Programming languages and compiler

Phase2: Parser Generator

Mahmoud Mohamed	**19016578**
Gamal Abdel Hameed Nasef Nowesar	**19015550**
Moamen Mahmoud Gharib	**19016208**
Salah Eldin Eltenihy	19015854

1- A description of the used data structures

Data Structure	**Usage**
`vector<pair<string,string>>Grammar_rules;`	Keep track of Grammar Rules LHS = RHS
`map<string,vector<string>> rules_map;`	Grammar rules in other form A -> B | C | D A -> {B, C, D}
`map<string,set<string>> first_sets;`	Keep track of first sets LHS -> first(LHS)
`map<string,map<string,string>> table;`	Parse table LHS -> Map between T -> (Production Rule)
`map<string,set<string>> follow_sets;`	Keep track of follow sets LHS -> follow(LHS)

2- Explanation of all algorithms and techniques used

Algorithm	**Steps**

Eliminate Left Recursion	For LHS:
	 For prod in RHS:
	  Alpha = “”
	  Beta = “”
	  If LHS exists in first of prod:
	   Alpha += prod
	  Else
	   Beta += prod
	 If there is Left Recursion:
	   Modify LHS in Grammar_rules:
	   LHS -> Beta LHS_DASH
	   LHS_DASH -> Alpha LHS_DASH|Epsilon

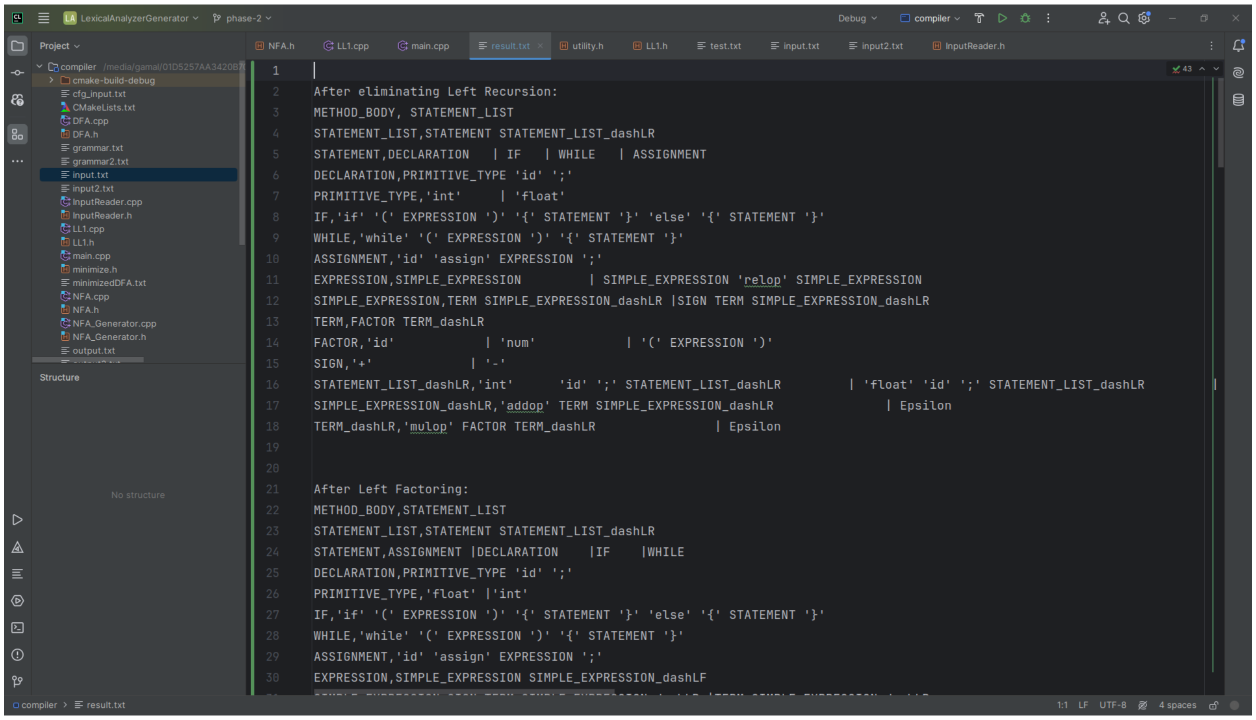
Left Factor	//Left factor RHS factors of same predecessor Curr_group = For LHS: For prod in RHS: map<string, vector<string>> groups If Curr_group substr of prod: Add prod to group Else: Wrap up old group [substr[A1, A2, …]] and add it to groups Curr_group = prod
Firset Set	For LHS: For prod in RHS: If prod is terminal: Add to first set Else:

