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

Group No.

10

Compiler Construction (CS F363)
II Semester 2019-20
Compiler Project (Stage-1 Submission)
Coding Details
(February 24, 2020)

---

1. IDs and Names of team members

ID: 2017A7PS0143P Name: Ujjwal Gandhi

ID:2017A7PS0101P Name: Atmadeep Banerjee

ID:2017A7PS0179P Name: Mayank Jain

ID:2017A7PS0157P Name: Aditya Mithal

2. Mention the names of the Submitted files :

1	lexerDef.h	7	parser.c	13	t2.txt
2	lexer.h	8	Makefile	14	t3.txt
3	parserDef.h	9	grammar.txt	15	t4.txt
4	parser.h	10	first.txt	16	t5.txt
5	driver.c	11	follow.txt	17	t6.txt
6	lexer.c	12	t1.txt	18	PRO FORMA

3. Total number of submitted files: **18** (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/ no): YES
[Note: Files without names will not be evaluated]

5. Have you compressed the folder as specified in the submission guidelines? (yes/no): YES

6. Lexer Details:

[A]. Technique used for pattern matching: Maximal munch principle

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

[C]. Keyword Handling Technique: Populated symbol table (using hash function) for all keywords and checked during matching identifiers

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

Hash = Sum of (i * ASCII(char_i)) % 199 for all i in 0 <= i < length(string)

[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: Identifiers of length > 20, operators which do not have any meaning (Eg., **=** or **!**), lexemes of type (12.03e#) where # is not +/-/digit (Error – 12.03e and processed # onwards), lexemes of type (12.#) where # is not a digit or . (For rangeop operator, Error - .(DOT) as 12 is matched as a NUM)

[H]. Data Structure Description for tokenInfo (in maximum two lines):

Token (enum value assigned to it), Lexeme (char array), line number, value (UNION of float and int for RNUM and NUM resp.), 1-bit tag (for the union)

[I]. Interface with parser: Parser calls the function **getNextToken()** with appropriate arguments which returns the token data structure for ONE token to it.

7. Parser Details:

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

i. grammar : Array of linked lists being populated from **grammar.txt**. For each non-terminal having n rules, it occupies n elements in the array. Each array element points to a linked list, which is, LHS (non-terminal) followed by its RHS. An array (**indexed by enum values**) stores the first occurrence index of each non-terminal in grammar table. Each node in the linked contains an enum value, tag for terminal/non-terminal and pointer to the next node.

ii. parse table: A 2D array of size N X M (N→ no. of non-terminals, M→ no. of terminals). Each element array[i][j] stores the **rule number (index of grammar table)** which should be used for ith non-terminal on stack with jth terminal in input.

iii. parse tree: (Describe the node structure also): Constructed an n-ary tree with a node's children being the non-terminals/terminal derived from it. The structure of node is lexeme, line number, enum value (token), value (for NUM/RNUM), symbol (token name), isLeaf, pointer to parent node and array of pointers to children.

iv. Parsing Stack node structure : enum value, tag for non-terminal/terminal, pointer to tree node and pointer to next stack node.

v. Any other (specify and describe): Implemented a lookup table to get the enum value directly from the string using **HashNode**, containing the string, enum value, tag for terminal/non-terminal and link to next node for open chaining.

[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):

In-order traversal of the tree. Print of each node is →

Lexeme (--- for non-terminals), line number, token (enum value), value (--- except for NUM/RNUM), parent node symbol, isLeaf, symbol (the token name, for eg., **ID** for an identifier)

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

i. Data structure for original grammar rules: Array of linked lists, as described in 7 [A] i.

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

iii. Data structure for representing sets: structure containing 64-bit (**uint64_t**) integer for binary representation (bit i =1 indicates presence of terminal with enum value i in the set)

iv. Time complexity of computing FIRST sets: O(n*m) where **n** is the number of rules in grammar.txt and **m** is the maximum length of a rule.

v. Name the functions (if automated) for computation of First and Follow sets: getFirst() (for one non-terminal), populateFirst() (calls getFirst for all non-terminals), follow() (for one non-terminal), allFollow() (for populating follow for all non-terminals)

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

[D]. Error Handling

i. Attempted (yes/ no): YES

ii. Printing errors (All errors/ one at a time) : All errors (for each error in input file, printing an error for each token till the error is resolved)

iii. Describe the types of errors handled: **Lexical errors** (as described earlier), **token mismatch** (top of stack terminal and input terminal don't match), **no rule found** (top of stack non-terminal doesn't correspond to the token found in the input string)

iv. Synchronizing tokens for error recovery (describe): For **token mismatch**, pop the stack and skip the token in the input. For **rule not found**, use **panic mode** (keep skipping the tokens in the input file until you encounter one which corresponds to either the first set or the follow set of the non-terminal on top of stack)

v. Total number of errors detected in the given testcase t6(with_syntax_errors).txt:
Successfully resolved all **11** errors in the file but error displayed for **14** lines (one of the errors is resolved after 3 lines). Total number of errors displayed on console – 33 (Error displayed for each token until resolved)

8. 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

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

10. Execution

[A]. status (describe in maximum 2 lines): Working correctly on all test cases

[B]. Execution time taken for

- t1.txt (in ticks): *CANNOT BE PARSED* and (in seconds): *CANNOT BE PARSED*
- t2.txt (in ticks): 35 and (in seconds): 0.000035

- t3.txt (in ticks): 346 and (in seconds): 0.000346
- t4.txt (in ticks): 818 and (in seconds): 0.000818
- t5.txt (in ticks): 1013 and (in seconds): 0.001013
- t6.txt (in ticks): 1281 and (in seconds): 0.001281

[C]. Gives segmentation fault with any of the test cases (1-6) uploaded on the course page. If yes, specify the testcase file name: -----

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

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

13. Declaration: We, Ujjwal Gandhi, Mayank Jain, Atmadeep Banerjee and Aditya Mithal declare that we have put our genuine efforts in 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 a 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.

ID: 2017A7PS0143P Name: Ujjwal Gandhi

ID:2017A7PS0101P Name: Atmadeep Banerjee

ID:2017A7PS0179P Name: Mayank Jain

ID:2017A7PS0157P Name: Aditya Mithal

Date: 24-02-2020
