**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.**

26

**(February 24, 2020)**

1. IDs and Names of team members

ID: **2017A7PS0048P** Name: **Abhinav Tuli**

ID: **2017A7PS0161P** Name: **Kushaghra Raina**

ID: **2017A7PS0184P** Name: **Tanmay Moghe**

ID: **2017A7PS0224P** Name: **Amratanshu Shrivatsava**

ID: **2017A7PS0225P** Name: **Rohit Bohra**

1. Mention the names of the Submitted files :

1 driver.c 7 parserDef.h 13 t5.txt

2 lexer.c 8 makefile 14 t6.txt

3 parser.c 9 t1.txt 15 t7.txt

4 lexer.h 10 t2.txt 16 grammar.txt

5 lexerDef.h 11 t3.txt 17 SampleParseTree.txt()

6 parser.h 12 t4.txt 18

1. Total number of submitted files: 15 (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** [Note: Files without names will not be evaluated]
3. Have you compressed the folder as specified in the submission guidelines? - **Yes**
4. **Lexer Details:**
   1. Technique used for pattern matching:  *Reading the input character by character in a* ***buffer*** *of size 100 and then traversing through the states of the DFA.*
   2. DFA implementation (State transition using switch case, graph, transition table, any other (specify): *Using* ***switch*** *case.*
   3. Keyword Handling Technique: ***Keyword Hash Table****.*
   4. Hash function description, if used for keyword handling: *Bit shifting and multiplication.*

*Function -> ((hash << 4) + (int)(\*str))*

* 1. Have you used a twin buffer?***No****. We have introduced a buffer check at some places (sufficient). Whenever at these checkpoints the buffer left to be read is less than 20 characters, the remaining characters are copied at the start and the remaining buffer is filled with new characters from the file.*
  2. Lexical error handling and reporting (Yes/No): ***Yes****. After each error, the error is reported and tokens are then read from the next line*.
  3. Describe the lexical errors handled by you - *1. Identifier length more than 20, 2. Any unexpected character, 3. No number after the decimal point, 4. A random underscore or equal-to or dot sign.*
  4. Data Structure Description for tokenInfo (in maximum two lines):

*It has a union which contains either an integer, float, boolean or string. The tag to identify this value. Pointers to the previous and next tokens. An integer for storing the line number. The enum value to identify the term.*

* 1. Interface with parser: *A function named* ***lexerRun****() takes care of everything and returns a pointer to the head of the token linked list. This function is called in the parser file.*

1. **Parser Details:** 
   1. **High Level Data Structure Description (in maximum three lines each, avoid giving C definitions used):**
      1. grammar : *Stored as an array of structures where each structure contains rules corresponding to one non-terminal. Each structure contains one string(containing non-terminal), int variable(number of rules) and a head of linked list of structures(each structure containing a token - can be a terminal or non-terminal). The Linked list is a store the whole rule in a sequential manner.*
      2. parse table - Stored as integer array of Non terminals x Terminals wherein each cell contains :
2. -1 for **Error**
3. Positive integer which denotes the grammar **rule no** which has to be used for that particular combination of NonTerminal and Terminal.
   * 1. parse tree: (Describe the node structure also) - Created as a Trie with every node containing
        1. **tnt** - stores the name of that Terminal OR Non Terminal
        2. **Tag** - 0 for NT, 1 for T
        3. **child** - pointer to the child (on the left of the parent)
        4. **next** - pointer to the next node (to the right on the same level.
     2. Parsing Stack node structure :
        1. **tnt** - stores the name of that Terminal OR Non Terminal
        2. **Tag** - 0 for NT, 1 for T
        3. **next** - pointer to the next node (below in the stack)
     3. Any other (specify and describe) :
     4. **Parse tree**
     5. Constructed (yes/no): **Yes**
     6. Printing as per the given format (yes/no): **Yes**
     7. Describe the order you have adopted for printing the parse tree nodes (in maximum two lines):

The parse tree nodes are printed in order, i.e. for a node, we first print leftmost child, then the

root and then remaining children to the right.

* 1. **Grammar and Computation of First and Follow Sets** 
     1. Data structure for original grammar rules - **ntRules**
        1. **nt** - name of the Non Terminal
        2. **numRules** - no of rules from that non terminal
        3. **struct ruleToken heads[10]** - First T/NT of RHS of each rule (there will be **numRules** heads)
     2. FIRST and FOLLOW sets computation automated (yes /no) **Yes**
     3. Data structure for representing sets - **ntfirstFollow** (storing first-follow of only non terminals)
        1. **nt** - name of that non terminal
        2. **first** - array of names of terminals in first(**nt**) , **numFirsts** - no of terminals in the first
        3. **follows** - array of names of terminals in follow(**nt**), **numFollows** - no of terminals in the follow
     4. Time complexity of computing FIRST sets
     5. Name the functions (if automated) for computation of First and Follow sets

**computeFirstAndFollow()**

* + 1. If computed First and Follow sets manually and represented in file/function (name that) **N/A**
  1. **Error Handling** 
     1. Attempted (yes/ no): **Yes**
     2. Printing errors (All errors/ one at a time) : **One at a time**, Lexical errors are printed on the top.
     3. Describe the types of errors handled: All syntactical errors (generated during parsing) are handled. These arise when the parse table doesn’t have a corresponding rule to the top of stack and the current token read.
     4. Synchronizing tokens for error recovery (describe): Error recovery is done by discarding tokens till we reach a token that corresponds to the first or follow of the topmost non terminal of stack. In case the token found lies in the first of the non terminal, the parsing continues from it. In case the token found lies in the follow of the non terminal, we pop the topmost non-terminal of the stack and then continue parsing.
     5. Total number of errors detected in the given testcase t6(with\_syntax\_errors).txt:

**11 errors (2 lexical and 9 syntax errors)**

1. **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**
2. **Driver Details**: Does it take care of the options specified earlier(yes/no): **Yes**
3. **Execution** 
   1. status (describe in maximum 2 lines): **The driver executes and provides the 4 options as specified. It handles all the 4 options properly and calls appropriate functions.**
   2. Execution time taken for
      * t1.txt (in ticks) **3421** and (in seconds) **0.003421s**
      * t2.txt (in ticks) **5376** and (in seconds) **0.005376s**
      * t3.txt (in ticks) **4801** and (in seconds) **0.004801s**
      * t4.txt (in ticks) **8292** and (in seconds) **0.008292s**
      * t5.txt (in ticks)  **15534** and (in seconds) **0.015534s**
      * t6.txt (in ticks) **5940**  and (in seconds) **0.005940s**
   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, all testcase files work properly.**
4. Specify the language features your lexer or parser is not able to handle (in maximum one line): **None**
5. Are you availing the lifeline: **Yes**
6. Declaration: We, **Abhinav Tuli, Amratanshu Shrivastava, Rohit Bohra, Tanmay Moghe and Kushagra Raina**, 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 **2017A7PS0048P** Name: **Abhinav Tuli**

ID **2017A7PS0224P** Name: **Amratanshu Shrivastava**

ID **2017A7PS0225P** Name: **Rohit Bohra**

ID **2017A7PS0184P** Name: **Tanmay Moghe**

ID **2017A7PS0161P** Name: **Kushagra Raina**

Date: 25/02/2020

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

Should not exceed 4 pages.