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, 2022)

Group Number
15

1. Team Members Names and IDs

ID:	2021A7PS2430P	Name:	Dev Kulkarni
ID:	2021A7PS2534P	Name:	Radhey Kanade
ID:	2021A7PS2539P	Name:	Shardul Shingare
ID:	2021A7PS2540P	Name:	Shantanu Ambekar
ID:	2021A7PS0001P	Name:	Yash Mundada
ID:	2021A7PS2222P	Name:	Aaryan Garg

2. Mention the names of the Submitted files:

1.	Makefile	7.	parser.c	13.	stack.h	19.	t1.txt
2.	driver.c	8.	parser.h	14.	symbol_table.c	20.	t2.txt
3.	grammar.txt	9.	parserDef.h	15.	symbol_table.h	21.	t3.txt
4.	lexer.c	10.	parsetree.c	16.	testcase1.txt	22.	t4.txt
5.	lexer.h	11.	parsetree.h	17.	testcase2.txt	23.	t5.txt
6.	lexerDef.h	12.	stack.c	18.	testcase3.txt	24.	t6.txt

3. Total number of submitted files (including copy the pdf file of this coding details pro forma): 25

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 have used a switch case statement in which every case represents a state in our DFA, where we always start from state 0 and based on the characters encountered while scanning the input, we transition between the states till a pattern is matched, and thus recognize tokens.

[B]. Keyword Handling Technique: Before reading any input, we insert all keywords into our symbol table and then whenever we get a lexeme of type TK_FIELDID or TK_FUNID, we first check if it is present in the symbol table. If it is present, this indicates that it is a keyword and we set the token for our lexeme to that of the keyword.

[C]. Hash function description, if used for keyword handling: For a lexeme, first we initialize the hash value to a constant. Then we iterate over each character of the lexeme and shift the current hash value by 5 bits and add it to itself. Then we add the ASCII value of the character to this sum and take the modulus with 197 to get the final index in the symbol table.

[D]. Have you used twin buffer? (yes/ no) Yes

[E]. Error handling and reporting (yes/No): Yes

[F]. Describe the errors handled by you: Unknown pattern (A sequence of valid characters which does not match any token specification), Unknown symbol (A symbol which is not a part of the language), Length longer than prescribed length (Length of lexeme is more than the maximum specified length).

[G]. Data Structure Description for tokenInfo (in maximum two lines): Has 5 fields: tokName name is an enum that stores the token name for the lexeme, int lineNo stores the line number on which the token was found, char* string contains a pointer to the actual lexeme, int integer contains the value if token is TK_NUM, else is set to -1 and float realNum contains the value if token is TK_RNUM, else is set to -1.

6. Parser Details:

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

i. grammar: We have used an array of linked lists; every index of this array contains a linked list in which the first node represents the LHS of the grammar rule i.e. a non-terminal and every node after this represents all its RHS derivations i.e. sequence of terminals and non-terminals.

ii. FIRST and FOLLOW sets: The struct FIRSTFOLLOW contains a pointer to a linked list which contains the first or follow set symbols, an integer indicating if the first or follow set has epsilon or not, an integer representing if the first or follow set has been calculated or not.

iii. parse table: We are using a two-dimensional array of pointers to PNODE structs. Each PNODE struct represents a cell of the parse table and stores a pointer to the next node to be parsed, an integer indicating if the cell represents a synchronization entry or not, and an integer indicating if the cell represents an error in parsing or not.

iv. parse tree (Describe the node structure also): The ParseTree structure contains a pointer to the root of the parse tree. Each node is represented by a Node data structure which contains the terminal or non-terminal associated with that node, a pointer to an array of pointers to Node structures which represents the children nodes of the current node and the number of children nodes the current node has.

v. Any other (specify and describe): -

[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 in which each node's child nodes are recursively traversed before visiting the node itself, ensuring a left-root-right sequence across all nodes in the tree.

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

i. Data structure for original grammar rules: We have used an array of linked lists; the first node of each linked list represents the LHS of the grammar rule and every node after this represents all its RHS derivations.
ii. FIRST and FOLLOW sets computation automated (yes /no) Yes
iii. Name the functions (if automated) for computation of First and Follow sets: find_first_set and findFollowSet
iv. If computed First and Follow sets manually and represented in file/function (name that): -

[D]. Error Handling

i. Attempted (yes/ no): Yes
ii. Describe the types of errors handled: Terminal on top of stack does not match with expected token and Token does not match with non-terminal at the top of the stack.

7. Compilation Details:
   [A]. Makefile works (yes/no): Yes
   [B]. Code Compiles (yes/ no): Yes
   [C]. Mention the .c files that do not compile: -
   [D]. Any specific function that does not 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): The lexer is processing the source code into tokens, the parser is verifying the syntax and constructing the parse tree and the errors recovery is being done as expected.
   [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

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

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

12. Declaration: We, Dev Kulkarni, Radhey Kanade, Shardul Shingare, Shantanu Ambekar, Yash Mundada and Aaryan Garg 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	Dev Kulkarni	ID	2021A7PS2430P
Name	Radhey Kanade	ID	2021A7PS2534P
Name	Shardul Shingare	ID	2021A7PS2539P
Name	Shantanu Ambekar	ID	2021A7PS2540P
Name	Yash Mundada	ID	2021A7PS0001P
Name	Aaryan Garg	ID	2021A7PS2222P

Date: 05-03-2024
