**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.4**

**(February 24, 2020)**

1. IDs and Names of team members

ID: 2016B1A70280P Name: Akshina Jindal

ID: 2017A7PS0276P Name: Kavya Gupta

ID: 2016B4A70322P Name: Satyavrat Sharma

ID: 2016B4A70509P Name: Nikita N. Singh

1. Mention the names of the Submitted files :
   * + 1. lexerDef.h 5. parser.h 9. makefile 13. t2.txt 17. t6.txt 21. t10.txt
       2. lexer.h 6. parser.c 10. Readme 14. t3.txt 18. t7.txt 22. t11.txt
       3. lexer.c 7. grammar.txt 11. driver.c 15. t4.txt 19. t8.txt
       4. parserDef.c 8. Coding\_details.docx. 12. t1.txt 16. t5.txt 20. t9.txt
2. Total number of submitted files: **22** (All files should be in **ONE folder** named exactly as Group\_#, # is your group number)
3. Have you mentioned your names and IDs at the top of each file (and commented well)? (Yes/ no) **YES**
4. [Note: Files without names will not be evaluated]
5. Have you compressed the folder as specified in the submission guidelines? (yes/no) **YES**
6. **Lexer Details:**
   1. Technique used for pattern matching: *Built a mapping table that maps the char strings(token) to the enum values. Mapping table also keeps track if the string is of a terminal or a non-terminal*
   2. DFA implementation (State transition using switch case, graph, transition table, any other (specify): *Implemented using switch case for state transitions*
   3. Keyword Handling Technique: *Used hash maps for storing keywords. Maps the string (keyword) to the corresponding token*
   4. Hash function description, if used for keyword handling: *The hash function adds the ascii value of each character and position dependent value for the entire string. (* ***index=(index+i\*constant + Ascii(str[i]))%size)*** *.*
   5. Have you used twin buffer? (yes/ no) **NO**
   6. Lexical error handling and reporting (yes/No): **YES**
   7. Describe the lexical errors handled by you :
      1. In case the comment doesn’t end, and end of file is reached.
      2. Maximum length of the identifier exceeded
      3. Digit is absent after a decimal in a floating-point number
      4. Lexer isn’t able to recognize the input ( examples are = )
   8. Data Structure Description for tokenInfo (in maximum two lines): *The* ***Token*** *structure contains the lexeme name, token name and line number for each token.*
   9. Interface with parser: *The* ***getStream*** *function helps integrate parser and lexer. It calls* ***getNextToken*** *and saves the tokenized string in a global Token variable and increments the file pointer. The function* ***parseInput*** *calls* ***getStream*** *repeatedly until the entire token stream is parsed.*
7. **Parser Details:** 
   1. **High Level Data Structure Description (in maximum three lines each, avoid giving C definitions used):**
      1. **grammar :** *The grammar is read from a text file and then an array of linked lists is formed. Each array cell stores the LHS of the grammar i.e a Non-Terminal and pointer to the head of RHS. The RHS is represented using a linked list of nodes(having NT/T(union) enum value, flag and pointer to next T/NT)*
      2. **parse table:** *2D interger array of the size (num\_non-terminal,num\_terminals) containing the grammar rule numbers. Incase of error the rule number is -1.*
      3. **parse tree:** (Describe the node structure also): *Each parse tree node contains a flag to distinguish between T/NT, value(to store enum value), pointer to the leftmost child and rightmost sibling and token info (only in case of a terminal). The parse tree is an n-ary tree, and the empty strings nodes are also represented.*
      4. **Parsing Stack node structure :** *Each stack node has a flag ( is it’s a Terminal or a Non Terminal), a value to store the enum of T/NT, pointer to the next node and the address of the Parse Tree Node corresponding to that Terminal or Non-Terminal*
      5. Any other (specify and describe): *Assumed a $ symbol(end of marker) at the end of the tokenized string.*
   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 nodes. For a given node first the child is printed then that node and thereafter all the siblings of the child. This is done using a recursiev function call* ***display.***
   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: ***2D ARRAYS***
      4. Time complexity of computing FIRST sets ***O(num\_rules\*maxRhsLen)***
      5. Name the functions (if automated) for computation of First and Follow sets: ***findFirst()*** *and* ***findFollow().*** *Also computed the first of the rhs of each rule using* ***fillFirstRhs().***
      6. If computed First and Follow sets manually and represented in file/function (name that) **N/A**
   4. **Error Handling** 
      1. Attempted (yes/ no): **YES**
      2. Printing errors (All errors/ one at a time) : ***Prints errors as and when it encounters one***
      3. Describe the types of errors handled
         1. *Lexical errors (as described above)*
         2. *Syntactic error: The input terminal and the terminal on the top of the stack do not match*
         3. *Syntactic error: Terminal does not occur in both the first and follow sets of the Nonterminal present on the top of the stack*
      4. Synchronizing tokens for error recovery (describe): *Once an error rule is encountered, the tokens are discarded until any terminal in the first of the nonterminal at the top of the stack is found.*
      5. Total number of errors detected in the given testcase t6(with\_syntax\_errors).txt :
         1. *2 Lexical Errors (ERROR 2 and 6)*
         2. *12 Syntactic Errors ( rest all errors mentioned in the file)*

*\*Errors are printed line number wise for both cases.*

1. **Compilation Details:**
   1. Makefile works (yes/no): **YES**
   2. Code Compiles (yes/ no): **YES**
   3. Mention the .c files that do not compile: **ALL COMPILE**
   4. Any specific function that does not compile: **NONE**
   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 code is fully functional. First and follow sets automated and all requirements satisfied.*
   2. Execution time taken for **\*(both lexical and syntactical analysis)**
      * t1.txt (in ticks) 427.000000 and (in seconds) 0.000427 **(only lexical)**
      * t2.txt (in ticks) 512.000000 and (in seconds) 0.000512
      * t3.txt (in ticks) 761.000000 and (in seconds) 0.000761
      * t4.txt (in ticks) 1612.000000 and (in seconds) 0.001612
      * t5.txt (in ticks) 1852.000000 and (in seconds) 0.001852
      * t6.txt (in ticks) 2164.000000 and (in seconds) 0.002164
   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**
4. Specify the language features your lexer or parser is not able to handle (in maximum one line): *The language cannot accept $ as any character or EPS as any identifier. They have been tokenized for end of marker symbol and empty string.*
5. Are you availing the lifeline (Yes/No): **NO**
6. Declaration: We, *Akshina Jindal, Kavya Gupta, Nikita N. Singh, Satyavrat Sharma* 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: 2016B1A70280P Name: Akshina Jindal

ID: 2017A7PS0276P Name: Kavya Gupta

ID: 2016B4A70322P Name: Satyavrat Sharma

ID: 2016B1A70509P Name: Nikita N. Singh

Date: *24th March 2020*

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

Should not exceed 4 pages.