**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**

**Group No.**

**48**

**(February 24, 2020)**

1. IDs and Names of team members  
   ID: 2016B4A70487P Name: Puneet Anand

ID: 2016B1A70703P Name: Mayank Jasoria

ID: 2016B4A70935P Name: Shubham Tiwari

ID: 2016B4A70594P Name: Vibhav Oswal

1. Mention the names of the Submitted files:

1 driver.c 7 parserDef.h 13 stack.c

2 lexer.c 8 linked\_list.c 14 stack.h

3 lexer.h 9 linked\_list.h 15 utils.h

4 lexerDef.h 10 makefile 16 grammar\_new.txt

5 parser.c 11 n\_ary\_tree.c

6 parser.h 12 n\_ary\_tree.h

1. Total number of submitted files: **23 including all test cases and ‘coding details stage 1.pdf’. Otherwise 16 files including the text file specifying grammar rules** (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 (except in the text file for grammar rules)**  
   [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:**
   1. Technique used for pattern matching: **Constructed a DFA for each token that is matched from the source file and combined them to form a single DFA. Implemented the DFA in the lexer.**
   2. DFA implementation (State transition using switch case, graph, transition table, any other (specify): **State transitions using switch case.**
   3. Keyword Handling Technique: **Matched each token corresponding to ‘ID’ in the DFA with the set of pre-hashed keywords to identify whether the resulting token is a keyword or an ID.**
   4. Hash function description, if used for keyword handling: **The hash function used by us was authored by Dan Bernstein. It uses the following recurrence equation for computation of the hash:**hash[i] = hash[i-1]\*33 + 5, with hash[-1] = 5381 as a base case.  
      Finally, the returned value is (Σ hash) mod HASH\_TABLE\_SIZE
   5. Have you used twin buffer? (yes/ no) **Yes**
   6. Lexical error handling and reporting (yes/No): **Yes**
   7. Describe the lexical errors handled by you  
      **Majorly two kinds of errors:  
      (i) For ID, error of encountering tokens having length beyond the max. allowed length of 20 characters  
      (ii) Appearance of tokens that bear no meaning in our language, including characters that are not part of the alphabet of the language (ex. #, ~).**
   8. Data Structure Description for tokenInfo (in maximum two lines):

**Represented by a record data type bearing details of the line number in which it appears,**

* 1. Interface with parser **Yes, the parser calls getNextToken() requesting for the next token, which in turn calls getStream() whenever the lexer needs to read more data from the file.**

1. **Parser Details:** 
   1. **High Level Data Structure Description (in maximum three lines each, avoid giving C definitions used):**
      1. grammar: **Array of Linked Lists, with each node of the linked list corresponding to a terminal or a non-terminal of the right-hand side of the grammar rule, where the non-terminal on the left-hand side is indexed into the array.**
      2. parse table: **Two-dimensional array**
      3. parse tree: (Describe the node structure also) **The parse tree is constructed as an n-ary tree (each node can have 0 or more children).** **Each node contains the symbol, the type of symbol (terminal/non-terminal), line number on which the terminal was encountered (if applicable), the lexeme corresponding to the terminal (if applicable), the depth of the node, and pointers to the parent, left-most child, previous and next siblings.**
      4. Parsing Stack node structure: **Each node contains symbol, type of symbol, and pointer to a tree node (used for n-ary tree construction).**
      5. Any other (specify and describe) **Record for each element that appears on the right-hand side of a grammar rule. It contains the symbol, the type of symbol, and pointer to the next node.**
   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)

**Inorder traversal of the tree (leftmost child -> parent -> all other children in order from left to right, for each node).**

* 1. **Grammar and Computation of First and Follow Sets** 
     1. Data structure for original grammar rules **Array of Linked Lists, with each node of the linked list corresponding to a terminal or a non-terminal of the right hand side of the grammar rule, where the non-terminal on the left hand side is indexed into the array.**
     2. FIRST and FOLLOW sets computation automated (yes /no) **Yes**
     3. Data structure for representing sets **Set for each non-terminal is represented by a 64-bit number, where a set bit denotes the corresponding terminal (which is stored in an array)**
     4. Time complexity of computing FIRST sets **O(G\*R), where G represents number of grammar rules, and R represents length of longest rule (by no. continuous ε-deriving non-terminals)**
     5. Name the functions (if automated) for computation of First and Follow sets **ComputeFirstAndFollowSets(), firstSet(), followSet(), firstFollow(), findinSet(), setUnion()**
     6. If computed First and Follow sets manually and represented in file/function (name that) **NA**
  2. **Error Handling** 
     1. Attempted (yes/ no): **Yes**
     2. Printing errors (All errors/ one at a time): **All errors (with line numbers)**
     3. Describe the types of errors handled

**Syntax: Unexpected terminals, missing terminals, and incorporated reporting of expected terminals, with line numbers.**

* + 1. Synchronizing tokens for error recovery (describe)  
       **Implemented a recovery system based on a combination of first and follow sets of the non-terminal on the top of the stack. We can divide the errors encountered into two categories based on the state of stack and token stream:**

**1. If the top of the stack is a terminal, not matching the input token, then we report the token as missing, report the expected terminal on the top of the stack and keep popping until we either match with an input terminal, or we encounter a non-terminal on the top of the stack.**

**2. If the top of the stack is a non-terminal whose first set does not match with the input token, then we, compute the follow set of the non-terminal on the top of the stack, print them as expected terminals, and keep popping from the top of the stack, until we encounter a non-terminal whose first set matches the input terminal or a matching terminal.**

* + 1. Total number of errors detected in the given testcase t6(with\_syntax\_errors).txt **11**

1. **Compilation Details:**
   1. Makefile works (yes/no): **Yes**
   2. Code Compiles (yes/ no): **Yes**
   3. Mention the .c files that do not compile: **None**
   4. Any specific function that does not compile: **No**
   5. 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** 
   1. status (describe in maximum 2 lines): **Lexer and parser, with the generation of the parse tree, are functional.**
   2. Execution time taken for
      * t1.txt (in ticks) **24.000000** and (in seconds) **0.024000**
      * t2.txt (in ticks) **9.000000** and (in seconds) **0.009000**
      * t3.txt (in ticks) **13.000000** and (in seconds) **0.013000**
      * t4.txt (in ticks) **15.000000** and (in seconds) **0.015000**
      * t5.txt (in ticks) **14.000000** and (in seconds) **0.014000**
      * t6.txt (in ticks) **27.000000** and (in seconds) **0.027000**
   3. Gives segmentation fault with any of the test cases (1-6) uploaded on the course page. If yes, specify the testcase file name: **No**
4. Specify the language features your lexer or parser is not able to handle (in maximum one line)   
   **Able to handle all features specified.**
5. Are you availing the lifeline (Yes/No): **No**
6. Declaration: We, **Puneet Anand, Mayank Jasoria, Shubham Tiwari, and Vibhav Oswal** (your names) 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 2016B4A70487P Name: Puneet Anand

ID 2016B1A70703P Name: Mayank Jasoria

ID 2016B4A70935P Name: Shubham Tiwari

ID 2016B4A70594P Name: Vibhav Oswal

Date: 24th February 2020

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

Should not exceed 4 pages.