2019-2020 SPRING SEMESTER
CS315 PROGRAMMING LANGUAGES
LANGUAGE NAME: TRIP

PROJECT - 2 REPORT (GROUP 26)

TEAM MEMBERS:

MEMBER NAME	ID	SECTION
BERK YILDIZ	21502040	1
BURAK BAYAR	21401971	2
OSMAN CAN YILDIZ	21302616	1

BNF OF TRIP LANGUAGE

<program> -> <stmts>

<stmts> -> <stmt> | <stmt>; <stmts>

<stmt> -> <if_stmt> | <non_if_stmt>

<if_stmt> -> <matched> | <unmatched>

<matched> -> if (<logic_expr>) <matched> else <unmatched>

| <non_if_stmt>

<unmatched> -> if (<logic_expr>) <if_stmt>

| if (<logic_expr>) <matched> else <unmatched>

<non_if_stmt> -> def_function | <call_stmt> | <input_stmt> | <assignment_stmt>

| <loop_stmt> | <delete_stmt> | <output_stmt> | <return_stmt> | {<stmts>}

<logic_expr> -> <simple_logic_expr> | <logic_expr> <logic_op> <logic_expr>

| (<logic_expr>) | !(<logic_expr>)

<def_function> -> <function_name><LP><RP><LB><stmts><RB>

| <function_name><LP><RP><LB><stmts><<return_stmt><RB>

| <function_name><LP><parameter_dec><RP><LB><stmts><RB>

| <function_name><LP><parameter_dec><RP><LB><stmts><return_stmt><RB>

<parameter_dec>-> <parameter_dec>

| <parameter_dec><types><variable_identifier>

|<types><variable_identifier>

<function_name> -> <letter>| <function_name><letter>

<call_stmt> -> <function_identifier> (<element_list>) | <function_identifier>( )

<input_stmt> -> insert <variable_identifier>

<assignment_stmt> -> <variable_identifier> \= <variable_identifier>

| <variable_identifier> \= <number>

<loop_stmt> -> <for_stmt> | <while_stmt>

<for_stmt> -> for(<variable_identifier>: in <set>) {<stmts>}

<while_stmt> -> while(<logic_expr>) {<stmts>}

<delete_stmt> -> delete <set>

<output_stmt> -> get <set>

<return_stmt> -> return <set>;

<set> -> <types> | {<element_list>}

<element_list> -> <types> | <element_list>, <types>

<types> -> <variable_identifier> | <function_identifier> | <string> | <number>

<string> -> ".+"

<variable_identifier> -> <lowcase_letter> | <lowcase_letter> <alphnumeric_string>

<function_identifier> -> <upcase_letter> | <upcase_letter> <alphanumeric_string>

<arithmetic_op> -> <term> | <arithmetic_op>+<term> | <arithmetic_op>−<term>

<term> -> <factor> | <term>∗<factor> | <term>/<factor>

<factor> -> <primary> | (<arithmetic_op>)

<primary> -> digit | number

<number> -> <signed_integer> . <unsigned_integer>

<lowcase_letter> -> a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<upcase_letter> -> A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<letter> -> <upcase_letter> | <lowcase_letter>

<sign> -> +|-

<signed_integer> -> <digit> | <sign> <signed_integer> | <digit> <signed_integer>

<unsigned_integer> -> <digit> | <digit> <unsigned_integer>

<digit> -> 0|1|2|3|4|5|6|7|8|9

<alphanumeric> -> <letter> | <digit>

<alphanumeric_string> -> <alphanumeric> | <alphanumeric> <alphnumeric_string>

<logic_op> -> && | \|\|

<set_relations> -> > | < | != | ==

<set_operations> -> <difference> | <intersection> | <union> | <cartesian>

<difference> -> ^

<intersection> -> ?

<union> -> !

<cartesian> -> %

<simple_logic_expr> -> <set> <set_relations> <set>

<comment> -> #string

<space> -> [\t]

<dolar> -> $

<underscore> -> _

<slash> -> \

BNF DESCRIPTION OF TRIP LANGUAGE

<program>: This non-terminal indicates the beginning.

<stmts>: This non-terminal indicates the statements which are used in the program. There can be one statement or many statements.

<stmt>: This non-terminal includes all kinds of statements that can be written in our language. The non-terminal divides into 2 main statements which are if statement and non-if statement.

<if_stmt>: Indicates the if statements that can be written in our language. It works in the same way with C. Also denotes the rule for writing if statement.

<matched>: Indicates the if statements with else parts.

<unmatched>: Indicates the if statements without else parts.

<logic_expr>: This non-terminal contains all logical expressions that can be written in our language.

<def_function>: This non-terminal indicates the rule for function definition.

<parameter_dec>: Denotes the rule for parameters given to the function

<function_name>: Denotes the rule for naming a function.

<non_if_stmt>: This non-terminal indicates the statements which are not an “if” statement. It includes loop statements, call statement, input statement, assignment statement, delete statement, output statement and return statement.

<call_stmt>: This non-terminal indicates the rule for calling a function.

<input_stmt>: This non-terminal indicates the rule getting a user input.

<assignment_stmt>: This non-terminal indicates the assignment statements like a =3 or b = [1,2,3].

<loop_stmt>: This non-terminal indicates the loop statements that can be written in the language. These loop statements are for and while loops which all developers are familiar with from the common programming languages.

<for_stmt>: Denotes to rule for writing a for loop.

<while_stmt>: Denotes to rule for writing a while loop. It has similar rules with C group languages..

<delete_stmt>: Denotes the rule for deleting a set.

<output_stmt>: Denotes the rule for getting output on the console.

<return_stmt>: Denotes the rule for returning a statement.

<set>: This non-terminal denotes the rule for defining a set. A set can include different types of variables.

<element_list>: This non-terminal is the list of the types of a set.

<types>: These are the types that exist in the language. These types are variable identifier, function identifier, string and number.

<string> : This non-terminal denotes the rule for a string.

<variable_identifier>: This non-terminal indicates the rule for describing the variable names.

<function_identifier>: This non-terminal indicates the rule for describing the function names.

<arithmetic_op>: Denotes the arithmetic operations that can be used in our language which are summation, subtraction, multiplication and division.

<term> ::=: Denotes the elements of arithmetic operation.

<lowcase_letter>: Indicates lower case letter.

<upcase_letter>: Indicates upper case letter.

<letter>: Indicates a letter from the alphabet.

<sign>: Indicates the sign of a number.

<signed_integer>: Indicates number which has sign.

<unsigned_integer>: Indicates number without a defined sign.

<digit>: Simply indicates a digit.

<alphanumeric>: Indicates alphanumeric character.

<alphanumeric_string>: Indicates sequence of alphanumeric characters.

<logic_op>: Indicates the logic operations ‘and’ & ‘or’.

<set_relations>: Indicates the set relations which are subset, equal and not equal.

<set_operations>: Indicates the set operations which are intersection, union, difference and cartesian.

<simple_logic_expr>: Indicates rule for having a simple logic expression.

<comment>: This non-terminal indicates the rule for adding a comment in the code. Comments are not compiled by the compiler and they do not affect the flow of the code.

<space> -> [\t]

<dolar> -> $

<underscore> -> _

<slash> -> \

CONFLICTS

In our language, our program is returning tokens well during the execution of the lex file but after executing the parser, we have some conflicts that cause syntax error at the line 1 which we couldn’t solve or understand why. On the other hand, we have handled some shift/reduce conflicts.