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)

Group No.
29

---

1. IDs and Names of team members

ID: 2017A7PS0004P	Name: SUBHAM KUMAR DASH
ID: 2017A7PS0036P	Name: RAHUL JHA
ID: 2017A7PS0084P	Name: ANIRUDDHA JAYANT KARAJGI
ID: 2017A7PS0128P	Name: MEET KANANI
ID: 2017A7PS0193P	Name: AYUSH GARG

2. Mention the names of the Submitted files :

1. lexerDef.h	7. hashtableDef.h	13. t3.txt	19. t1.txt
2. lexer.h	8. grammar.txt	14. t4.txt	20. t9.txt
3. lexer.c	9. keywords.txt	15. t5.txt	21. Coding details pro forma
4. parserDef.h	10. makefile	16. t6.txt	
5. parser.h	11. driver.c	17. t7.txt	
6. parser.c	12. t2.txt	18. t8.txt	

3. Total number of submitted files: **21** (All files should be in **ONE folder** named exactly as Group_#, # is your group number)

4. 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]

5. Have you compressed the folder as specified in the submission guidelines? (yes/no) **Yes**

6. **Lexer Details:**

[A]. Technique used for pattern matching: Read character by character and when the final state of the DFA is reached, return the token generated.

[B]. DFA implementation (State transition using switch case, graph, transition table, any other (specify) : **Switch case**

[C]. Keyword Handling Technique: **HashTable**

[D]. Hash function description, if used for keyword handling:
hash_val=(((hash_val*47)%hash->size)+(key[j]%hash->size))%hash->size where hash->size is 41

[E]. Have you used twin buffer? (yes/ no) **<u>No</u>**

[F]. Lexical error handling and reporting (yes/No): **Yes**

[G]. Describe the lexical errors handled by you : We are printing the line number along with the incorrect lexeme which we are getting while doing lexical analysis of the given file and the ERROR token.

[H]. Data Structure Description for tokenInfo (in maximum two lines): A structure containing enum for the terminal, line number, its lexeme and a union for storing the value (integer or float).

[I]. Interface with parser: We have created a function **getNextToken()** which gets called in the parser until End of File is reached.

7. Parser Details:

[A]. High Level Data Structure Description (in maximum three lines each, avoid giving C definitions used):

i. grammar : An **array of structures** represent the LHS portion of each grammar rule. Each of these structures has an **array of linked lists**, each of which represents a rule with the same LHS. The RHS of each rule is represented with the linked lists. A **Trie** has been used for the lookup of the index of Non Terminals and Terminals.

ii. parse table: It is a **2-D array** where each cell contains the rule number corresponding to the rule of a specific non-terminal the cell represents.

iii. parse tree: Nodes are represented using a **tagged UNION** data structure, since each node is either a leaf node or an internal node. Each internal node contains an **array of pointers to its children**.

iv. Parsing Stack node structure : The stack has been implemented using an **array of pointers of TreeNode's** where a TreeNode is the structure representing a node in the parse tree. This allows us to parse and update the parse tree simultaneously.

v. Any other (specify and describe): Our implementation allows the user to use either **inorder** or **level order** traversal to display the parse tree. The level-order output has been implemented using a **queue**. We have used a **Trie** instead of string to enum mapping so that the lookup complexity is reduced from **O(n) to O(length of token)**

[B]. Parse tree

i. Constructed (yes/no):**Yes**

ii. Printing as per the given format (yes/no): **Yes**

iii. Describe the order you have adopted for printing the parse tree nodes (in maximum two lines) :We have provided the functionalities for printing the parse tree in two ways - **inorder and level order traversal**

[C]. Grammar and Computation of First and Follow Sets

i. Data structure for original grammar rules : An **Array of Linked Lists** with each index representing a Non Terminal and each rule of that Non Terminal being represented by a separate linked list in that index.

ii. FIRST and FOLLOW sets computation automated (yes /no) : **Yes**

iii. Data structure for representing sets: A **structure with integer pointers** having sufficient bits to represent all the terminals has been created. Each bit represents a terminal. Insertion is handled using **bit masking**.

iv. Time complexity of computing FIRST sets: **O(NT*NT)**; where NT= number of non terminals and T = number of terminals. (NT+T) is the maximum length of the rule for any non terminal.

v. Name the functions (if automated) for computation of First and Follow sets: **void ComputeFirstAndFollowSets(Grammar* G,FirstAndFollow* F)**;

vi. If computed First and Follow sets manually and represented in file/function (name that): **N/A**

[D]. Error Handling

i. Attempted (yes/ no): **Yes**

ii. Printing errors (All errors/ one at a time) : **One at a time**.

iii. Describe the types of errors handled : Both **Lexical** and **Syntactical** errors.

iv. Synchronizing tokens for error recovery (describe): Using **combination of both Follow and First Sets** for a given Non Terminal.

v. Total number of errors detected in the given testcase t6(with_syntax_errors).txt : We were able to detect all the errors as given the t6.txt file - **11 errors**

8. Compilation Details:

[A]. Makefile works (yes/no): **Yes**

[B]. Code Compiles (yes/ no): **Yes**

[C]. Mention the .c files that do not compile: **N/A**

[D]. Any specific function that does not compile:**No**

[E]. Ensured the compatibility of your code with the specified gcc version(yes/no) : **Yes**

9. Driver Details: Does it take care of the options specified earlier(yes/no): **Yes**

10. Execution

[A]. status (describe in maximum 2 lines): All modules are working correctly with both lexer and parser.

[B]. Execution time taken for

- t1.txt (in ticks) 2469.00 and (in seconds) 0.002469
- t2.txt (in ticks) 2607.00 and (in seconds) 0.002607
- t3.txt (in ticks) 2820.00 and (in seconds) 0.002820
- t4.txt (in ticks) 2894.00 and (in seconds) 0.002894
- t5.txt (in ticks) 3064.00 and (in seconds) 0.003064
- t6.txt (in ticks) 3270.00 and (in seconds) 0.003270

[C]. Gives segmentation fault with any of the test cases (1-6) uploaded on the course page. If yes, specify the test case file name: **N/A**

11. Specify the language features your lexer or parser is not able to handle (in maximum one line): All features specified in the language specifications file have been handled.

12. Are you availing the lifeline (Yes/No): **No**

13. Declaration: We, SUBHAM KUMAR DASH, RAHUL JHA, ANIRUDDHA JAYANT KARAJGI, MEET KANANI, AYUSH GARG of Group 29 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.

ID : 2017A7PS0004P	Name: SUBHAM KUMAR DASH
ID: 2017A7PS0036P	Name: RAHUL JHA
ID : 2017A7PS0084P	Name: ANIRUDDHA JAYANT KARAJGI
ID : 2017A7PS0128P	Name: MEET KANANI
ID : 2017A7PS0193P	Name: AYUSH GARG

Date: 24/02/2020