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

Compiler Construction (CS F363)
II Semester 2023-24
Compiler Project
Coding Details
(March 5, 2024)

Group Number
1

1. Team Members Names and IDs

ID: 2021A7PS2421	Name: Hardik Gupta
ID: 2021A7PS0661	Name: Yash Pandey
ID: 2021A7PS1457	Name: Achinthya Hebbar S
ID: 2021A7PS2427	Name: Ujjwal Aggarwal
ID: 2021A7PS2215	Name: Vansh Anil Agrawal

2. Mention the names of the Submitted files :

1. lexer.c	9. transition_table_generator.c	17. t6.txt
2. lexer.h	10. transition_table.c	18. testcase01.txt
3. lexerDef.h	11. symbol_table.c	19. testcase02.txt
4. parser.c	12. t1.txt	20. testcase03.txt
5. parser.h	13. t2.txt	21. testcase04.txt
6. parserDef.h	14. t3.txt	22. coding details.pdf
7. makefile	15. t4.txt	23. grammar.md
8. driver.c	16. t5.txt	

3. Total number of submitted files (including copy the pdf file of this coding details pro forma) : 23

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

5. **Lexer Details:**
   - [A]. Technique used for pattern matching: First the 128 bit character space is reduced to 36 character types. The DFA's transition table is implemented in a space efficient manner, using the system of a check, next and default array, with a function to generate the transition table and then use it to get the next state from current state and action. This takes 3-times less space than implementing a 2D array for the transition table.
   - [B]. Keyword Handling Technique: Entries for keywords are inserted prior to the execution of the lexer in the symbol table. For tokens that need to be inserted in the symbol table such as TK_FIELDID & TK_ID, we perform a lookup for the lexeme in the symbol table before inserting the entry. For TK_FIELDID, the identified lexeme could correspond to a keyword leading to the lookup returning that entry, whose line-number is changed to current line number and returned.
   - [C]. Hash function description, if used for keyword handling: **Hash is calculated by iterative operations of characters modulo the hash map size.**
   - [D]. Have you used twin buffers? Yes
   - [E]. Error handling and reporting: Yes
   - [F]. Describe the errors handled by you: Lexeme exceeding maximum length for identifiers and function identifiers, invalid characters, invalid patterns (not recognised by DFA), essentially all the possible errors in lexical analyser are handled.
   - [G]. Data Structure Description for tokenInfo (in maximum two lines): Contains enum of tokentype, line-number, lexeme and int-value & double-value. Int-value or double-value are only populated if the token is of type TK_INT/TK_REAL respectively and this data structure is inserted in the symbol table while a pointer to it is returned by the lexer.

6. Parser Details:

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

i. grammar : For each non terminal, a 2D array for various productions, and in each production an array of symbols (which might be a non terminal or a terminal). Each non-terminal or terminal is present in a variable data structure that has an enum value and a flag where flag is 0 for a terminal variable and 1 for a non-terminal.

ii. FIRST and FOLLOW sets: The sets were represented as boolean arrays where each boolean value represents whether or not the token exists in that set, along with a linked list which helps with faster traversal. Thus using both linked lists and arrays, we can efficiently perform lookup and traversal. Boolean arrays help in checking the presence or absence of terminals in the set in O(1) time while linked lists are efficient for traversing through all terminals in the set.

iii. Parse table: A 2-dimensional table between Non Terminals and Terminals where each entry in the table is an array which represents a pointer to grammar rule (or error or synch).

iv. Parse tree: (Describe the node structure also): The parse tree ADT (abstract data structure) contains a root node which is a pointer to a struct tree_node. Each struct of type tree_node contains the following fields: data of type variable, pointers to the head, next and parent. Head represents the first child from the left of the node (NULL if it is a terminal node) and next represents its sibling while parent is a pointer to the node's parent node (NULL if root node)

[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): Inorder traversal. Each printing is done in the following format with appropriate space justification (Lexeme Current Node) (Line No) (Token Name) (Value if number) (Parent Node Symbol) (IS leaf node) (Node symbol). If the values for Lexeme Current Node, Token name, Value if number does not exist then —-- is printed. Node Symbol is either the Non terminal or LEAF.

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

i. Data structure for original grammar rules: Array of arrays of Variables where each Variable has a value (enum of Non-terminal/Terminal) and a flag where flag is 1 for terminals and 0 for nonterminals. An array was used instead of a linked list for the grammar since it can be statically loaded as in the case of an array compared to dynamically creating the linked list every time a program runs.

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

iii. Name the functions (if automated) for computation of First and Follow sets: computeFirstAndFollow

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

[D]. Error Handling

v. Attempted (yes/ no): Yes

vi. Describe the types of errors handled:

1. Lexical Errors: Lexeme exceeding maximum length for identifiers and function identifiers, invalid characters, invalid patterns (not recognised by DFA).

2. Syntactical Errors: Blank entry in parsing table - Handled by using a synch set populated with the follow of corresponding non-terminal and standard delimiters such as TK_OP, TK_CL, TK_SEM, TK_DOT and skipping tokens until a token in the first set (valid rule exists) or synch set is encountered, in which case the non-terminal is popped from the stack and the parsing continues from the token if it was in synch set or from the next token if it was in the first set.

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: NA
[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: Both lexical analyser and syntax analyser execute successfully with each option of the driver working as expected and described in the interface mails. The lexemes and tokens for t1.txt and t2.txt match exactly with the provided files and test cases t4 & t5 give a completely correct parse tree with no errors and t6 gives errors, all as expected.
   [B]. Gives segmentation fault with any of the test cases (1-6) uploaded on the course page. If yes, specify the test case file name: NA

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

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

12. Declaration: We, Hardik Gupta, Yash Pandey, Achinthya Hebbar S, Ujjwal Aggarwal and Vansh Anil Agrawal 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: Hardik Gupta ID: 2021A7PS2421
Name: Yash Pandey ID: 2021A7PS0661
Name: Achinthya Hebbar S ID: 2021A7PS1457
Name: Ujjwal Aggarwal ID: 2021A7PS2427
Name: Vansh Anil Agrawal ID: 2021A7PS2215

Date: 05/03/2024
