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

Batch No. :

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**Compiler Construction (CS F363)**

**Group Number**

**8**

**II Semester 2021-22**

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

**Coding Details**

**(March 3, 2022)**

1. Team Members Names and IDs

ID 2019A7PS0484P Name: Yash Bansal

ID 2018B3A70815P Name: Sourabh S Yelluru

ID 2018B4A70701P Name: Nihir Agarwal

ID 2018B4A70887P Name: Aakash

1. Mention the names of the Submitted files:
2. lexerDef.h
3. lexer.h
4. lexer.c
5. parserDef.h
6. parser.h
7. parser.c
8. Trie.c
9. Trie.h
10. Stack.c
11. Stack.h
12. driver.c
13. DFA\_structure.txt
14. grammar.txt
15. makefile
16. testcase1.txt
17. testcase2.txt
18. testcase3.txt
19. testcase4.txt
20. testcase5.txt
21. testcase6.txt
22. Total number of submitted files (including copy the pdf file of this coding details pro forma): 21 (All files should be in ONE folder named as Group\_#)
23. Have you compressed the folder as specified in the submission guidelines? (yes/no) yes
24. **Lexer Details:**
    1. Technique used for pattern matching: DFA is implemented as a transition of states upon consuming an input. We check for the longest match in the DFA. If we move from a final state to a dead state, Token is returned. If we go from a non-final state (other than start state) to dead state we return an error.
    2. Keyword Handling Technique: For every **TK\_FUNID** and **TK\_FIELDID** token, we parse a trie to check if it is a keyword and return token appropriately
    3. Hash function description, if used for keyword handling: Trie data structure is used. All keywords are inserted into a Trie, and nodes where a keyword end store the Token Type.
    4. Have you used twin buffer? (yes/ no): Yes
    5. Error handling and reporting (yes/No): Yes
    6. Describe the errors handled by you
       1. Unknown Symbol – When symbol is not part of the language
       2. Unknown Pattern – When Pattern is not valid according to the DFA
       3. Length Exceeding the Limit – When lexeme exceeds prescribed length for **TK\_FUNID** and **TK\_ID**
    7. Data Structure Description for tokenInfo (in maximum two lines): Contains token information - token type, line number, length, start index of the token and the lexeme.
25. **Parser Details:** 
    1. High Level Data Structure Description (in maximum three lines each, avoid giving C definitions used):
       1. grammar: We used a two-dimensional jagged array. The first element of each 1D array stores the LHS of the rule. **Grammar**[i][j+1] stores the jth element of the RHS of the ith production rule. The size of every rule is stored in a different array, indexed by rule number.
       2. FIRST and FOLLOW sets:
       3. parse table: This is implemented as a two-dimensional integer array of size (no. of non-terminals \* no. of terminals). Rows are indexed by the non-terminals and Columns are indexed by terminals
       4. parse tree: (Describe the node structure also): The start symbol becomes the root of the tree. Every tree node which contains a non-terminal has one or more children. In sequence, each child contains either a terminal or a non-terminal, which results from one of its parents’ production rules. Every leaf node contains a terminal. Stack works in tandem with the parse tree maintaining the current state, helping us to traverse the tree.
       5. Any other (specify and describe): We have implemented Trie, Stack and BITSET for utility in our functions.
    2. 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 parse tree nodes are printed through in order traversal of the parse tree.
    3. Grammar and Computation of First and Follow Sets
       1. Data structure for original grammar rules: two-dimensional jagged array
       2. FIRST and FOLLOW sets computation automated (yes /no): Yes
       3. Name the functions (if automated) for computation of First and Follow sets: **populateFirstSets, populateFollowSets.** These are invoked when we load our Parser.
       4. If computed First and Follow sets manually and represented in file/function (name that): They are stored in **parserData** as array of sets **firstSets**, **followSets**. They can be viewed with the functions named **printFirstSets** and **printFollowSets**
    4. Error Handling
       1. Attempted (yes/ no): yes
       2. Describe the types of errors handled:
          1. When top of stack is a non-terminal and the next token is a terminal which is not the First set of the non-terminal, error is reported and handled.
             1. If this next token is in the synchronization set, we pop the non-terminal from the stack and move forward.
             2. If this next token is not in the synchronization set, we discard this token and move forward.
          2. When top of stack is a terminal and the next token is a different terminal, we pop the terminal from the stack.
          3. Lexical errors are reported through error tokens given by the lexer.
26. Compilation Details:
    1. Makefile works (yes/no): yes
    2. Code Compiles (yes/ no): yes
    3. Mention the .c files that do not compile: N/A
    4. Any specific function that does not compile: N/A
    5. Ensured the compatibility of your code with the specified gcc version (yes/no) yes
27. Driver Details: Does it take care of the options specified earlier(yes/no): yes
28. Execution
    1. status (describe in maximum 2 lines): The lexer tokenizes the code and reports errors appropriately in all testcases. The parser parser the source code and builds the parse tree, reporting errors when encountered.
    2. Gives segmentation fault with any of the test cases (1-6) uploaded on the course page. If yes, specify the testcase file name:
29. Specify the language features your lexer or parser is not able to handle (in maximum one line) N/A
30. Are you availing the lifeline (Yes/No): Yes
31. Declaration: We, Yash Bansal, Sourabh S Yelluru, Nihir Agarwal and Aakash declare that we have put our genuine efforts in creating the compiler project code and have submitted the code developed only by us. 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 all of us in our team and we will accept the penalty as decided by the department of Computer Science and Information Systems, BITS, Pilani.

Your names and IDs

Name: Yash Bansal ID 2019A7PS0484P

Name: Sourabh S Yelluru ID 2018B3A70815P

Name: Nihir Agarwal ID 2018B4A70701P

Name: Aakash ID 2018B4A70887P

Date: 3rd March 2022

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

*Not to exceed 3 pages.*