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

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**Compiler Construction (CS F363)**

**II Semester 2022-23**

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

**Coding Details**

**Group No.**

51

**(March 2, 2023)**

1. IDs and Names of team members
2. ID: 2019B2A70966P Name: Arnav Agrawal

ID: 2019B3A70411P Name: Aviral Omar

ID: 2019B4A70634P Name: Chandra Sekhar Reddy E

ID: 2019B5A70697P Name: Vatsal Pattani

1. Mention the names of the Submitted files :

1 driver.c 7 lookupTable.h 13 stack.c 19 tree.h

2 grammar.txt 8 lookupTableDef.h 14 stack.h 20 treeDef.h

3 lexer.c 9 makefile 15 stackDef.h

4 lexer.h 10 parser.c 16 symbolDef.h

5 lexerDef.h 11 parser.h 17 testcase.txt

6 lookupTable.c 12 parserDef.h 18 tree.c

1. Total number of submitted files: **20** (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** [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: **We have used switch case statements where we are using a lookahead pointer to match the pattern.**
   2. DFA implementation (State transition using switch case, graph, transition table, any other (specify): **State transition has been implemented using switch case statements.**
   3. Keyword Handling Technique: **We are matching keywords using a lookup table and for that we are using a hash function.**
   4. Hash function description, if used for keyword handling: **Hash function is implemented by summing up the ascii value of the keyword and subtracting the ascii value of A (Capital) from their sum.**
   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:
5. **Comment ending missing for comment starting.**
6. **Presence of single “=”, “!” and “.” in a particular line (we are reporting the line number as well)**
7. **Identifier or keyword longer than 20 characters**
8. **Incomplete integer or floating-point number is found (Handling minor edge case)**
9. **Invalid character present at a particular line number**

**Additional Warnings:**

1. **Integer is too big; Exponent is too big or Decimal part of float is too big; there is a possibility it may be misinterpreted.**
   1. Data Structure Description for tokenInfo (in maximum two lines):

**TokenInfo contains token which is of the form Token (Enum separately created), unsigned int line number, union lexeme data (containing lexeme with maximum size of 21 chars, int value and a float value)**

* 1. Interface with parser: **We are parsing one token at a time, and we are entering the token info in the parse tree by copying them from the lexer.**

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 which is of the type LexicalSymbol. LexicalSymbol is a structure which contains a next pointer, two-character type variables and data of type Symbol. Symbol is a structure which is a union of nonterminal and terminal tokens of type enums.**
      2. parse table: **Parse Table is a 2D Array consisting of non-Terminals as rows and Terminals as columns. We are storing rule number corresponding to each entry in the parse table. If the entry is equal to -1, this indicates that there is error with respect to that Non-Terminal and Terminal Matching whereas -2 indicates that terminal is present in the synch set of the non-terminal.**
      3. parse tree: (Describe the node structure also): **Parse Tree Node has parent, child and sibling pointers. It has the symbol, lexeme information, line number, value in case of Number or Real Number (in certain cases). Parse Tree is getting printed using in order traversal and we are printing the required columns as specified in the interface details.**
      4. Parsing Stack node structure : **It contains symbol information, type information (Terminal, Non-Terminal or Epsilon), a next pointer (Stack is implemented as a linked list) and a tree node (connected to parse tree node). Stack is a different struct containing top pointer and size information.**
      5. Any other (specify and describe)
   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 ( Root will call its child node which in turn will keep calling the child node unless it encounters NULL. After it has finished with the child calls, it will go and print the sibling attached to that particular node and return back the call to its parent).**

* 1. **Grammar and Computation of First and Follow Sets** 
     1. Data structure for original grammar rules: **Used an array of pointers to a struct named LexemeSymbol which has symbol type, union of Terminal and NonTerminal and pointer to next LexemeSymbol in the rule**
     2. FIRST and FOLLOW sets computation automated (yes /no) **Yes**
     3. Data structure for representing sets: **An array of a struct named FFEntry which has a pointer to head of linked list of terminals in first set and a pointer to head of linked list of terminals in follow set. Terminals in a TerminalInfo struct which has terminal type and pointer to next.**
     4. Time complexity of computing FIRST sets: **O(n), n: no. of rules, takes 6 iterations of all rules**
     5. Name the functions (if automated) for computation of First and Follow sets

**computeFirstAndFollowSets, computeFirstSets, computeFollowSets**

* + 1. If computed First and Follow sets manually and represented in file/function (name that): **No**
  1. **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 tokens in statements**

**Missing tokens in module definition**

**Synchronization after lexical error**

**Tokens/Keywords at inappropriate positions according to grammar**

**Missing semicolons**

**Keywords in place of identifiers**

* + 1. Synchronizing tokens for error recovery (describe):

**SEMICOL has been used been used to synchronize in case of erroneous statements**

**Follow sets of non-terminals have been used in the SYN sets**

* + 1. Total number of errors detected in the given testcase t6(with\_syntax\_errors).txt: **12**

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: **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**

**An extra option is also available to print FIRST sets, FOLLOW sets and Parse Table in appropriate files.**

1. **Execution** 
   1. status (describe in maximum 2 lines): **All the files are compiling successfully and working perfectly. We are receiving the desired output in terms of the functionality that we wanted to implement.**
   2. Execution time taken for
      * t1.txt (in ticks) Parser doesn’t terminate, lexer works so time not calculated
      * t2.txt (in ticks) **1086.000000** and (in seconds) **0.001086**
      * t3.txt (in ticks) **2748.000000** and (in seconds) **0.002748**
      * t4.txt (in ticks) **5040.000000** and (in seconds) **0.005040**
      * t5.txt (in ticks) **5820.000000** and (in seconds) **0.005820**
      * t6.txt (in ticks) **6181.000000** and (in seconds) **0.006181**
   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**
2. Specify the language features your lexer or parser is not able to handle (in maximum one line): **All features are getting implemented successfully by lexer and parser.**
3. Are you availing the lifeline (Yes/No): **Yes**
4. Declaration: We, **Arnav Agrawal**, **Aviral Omar**, **Chandra Sekhar Reddy E**, **Vatsal Pattani** (your names), declare that we have put our genuine efforts into 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 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 name below]

ID: 2019B2A70966P Name: Arnav Agrawal

ID: 2019B3A70411P Name: Aviral Omar

ID: 2019B4A70634P Name: Chandra Sekhar Reddy E

ID: 2019B5A70697P Name: Vatsal Pattani

Date: **03.03.2023**

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

Should not exceed 4 pages.