getFollowSets	Make a map to store the rightmost non-terminals of each LHS. Then apply topological sort. Then get the follow set for each non-terminal ‘str’: loop overall the productions and for each production, loop over the production’s parts: If the ‘str’ is followed directly by a terminal, add this terminal to the follow set of ‘str’. If the ‘str’ is followed directly by a non-terminal, then add the first set of this non-terminal to the follow set of ‘str’. Then remove ‘Epsilon’ from the follow set if it exists. If ‘str’ is a rightmost nonterminal of the current LHS, then add the follow set of the LHS to the follow set of ‘str’.
createTable	Iterate through each non-terminal in the grammar. Iterate through each production of the current non-terminal. Split the production into individual parts based on spaces. Check if the first symbol of the production is a terminal: If the entry in the parsing table is already occupied, print an error. Else, Set the entry in the parsing table for the current non-terminal and

	terminal symbol. Else If the first symbol of the production is a non-terminal: Iterate through the First set of the first non-terminal in the production: If the entry in the parsing table is already occupied, print an error. Else, Set the entry in the parsing table for the current non-terminal and terminal symbol. Iterate through the Follow set of the current non-terminal: If the entry in the parsing table is not present or is set to "Sync": Set the entry in the parsing table to "Epsilon" or "Sync" based on the presence of "Epsilon" in the First set. If "Epsilon" is in the First set but the entry in the parsing table is not "Epsilon," print an error.
LL1_parse	Initially, flag is true. Loop until flag is false: Get the top of the stack. If top is the end-of-input marker "$": If input is "$", then the accepted, else rejected. If top is a terminal symbol: If top is a single-character terminal

	(e.g., 'a'), then match the input with the terminal symbol, and pop it from the stack. Else, Error (the input is missing). Else, if the top of the stack is a non-terminal: If the entry in the parsing table for the current non-terminal and terminal symbol of input is empty, then error. Else, if the production is "Sync", then pop from the stack. If the production is "Epsilon", then pop from the stack. Else, Expand the stack with the production parts. Pop from the stack and push the production parts to the stack in reverse order.

3- Screenshots:



```

After eliminating Left Recursion:
METHOD_BODY, STATEMENT_LIST
STATEMENT_LIST,STATEMENT STATEMENT_LIST_dashLR
STATEMENT,DECLARATION   | IF   | WHILE   | ASSIGNMENT
DECLARATION,PRIMITIVE_TYPE 'id' ';'
PRIMITIVE_TYPE,'int'     | 'float'
IF,'if' '(' EXPRESSION ')' '{' STATEMENT '}' 'else' '{' STATEMENT '}'
WHILE,'while' '(' EXPRESSION ')' '{' STATEMENT '}'
ASSIGNMENT,'id' 'assign' EXPRESSION ';'
EXPRESSION,SIMPLE_EXPRESSION         | SIMPLE_EXPRESSION 'relop' SIMPLE_EXPRESSION
SIMPLE_EXPRESSION,TERM SIMPLE_EXPRESSION_dashLR |SIGN TERM SIMPLE_EXPRESSION_dashLR
TERM,FACTOR TERM_dashLR
FACTOR,'id'            | 'num'            | '(' EXPRESSION ')'
SIGN,'+'             | '-'
STATEMENT_LIST_dashLR,'int'      'id' ';' STATEMENT_LIST_dashLR         | 'float' 'id' ';' STATEMENT_LIST_dashLR
SIMPLE_EXPRESSION_dashLR,'addop' TERM SIMPLE_EXPRESSION_dashLR               | Epsilon
TERM_dashLR,'mulop' FACTOR TERM_dashLR                | Epsilon


After Left Factoring:
METHOD_BODY,STATEMENT_LIST
STATEMENT_LIST,STATEMENT STATEMENT_LIST_dashLR
STATEMENT,ASSIGNMENT |DECLARATION    |IF    |WHILE
DECLARATION,PRIMITIVE_TYPE 'id' ';'
PRIMITIVE_TYPE,'float' |'int'
IF,'if' '(' EXPRESSION ')' '{' STATEMENT '}' 'else' '{' STATEMENT '}'
WHILE,'while' '(' EXPRESSION ')' '{' STATEMENT '}'
ASSIGNMENT,'id' 'assign' EXPRESSION ';'
EXPRESSION,SIMPLE_EXPRESSION SIMPLE_EXPRESSION_dashLF
```