**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**Compiler Construction (CS F363)**

**II Semester 2019-20**

**Compiler Project (Stage-1 Submission)**

**Coding Details**

**Group No. 20**

**(February 24, 2020)**

1. IDs and Names of team members

ID: 2016B3A70398P Name: Ayush Vachaspati

ID: 2016B1A70938P Name: Indraneel Ghosh

ID: 2016B1A70929P Name: G Adityan

1. Mention the names of the Submitted files :

1 makefile 2 nary\_tree.c 3 stack.h 4 lexerDef.h

5 parser.c 6 nary\_tree.h 7 color.c 8 set.c

9 hashtable.h 10 driver.c 11 stack.c 12 color.h

13 hashtable.c 14\_productions.txt 15 parser.h 16 lexer.c

17 set.h 18parserDef.h 19 hash.h 20 lexer.h

21 hash.c 22 t1.txt 23 t2.txt 24 t3.txt

25 t4.txt 26 t5.txt 27 t6.txt 28 t7.txt

29 t8.txt 30 coding details stage1.docx

1. Total number of submitted files: 30

(All files should be in **ONE folder** named exactly as Group\_#, # is your group number)

1. 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]

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

yes

1. **Lexer Details:**
   1. Technique used for pattern matching: Twin Buffer for efficient I/O and DFA to match lexemes
   2. DFA implementation (State transition using switch case, graph, transition table, any other (specify): switch case
   3. Keyword Handling Technique: Hashtable to store all keywords
   4. Hash function description, if used for keyword handling: DJB Hash Function: An algorithm produced by Professor Daniel J. Bernstein
   5. Have you used twin buffer? (yes/ no) Yes
   6. Lexical error handling and reporting (yes/No): Yes
   7. Describe the lexical errors handled by you

1) ID Length Error(If ID length>20 it is an error)

2) Invalid Number error.

3) Invalid Operations error.

4) Unrecognized symbol in the source code error.

All errors are reported sequentially with Line numbers along with the lexeme that caused the error in order to allow the user to easily find and fix lexical errors.

* 1. Data Structure Description for tokenInfo (in maximum two lines):

Token is stored in struct containing enum for token, int for line number and string for the lexeme which can be converted to int and float on demand when needed later.

* 1. Interface with parser Implemented

Nary\_tree parse\_input(type start\_symbol, char\* sourcefile);

Returns the Nary Tree formed by the input source code. Also reports all errors sequentially on the console along with Line Number.

1. **Parser Details:** 
   1. **High Level Data Structure Description (in maximum three lines each, avoid giving C definitions used):**
      1. grammar : Struct of production and number of productions.

Each production has a rule stored as an array of enum, size of the rule and the left side non terminal of the rule.

* + 1. parse table : 2D array of productions. Indexed by Nonterminal enum and Terminal enum.

Stores NULL productions in entries where rules are not present(error entries).

* + 1. parse tree: (Describe the node structure also)

Nary tree containing Nonterminal/Terminal symbol, Token info ,Children nodes and Number of Children.

struct treenode{

token\* lexeme;

int tok;

struct treenode\*\* children;

int n;

};

struct Nary\_tree{

treenode\* root;

} ;

* + 1. Parsing Stack node structure : Uses treenode from the nary tree itself to construct the syntax tree while parsing.
    2. Any other (specify and describe): Set Implemented as Hahtable

Set has a function setUnionEPS which taked the union of two sets excluding EPS.

* 1. **Parse tree** 
     1. Constructed (yes/no): Yes
     2. Printing as per the given format (yes/no): Yes
     3. Describe the order you have adopted for printing the parse tree nodes (in maximum two lines)

Inorder traversal of the tree as described in notice with the order

leftmost child → parent → all other children.

* 1. **Grammar and Computation of First and Follow Sets** 
     1. Data structure for original grammar rules struct containing array of enum type and the size of rule (prodn)
     2. FIRST and FOLLOW sets computation automated (yes /no): Yes
     3. Data structure for representing sets Hashtable
     4. Time complexity of computing FIRST sets O(n2\_) where n is the number of rules
     5. Name the functions (if automated) for computation of First and Follow sets void makeFirstAndFollow(productions grammar, type start\_symbol)

void getFirstSet(productions grammar)

void getFollowSet(productions grammar)

* + 1. If computed First and Follow sets manually and represented in file/function (name that) Automated
  1. **Error Handling** 
     1. Attempted (yes/ no): Yes
     2. Printing errors (All errors/ one at a time) : One at a time
     3. Describe the types of errors handled

Parser handles Non-terminal and terminal pair rule not exists I.e error entry in the parsing table. It also handles terminal-terminal mismatch.

* + 1. Synchronizing tokens for error recovery (describe): Follow Set and First set of Non terminals used for synchronization as described in panic mode recovery in slides.
    2. Total number of errors detected in the given testcase t6(with\_syntax\_errors).txt: 10/11 errors reported and handles correctly.

1. **Compilation Details:**
   1. Makefile works (yes/no): yes
   2. Code Compiles (yes/ no): yes
   3. Mention the .c files that do not compile: N/A
   4. Any specific function that does not compile: N/A
   5. Ensured the compatibility of your code with the

specified gcc version(yes/no): yes

1. **Driver Details**: Does it take care of the options specified earlier(yes/no): yes
2. **Execution** 
   1. status (describe in maximum 2 lines): The code runs correctly for all test cases. It also handles and reports all errors sequentially along with line numbers.
   2. Execution time taken for
      * t1.txt (in ticks) 2827 and (in seconds) 0.002827
      * t2.txt (in ticks) 3500 and (in seconds) 0.003500
      * t3.txt (in ticks) 4612 and (in seconds) 0.004612
      * t4.txt (in ticks) 4756 and (in seconds) 0.004756
      * t5.txt (in ticks) 7115 and (in seconds) 0.007115
      * t6.txt (in ticks) 6738 and (in seconds) 0.006738
   3. Gives segmentation fault with any of the test cases (1-6) uploaded on the course page. If yes, specify the testcase file name: No Segmentation Faults
3. Specify the language features your lexer or parser is not able to handle (in maximum one line) None
4. Are you availing the lifeline (Yes/No): No
5. Declaration: We, Ayush Vachaspati, Indraneel Ghosh and G Adityan 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. [Write your ID and names below]

ID: 2016B3A70398PName: Ayush Vachaspati

ID: 2016B1A70938P Name: Indraneel Ghosh

ID: 2016B1A70929P Name: G Adityan

Date: 24/02/2020