**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: 2016B5A70590P Name: Anwesh Bhattacharya

ID: 2017A7PS0147P Name: Deepak Chahar

ID: 2017A7PS0103P Name: Komal Vasudeva

ID: 2016B1A70822P Name: Rohan Kela

1. Mention the names of the Submitted files:

1 driver.c 7 lexerDef.h 13 t3.txt

2 parser.c 8 lexTest.c 14 t4.txt

3 lexer.c 9 grammmar.txt 15 t5.txt

4 lexer.h 10 makefile 16 t6(with\_syntax\_errors).txt

5 parser.h 11 t1.txt

6 parserDef.h 12 t2.txt

1. Total number of submitted files: (All files should be in **ONE folder** named exactly as Group\_#, # is your group number) **16**
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:**
   1. Technique used for pattern matching: DFA
   2. DFA implementation (State transition using switch case, graph, transition table, any other (specify): **Using switch-case**
   3. Keyword Handling Technique: **Global array with** **if-else statements**
   4. Hash function description, if used for keyword handling: **Not used**
   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: **Identifier size, comment , expected =** **sign, digit expected in real numbers, end of file while lexing, single \* in comment, comment ended prematurely, unknown input symbol**
   8. Data Structure Description for tokenInfo (in maximum two lines):

**The data structure for tokenInfo contains enum value of the tokenID, line number and the lexeme value.**

* 1. Interface with parser **parseTree (char \*inFile, char \*outFile)**

1. **Parser Details:** 
   1. **High Level Data Structure Description (in maximum three lines each, avoid giving C definitions used):**
      1. **grammar**: It is an array of structures. Each structure has two elements- “**LHS”** of type int (which stores the enum\_ID of the Non-terminal), and **“RHS**” (which is a pointer to a structure containing enumID of the first generation of the rule, and the pointer to next generation). For example, for S->ABC, LHS stores enum(S), and RHS is a pointer to a structure (which stores enum(A) and a pointer to structure doing the same for B).
      2. **parse table**: It is a 2-D array of type int. Row Numbers and Column Numbers are indexed according to the enumIDs. The array entry contains the rule number of the grammar, and -1 for error.
      3. **parse tree**: (Describe the node structure also): It is a k-ary tree, where every node of the tree has 3 pointers (to its parent, child and right sibling), tokenID and a tag (Terminal or Non-Terminal). If it is a terminal, it further stores information about the lexeme and its line number.
      4. **Parsing Stack node structure**: Stack is implemented using a linked list. Every node of the linked list stores the enum\_ID and a pointer to the next node. It is initialized with a TK\_EOF symbol. Further, push and pop functions have been defined.
   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 (as mentioned in the guidelines)**
   3. **Grammar and Computation of First and Follow Sets** 
      1. Data structure for original grammar rules: **Array of linked lists**
      2. FIRST and FOLLOW sets computation automated (yes /no): **Yes**
      3. Data structure for representing sets : **Linked list**
      4. Time complexity of computing FIRST sets : **O(n\*m) n = no of rules, m = no of grammar symbols (Terminals and non-terminals)**
      5. Name the functions (if automated) for computation of First and Follow sets: **populateFirst(), populateFollow()**
      6. If computed First and Follow sets manually and represented in file/function (name that) : **Same as above**
   4. **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: **Missing terminals are highlighted with a suggestion of the expected token. When the non-terminal on top of the stack does not have a production that produces the lexeme in the source code, we skip the lexeme and wait till a lexeme is found which is the follow of the non-terminal on top of the stack. We then pop the non-terminal and continue parsing.**
      4. Synchronizing tokens for error recovery (describe): **For non-terminals: follow of the non-terminal. For terminals: all other tokens.**
      5. Total number of errors detected in the given testcase t6(with\_syntax\_errors).txt **- 11**
2. **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: **No**
   5. Ensured the compatibility of your code with the specified gcc version(yes/no): **Yes**
3. **Driver Details**: Does it take care of the options specified earlier(yes/no): **Yes**
4. **Execution** 
   1. status (describe in maximum 2 lines): **The code is executing properly and correctly.**
   2. Execution time taken for
      * t1.txt (in ticks - **1410** in seconds - **1.423ms**)
      * t2.txt (in ticks - **643** in seconds - **0.645ms**)
      * t3.txt (in ticks - **570** in seconds - **0.575ms**)
      * t4.txt (in ticks - **568** in seconds - **0.577ms**)
      * t5.txt (in ticks - **614** in seconds - **0.616ms**)
      * t6.txt (in ticks - **593** in seconds - **0.597ms**)
   3. Gives segmentation fault with any of the test cases (1-6) uploaded on the course page. If yes, specify the testcase file name: **None**
5. Specify the language features your lexer or parser is not able to handle (in maximum one line) **It may not be able to handle comments that exceed the size of two buffers.**
6. Are you availing the lifeline (Yes/No): **Yes**
7. Declaration: We, **Anwesh, Deepak, Komal and Rohan,** 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: 2016B5A70590P Name: Anwesh Bhattacharya

ID: 2017A7PS0147P Name: Deepak Chahar

ID: 2017A7PS0103P Name: Komal Vasudeva

ID: 2016B1A70822P Name: Rohan Kela

Date: **24th February 2020**