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
10

1. Team Members Names and IDs
   ID: 2022A7PS0087P Name: Akshay Shukla
   ID: 2022A7PS0083P Name: Gobind Singh
   ID:2022A7PS0172P Name: Granth Jain
   ID: 2022A7PS0147P Name: Sriram Hebbale
   ID: 2022A7PS0070P Name: Siddhartha Gotur
2. Mention the names of the Submitted files :
   1. driver.c
   2. lexer.c
   3. parser.c
   4. lexer.h
   5. lexerDef.h
   6. parser.h
   7. parserDef.h
   8. common.h
   9. makefile
   10. testcases(t1.txt – t6.txt)
   11. parser_testcase1.txt
   12. final_grammar_rules.txt
   13. coding_details.pdf
   14. lexical_testcase1.txt

3. Total number of submitted files (including copy the pdf file of this coding details pro forma) : 19 (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-based Lexical Analysis
   [B]. Keyword Handling Technique: Combination of Linear search for Keyword recognition and Hashing for identifier Storage
   [C]. Hash function description, if used for keyword handling: The hash function in lexer.c uses the DJB2 algorithm, initializing a hash value at 5381 and updating it for each character using hash = hash * 33 + c. This ensures a uniform distribution of hash values, minimizing collisions. The final hash is taken modulo table size to fit within the symbol table.
   [D]. Have you used twin buffers? (yes/ no): Yes
   [E]. Error handling and reporting (yes/No): Yes
   [F]. Describe the errors handled by you: Errors handled are as follows-
      1. Unknown symbols: Characters that do not belong to the language are reported as errors when they are read in the input file.
      2. Unknown Pattern: A string which does not contain a valid sequence of characters based on the token rules given is reported as an error.
      3. Variable Identifier exceeding length of 20 characters is reported as an error.
      4. Function Identifier exceeding length of 30 characters is reported as an error.
   [G]. Data Structure Description for tokenInfo (in maximum two lines): tokenInfo is called as SymbolItem in our code. It contains attributes lexeme(string), token(Terminal - enum), type(string), iVal(int), fVal(float), lineCount(int), eof(int) and a self referential pointer to the next SymbolItem

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

i. grammar : Represented using ProductionRule, where each rule has a NonTerminal pointing to a linked list of RHSNode. Each RHSNode represents a terminal or non-terminal in the production's right-hand side. The next_rule pointer allows handling multiple productions for a single non-terminal.
ii. FIRST and FOLLOW sets: Managed using HashSet, storing terminal symbols appearing first in derivations (FIRST). FOLLOW stores terminals that can appear immediately after a non-terminal. These sets help in constructing the ParseTable for predictive parsing.
iii. parse table: A 2D mapping where rows represent NonTerminal and columns represent Terminal, storing the corresponding ProductionRule. This is used to decide which rule to apply based on the current NonTerminal and lookahead Terminal. The table enables efficient LL(1) parsing.
iv. parse tree: (Describe the node structure also) The ParseTreeNode structure represents each node with fields for symbol type, lexeme, and line number. It has pointers to child nodes forming a hierarchical tree structure. This tree is built during parsing and helps in syntax verification and further processing.
v. Any other (specify and describe) A stack implemented using stackNode is used for non-recursive parsing. It stores stackNode pointers and assists in top-down parsing. The stack ensures proper derivation order and helps in backtracking if needed.

[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 used Inorder Traversal to print the nodes in the tree, following the method specified in the instructions.

[C]. Grammar and Computation of First and Follow Sets
i. Data structure for original grammar rules: an Array of Linked lists
ii. FIRST and FOLLOW sets computation automated (yes /no): Yes
iii. Name the functions (if automated) for computation of First and Follow sets: computeFirstSet() and computeFollowSet()
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: Following is the heuristic we have followed for error handling. If the parser looks up entry M[A, a] and finds that it is blank, then the input symbol a is skipped. If the entry is “synch," then the nonterminal on top of the stack is popped in an attempt to resume parsing. If a token on top of the stack does not match the input symbol, then we pop the token from the stack, as mentioned above. Note that the synch set for any non-terminal A contains all those terminals in Follow(A) if that particular entry is empty. All entries that are present in a synch set are represented by token value -1 in the parse table

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
[D]. Any specific function that does not compile: None
[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): Finishes execution for all the given test files and creates a output file with the inorder traversal of the parse tree. Prints LineNumber, Lexeme and Token on selection of Option 2. Option 3 prints: Running the lexical analyser. Printing only the errors generated by the lexical analyser. The syntax analyser is running now. The output parse tree will be stored in the file parseTreeOutFile.txt. Done generating the parse tree. Prints Total CPU time: 38350.000000. Total CPU time in seconds: 0.038350 for OPTION 4.

[B]. Gives segmentation fault with any of the test cases (1-6) uploaded on the course page. If yes, specify the testcase file name: None. Executes properly.

10. Specify the language features your lexer or parser is not able to handle (in maximum one line) It's able to handle all the features that are mentioned in the grammar and also does panic mode recovery.

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

12. Declaration: We, Gobind Singh, Akshay Shukla, Granth Jain, Sriram Hebbale, Siddhartha Gotur (your names) 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: Akshay Shukla ID: 2022A7PS0087P
Name: Gobind Singh ID: 2022A7PS0083P
Name: Granth Jain ID: 2022A7PS0172P
Name: Sriram Hebbale ID: 2022A7PS0147P
Name: Siddhartha Gotur ID: 2022A7PS0070P

Date: 15-03-2025

Not to exceed 3 pages.