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

Compiler Construction (CS F363)
II Semester 2024-25
Compiler Project
Coding Details
(March 15, 2025)

Group Number :9

1. Team Members Names and IDs
   ID 2022A7PS0169P_____Name__Harsh Shah
   ID 2022A7PS0174P_____Name__Shah Harsh Dhaval
   ID 2022A7PS0119P_____Name__Vrutant Shah
   ID 2022A7PS0005P_____Name__Devansh Nikhil Shah
   ID 2022A7PS0081P_____Name__Siddh Gandhi
   ID 2022A7PS0085P_____Name__Sahej Preet Singh
2. Mention the names of the Submitted files :

1_____driver.c_____	7_____parser.h_____	13 Grammer.txt__
2_____parser.c_____	8_____parserDef.h____	14 out.txt_
3_____lexer.c_____	9_____tree.h_____	15 coding details.docx
4_____stack.c_____	10_____treeDef.h_____	16__t1.txt_____
5_____lexer.h_____	11_____stack.h_____	17__t2.txt_____
6_____lexerDef.h_____	12_____makefile_____	18__t3.txt_____

19 t4.txt
20 t5.txt
21 t6.txt
22 testcase1.txt (Generated by us)
23 testcase2.txt (Generated by us)
24 testcase3.txt (Generated by us)

3. Total number of submitted files (including copy the pdf file of this coding details pro forma) : ____24_____
   (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: **Deterministic Finite Automaton (DFA)** It uses a set of states (denoted by integers) to track the progress of the pattern matching.

   [B]. Keyword Handling Technique: **Hash Table (specifically Hash Map)**
   It is used to efficiently store and lookup predefined keywords.

   [C]. Hash function description, if used for keyword handling: We have multiplied by number 17 (prime number)_because prime numbers helps reduces the number of hash collisions and then returning hashkey % (2*NUMBER_OF_KEYWORDS)

   [D]. Have you used twin buffer? (yes/ no): No
   [E]. Error handling and reporting (yes/No): Yes
   [F]. Describe the errors handled by you: file opening error , memory allocation error , unknown symbol in token recognition , unknown pattern error , overflow in token list , invalid variable length and end of file handling.

[G]. Data Structure Description for tokenInfo (in maximum two lines): The `Token_Info` is a structure that stores information about a token, including its type (`char_terminal`), the corresponding lexeme (`lexeme`), the line number (`line`), and the token's value (either integer or real). It also includes flags to indicate if the token is an integer or a real number (`is_Integer`), and stores the token's actual value (`value.int_Val` or `value.real_Val`).

6. Parser Details:

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

i. grammar : The `grammar` structure holds the entire grammar definition, including rules and the range of non-terminal symbols. Each rule consists of a sequence of terminal and non-terminal symbols. It also stores information about the number of grammar rules and non-terminals in the grammar.

ii. FIRST and FOLLOW sets :
FIRST Set: Stores which terminals can appear at the start of a string derived from a non-terminal. It is represented as a 2D array where each non-terminal points to a set of terminals that could begin strings derived from that non-terminal.
FOLLOW Set: Represents which terminals can appear immediately after a non-terminal in any derivation. It is also stored in a 2D array where each non-terminal points to a set of terminals that could follow it.

iii. parse table: The `parse_table` is a 2D table where each entry corresponds to a cell for a non-terminal and a terminal symbol. The value at each cell represents the grammar rule to apply when parsing the corresponding non-terminal and terminal symbol pair. If no rule is applicable, the cell contains `-1`, indicating an error or mismatch.

iv. parse tree: (Describe the node structure also):
The `parse_tree` is a tree structure where each node represents a syntactic element (either a terminal or non-terminal). Each `Tree_Node` contains:

- **Terminal nodes**: Hold tokens (such as keywords, identifiers, or operators) from the input.
- **Non-terminal nodes**: Represent grammar symbols that are further expanded using the rules. Each node has a parent-child relationship and can have multiple children depending on the grammar rule being applied.

v. Any other (specify and describe)
Stack: Used during the parsing process to manage the current position in the parse tree. The stack stores nodes of the parse tree, allowing the parser to backtrack or explore different derivations.
Error_Record: Keeps track of parsing errors, storing error messages and line numbers for reporting.

[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): The `printNode` function adopts a **depth-first, left-to-right traversal** for printing parse tree nodes. It recursively processes the leftmost child before printing the current node's information, then proceeds to the remaining children in order.

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

i. Data structure for original grammar rules: Hash map where non-terminals are keys and their corresponding production rules are stored as lists of symbols.
ii. FIRST and FOLLOW sets computation automated (yes /no): yes
iii. Name the functions (if automated) for computation of First and Follow sets:
compute_first():- for FIRST set computation
compute_follow():- for FOLLOW set computation

iv. If computed First and Follow sets manually and represented in file/function (name that): We have automated it so not applicable

[D]. Error Handling

v. Attempted (yes/ no): yes

vi. Describe the types of errors handled : file handling errors , memory allocation errors , lexical errors , parsing errors and panic mode

7. Compilation Details:
   [A]. Makefile works (yes/no): yes
   [B]. Code Compiles (yes/ no): yes
   [C]. Mention the .c files that do not compile: None (All files will compile)
   [D]. Any specific function that does not compile: None (All functions will compile)
   [E]. Ensured the compatibility of your code with the specified gcc version (yes/no): yes

8. Driver Details: Does it take care of the options specified earlier(yes/no): yes
9. Execution
   [A]. status (describe in maximum 2 lines): there are no segmentation faults and the test cases are running perfectly with no errors
   [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 fault

10. Specify the language features your lexer or parser is not able to handle (in maximum one line): It is able to handle everything as specified.

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

12. Declaration: We, Harsh Shah, Shah Harsh Dhaval, Vrutant Shah, Devansh Nikhil Shah, Siddh Gandhi and Sehaj Preet Singh 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

Name:Harsh Shah ID: 2022A7PS0169P
Name:Shah Harsh Dhaval ID: 2022A7PS0174P
Name:Vrutant Shah ID: 2022A7PS0119P
Name:Devansh Nikhil Shah ID: 2022A7PS0005P
Name:Siddh Gandhi ID: 2022A7PS0081P
Name:Sehaj Preet Singh ID: 2022A7PS0085P

Date: 15 March 2025
