for the Error entries , ie. pairs without any corresponding rule.

* + 1. parse tree: (Describe the node structure also)

Treenode is a structure with data, pointer to parent, pointer to right sibling and tag for (Terminal/NonTerminal).Data is of type union which can either be a nonterminal of type hash table item or a token of type Token.

* + 1. Parsing Stack node structure :

Stacknode is a structure with a hash table item , pointer to previous element and pointer to next element. Hash Table item is for storing the right hand side of the rules.

* + 1. Any other (specify and describe)

ht\_item with key of token, index of token and tag of token for storing the terminals and nonterminals entered into the hash table.

error\_list to store the errors of the parser.

* + **Parse tree** 
    1. Constructed (yes/no): **YES**
    2. Printing as per the given format (yes/no): **YES**
    3. Describe the order you have adopted for printing the parse tree nodes (in maximum two lines)

The inorder traversal for an n‐ary tree can be described as follows

Leftmost child ‐‐> parent node‐‐> remaining siblings (excluding the leftmost child)

* + **Grammar and Computation of First and Follow Sets** 
    1. Data structure for original grammar rules:

**rule\_rhs for storing the right hand side of a rule which stores the hash table entry pointer and next pointer**

**rules\_table for storing the key value pair, i.e. rule LHS as pointer to mapping table entry and RHS as rule\_rhs type linked list**

**rules array of type rules\_table for storing individual rules.**

* + 1. FIRST and FOLLOW sets computation automated (yes /no): **YES**
    2. Data structure for representing sets: **long int while computing the sets and ht\_item\_list for storing the hash table items corresponding to the set elements.**
    3. Time complexity of computing FIRST sets : **O(n) where n = # of rules**
    4. Name the functions (if automated) for computation of First and Follow sets :

**computeFirst(), computeFollow() and computeFirstofRule()**

* + 1. If computed First and Follow sets manually and represented in file/function (name that) **N/A**
  + **Error Handling** 
    1. Attempted (yes/ no): **YES**
    2. Printing errors (All errors/ one at a time) : All Errors
    3. Describe the types of errors handled

1. Invalid Token: lexical error
2. Unexpected Token Found i.e. no entry on parse table
3. Extra Tokens found
4. Reached end of the program without complete parse tree generation.
   * 1. Synchronizing tokens for error recovery (describe):

By searching for the next token which can provide us with a valid rule for the non terminal on the top of the stack.

* + 1. Total number of errors detected in the given testcase t6(with\_syntax\_errors).txt : 80

1. **Compilation Details:**
   * Makefile works (yes/no): **YES**
   * Code Compiles (yes/ no): **YES**
   * Mention the .c files that do not compile: N/A
   * Any specific function that does not compile: N/A
   * Ensured the compatibility of your code with the specified gcc version(yes/no): **YES**
2. **Driver Details**: Does it take care of the options specified earlier(yes/no): **YES**
3. **Execution** 
   * status (describe in maximum 2 lines): **Everything compiled and running.**
   * Execution time taken for
     + t1.txt (in ticks) : 5934 and (in seconds) : 0.005934
     + t2.txt (in ticks): 851 and (in seconds) : 0.000851
     + t3.txt (in ticks) : 1553 and (in seconds) : 0.001553
     + t4.txt (in ticks) : 4212 and (in seconds) : 0.004212
     + t5.txt (in ticks) : 4827 and (in seconds) : 0.004827
     + t6.txt (in ticks) : 4780 and (in seconds) : 0.004780
   * Gives segmentation fault with any of the test cases (1-6) uploaded on the course page. If yes, specify the testcase file name: **None**
4. Specify the language features your lexer or parser is not able to handle (in maximum one line): **N/A**
5. Are you availing the lifeline (Yes/No): **No**
6. Declaration: We, **PIKLU PAUL, RAJABABU SAIKIA, SAURAV VIRMANI, SIDDHANT KHARBANDA, SREYAS RAVICHANDRAN**, declare that we have put our genuine efforts in creating the compiler project code and have submitted the code developed only by our group. We have not copied any piece of code from any source. If our code is found plagiarized in any form or degree, we understand that a disciplinary action as per the institute rules will be taken against us and we will accept the penalty as decided by the department of Computer Science and Information Systems, BITS, Pilani. [Write your ID and names below]

ID: 2017A7PS0006P Name: PIKLU PAUL

ID: 2017A7PS0007P Name: RAJABABU SAIKIA

ID: 2017A7PS0090P Name: SAURAV VIRMANI

ID: 2017A7PS0111P Name: SIDDHANT KHARBANDA

ID: 2017A7PS0275P Name: SREYAS RAVICHANDRAN

Date: 24-02-2020

---------------------------------------------------------------------------------------------------------------------------------------------

Should not exceed 4 pages.