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

Group No.
11

Compiler Construction (CS F363)
II Semester 2022-23
Compiler Project (Stage-1 Submission)
Coding Details
(March 2, 2023)

---

1. IDs and Names of team members

ID: 2020A7PS0297P NAME: Sarang Sridhar

ID: 2020A7PS0995P NAME: Kashish Mahajan

ID: 2020A7PS0993P NAME: Satvik Sinha

ID: 2020A7PS0036P NAME: Aarya Attrey

ID: 2020A7PS0017P NAME: Urvashi Sharma

2. Mention the names of the Submitted files :

1 makefile	7 parserDef.h	13 treeDef.h	19 testcase5.txt
2 driver.c	8 stackAdt.h	14 grammar.txt	20 testcase6.txt
3 lexer.c	9 stackAdtDef.h	15 testcase1.txt	21 parser.h
4 lexer.h	10 stackAdt.c	16 testcase2.txt	22
5 lexerDef.h	11 tree.c	17 testcase3.txt	23
6 parser.c	12 tree.h	18 testcase4.txt	24

3. Total number of submitted files: ___**21**_____ (All files should be in **ONE folder** named exactly as Group_#, # is your group number)

4. Have you mentioned your names and IDs at the top of each file (and commented well)? **Yes** Note: Files without names will not be evaluated]

5. Have you compressed the folder as specified in the submission guidelines? **Yes**

6. Lexer Details:

[A]. Technique used for pattern matching: **Using DFA and Retraction**.

[B]. DFA implementation (State transition using switch case, graph, transition table, any other (specify):

Switch Case.

[C]. Keyword Handling Technique**: Initializing hash table for keywords with linear probing, and searching lexemes from the hash table**

[D]. Hash function description, if used for keyword handling:

H(str) = (str[0] + str[1] + str[2] + str[3] …..+str[n]) % 97

Where str is a string of length n.

[E]. Have you used twin buffer? (yes/ no): **yes**

[F]. Lexical error handling and reporting (yes/No): **yes**

[G]. Describe the lexical errors handled by you: **invalid lexeme length, invalid pattern, invalid Symbols.**

[H].Data Structure Description for tokenInfo (in maximum two lines): **A structure that contains tokens' line no. , an enumerated token name, corresponding lexemes, int and float values in case of numbers, and a flag that denotes whether the token has an error.**

[I]. Interface with parser: **Parser gets token by token from lexer by calling getNextToken()**

7. Parser Details:

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

i. grammar: An array of doubly linked lists. Each node of the linked list contains a flag for whether it is a terminal, a forward link to the next node, a backward link to the previous node, and a string for the name of the terminal/non terminal.

ii. parse table : 2D array of a struct (pointer to the grammar rule). Struct contains a flag for whether it is a terminal, a forward link to the next node, a backward link to the previous node, and a string for the name of the terminal/non terminal.

iii. parse tree: (Describe the node structure also) Each tree node contains a variant which stores data for the token(lexeme/integer/float values), line no, a string for token name, a flag for whether it is a terminal, node pointers to its left child, parent, left sibling(prevSibling) and right sibling(nextSibling).

iv. Parsing Stack node structure : Each node in the parsing stack is a C language struct which contains a flag for whether it is a terminal, a pointer to the node below it in the stack and a string which stores the name of the terminal/non-terminal.

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): **Inorder Traversal was adopted with the respective fields getting printed for nonTerminals(if defined) and terminals(if defined)**

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

i. Data structure for original grammar rules: **Array of Doubly Linked List**, where each index in the array stores the head of the DLL .

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

iii. Data structure for representing sets: **Implemented using arrays only with the help of a function for removing duplicates (removeDuplicates) for union operation**

iv. Time complexity of computing FIRST sets **O(No. of rules*max length of rule)**

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

createFirst(): to create the first set of non-terminals and terminals.

createFollow(): to create the follow set of non-terminals and terminals.

computeFirstandFollow(): to create the set of first and follow(if rule derives epsilon) for each rule in order to populate the parse table.

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

[D]. Error Handling

i. Attempted (yes/ no): **yes**

ii. Printing errors (All errors/ one at a time):

All errors (lexical errors not printed along with parser errors, included in token list only)

Line number for error is obtained by the next matched terminal in the program

iii. Describe the types of errors handled:

1) **Terminal Mismatch**
2) **Rule missing in Parse Table**
3) **Lexical Errors**
4) **Input finished but stack not empty**
5) **Stack empty but input not yet finished**

iv. Synchronizing tokens for error recovery (describe)

1) Follow set of the last expanded nonTerminal was used for error recovery(in case of missing rule in parse table)

2) Popping from stack until terminal matches (in case of Terminal Mismatch)

3) Line number for error is obtained by the next matched terminal in the program

v. Total number of errors detected in the given testcase t6(with_syntax_errors).txt

All errors detected

8. 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**

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

10. Execution

[A]. status (describe in maximum 2 lines): **Error-free compilation and execution.**

[B]. Execution time taken for

- t1.txt (in ticks) 1504 and (in seconds) 0.001504 (time for lexer)

- t2.txt (in ticks) 3951 and (in seconds) 0.003951
- t3.txt (in ticks) 3054 and (in seconds) 0.003054
- t4.txt (in ticks) 4936 and (in seconds) 0.004936
- t5.txt (in ticks) 2928 and (in seconds) 0.002928
- t6.txt (in ticks) 3137 and (in seconds) 0.003137

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

11. Specify the language features your lexer or parser is not able to handle (in maximum one line): **Type checking(for a few constructs) inside the expression will be handled at the Semantics Analysis phase.**

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

13. Declaration: We, Sarang Sridhar, Kashish Mahajan, Satvik Sinha, Aarya Attrey, and Urvashi Sharma , declare that we have put our genuine efforts into creating the compiler project code and have submitted the code developed only by our group. 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 disciplinary action as per the institute rules will be taken against us and we will accept the penalty as decided by the department of Computer Science and Information Systems, BITS, Pilani. [Write your ID and name below]

ID: 2020A7PS0297P NAME: Sarang Sridhar

ID: 2020A7PS0995P NAME: Kashish Mahajan

ID: 2020A7PS0993P NAME: Satvik Sinha

ID: 2020A7PS0036P NAME: Aarya Attrey

ID: 2020A7PS0017P NAME: Urvashi Sharma

Date: 02-03-23

Should not exceed 4 pages.