ParеBIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS

Compiler Construction (CS F363)
II Semester 2021-22
Compiler Project (Stage-1 Submission)
Coding Details
(March 3, 2022)

Group Number
11

1. Team Members Names and IDs

ID: 2019A7PS0104P	Name: Tushar Garg
ID: 2019A7PS0054P	Name: Vikas Balani
ID: 2019A7PS0067P	Name: Ruchir Jain
ID: 2019A7PS0127P	Name: Usneek Singh
ID: 2019A7PS0055P	Name: Abhijith S Raj

2. Mention the names of the Submitted files:

1 lexerDef.h	7 stack.h	13 parser.h
2 lexer.h	8 stack.c	14 parser.c
3 lexer.c	9 treeDef.h	15 driver.c
4 lookupTable.h	10 tree.h	16 grammar.txt
5 lookupTable.c	11 tree.c	17 proforma.pdf
6 stackDef.h	12 parserDef.h	

18. testcase1.txt to testcase12.txt
19. makefile

3. Total number of submitted files (including copy the pdf file of this coding details pro forma): ___30___ (All files should be in ONE folder named as Group_#)

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

5. **Lexer Details:**
   [A]. Technique used for pattern matching: We built a DFA that is handled using switch case statement. We are looking for next character in the buffer and then deciding in which state to land based on that. We have also used retracting technique for the implementation.
   [B]. Keyword Handling Technique: Lookup table using trie data structure
   [C]. Hash function description, if used for keyword handling: We used trie data structure that removed the necessity of the hash function and gives much faster time complexity as we are doing prefix matching instead of searching and matching the complete strings.
   [D]. Have you used twin buffer? (yes/ no): yes
   [E]. Error handling and reporting (yes/No): yes
   [F]. Describe the errors handled by you: Invalid input token, handling length of identifiers, illegal symbols
   [G]. Data Structure Description for tokenInfo (in maximum two lines): It has 5 fields-> token type for specifying type of lexeme, line no to state the line of token, pointer to lexeme for storing the input token, tag as a discriminator for the union used to store the real and int numbers, value to store the value of number

6. **Parser Details:**

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

i. grammar: **grammar** is a structure having (i) array of **rules** and (ii) Total number of symbol lists in grammar. A **rule** corresponds to a non-terminal and has a pointer to head of linked list of **symbol lists.** A symbol list is a linked list of symbols appearing in the grammar. **Symbol node** carries information about the symbols. These data structures together help in retrieving required information from grammar efficiently.

ii. FIRST and FOLLOW sets: The first and follow sets are stored as 2D arrays of size (Total number of non-terminals x Total Number of Terminals). array[i,j] is 1, if j'th Terminal is present in set (first or follow) of i'th non terminal otherwise it is 0.

iii. parse table: Parse table is a 2D array, where each row and column correspond to a non-terminal and a terminal/token respectively. Every non-terminal (from the stack)-terminal (from look-ahead pointer) combination corresponds to a cell from this table, which in turn will point us to the production rule to be applied next.

iv. parse tree: (Describe the node structure also): Parse Tree is a n-ary tree, comprising of tree nodes. **tree** structure stores the pointer to the root node of the created parse tree. Each node of the tree is of the **tNode** structure. tNode contains details regarding the token, pointer to the next sibling in the parse tree and pointer to the first child node. The children (correspond to the expansion of the non-terminal using the grammar rules) are of a linked list form and the length (and head) of the list is stored in the parent node.

v. Any other (specify and describe): Stack has been implemented using linked list. The **stack** structure contains nodes of stack_node in a linked list. It contains the head of the linked list and the size of stack currently. **stack_node** contains a tNode pointer (corresponding to the token that should be pushed on the stack. It also contains a pointer to next stack_node, used to make the linked list.

[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): Nodes are printed in inorder fashion. Leftmost child   parent   remaining children

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

i. Data structure for original grammar rules: Array (of rules) of linked lists (of sub-rules) of linked lists (of symbols in the sub rule).

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

iii. Name the functions (if automated) for computation of First and Follow sets: ComputeFirstAndFollowSets, get_Follow, get_First, init_fnf, printFirstOrFollowSets, combine_sets.

iv. If computed First and Follow sets manually and represented in file/function (name that) : NA

[D]. Error Handling

v. Attempted (yes/ no): yes

vi. Describe the types of errors handled: Errors for non-existent grammar rules, errors for toke not matching in the grammar rule

7. Compilation Details:

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

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

[C]. Mention the .c files that do not compile: NA

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

[E]. Ensured the compatibility of your code with the specified gcc version(yes/no): yes (compatible with gcc 9.3.0 Ubuntu 20.04)

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

9. Execution

[A]. status (describe in maximum 2 lines): All the test cases are giving correct results. There are few differences in errors of t6.txt due to different mode of error recovery.

[B]. Gives segmentation fault with any of the test cases (1-6) uploaded on the course page. If yes, specify the testcase file name: No segmentation faults

10. Specify the language features your lexer or parser is not able to handle (in maximum one line): Not anything to best of our knowledge

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

12. Declaration: We, Tushar Garg, Usneek Singh, Abhijith S. Raj, Ruchir Jain, Vikas Balani, declare that we have put our genuine efforts in creating the compiler project code and have submitted the code developed only by us. 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 all of us in our team and we will accept the penalty as decided by the department of Computer Science and Information Systems, BITS, Pilani.

    Your names and IDs

ID: 2019A7PS0104P	Name: Tushar Garg
ID: 2019A7PS0054P	Name: Vikas Balani
ID: 2019A7PS0067P	Name: Ruchir Jain
ID: 2019A7PS0127P	Name: Usneek Singh
ID: 2019A7PS0055P	Name: Abhijith S Raj

Date: 03/03/2022

Not to exceed 3 pages.