CS 315 Spring 2020

PROJECT 2

Group 24

Mehmet Ali Altunsoy - 21702531 - Section 1
Hasan Doğan - 21402109 - Section 1

Name of the language: splang

Part A – Revised and Augmented Language Design

Complete BNF of splang:

```
<program> ::=  exec< <stmts> >end
<stmts> ::=  < stmt > | < stmts > < stmt >
<stms> ::=  <matched> | <unmatched>
<matched> ::=  if ( <bool_expr> ) { <matched> } else {<matched> }
                          | <while_stmt>
                          | <assign_stmt>
                          | <return_stmt>
                          | <do_while_stmt>
                          | <dec_var_stmt>
                          | <set_stmt>
                          | <func_call_stmt>
                          | <func_imp>
<unmatched> ::=  if (<bool_expr>) {<matched>}
             | if (<bool_expr>) {<matched>} else {<unmatched>}
<func_call_stmt> ::=  identifier( <func_params> ){ <stmts> };
                        |get_size (<set_variable>);
                        |is_empty (<set_variable>);
                        |readFile(<string>);
<func_call_imp> ::=  identifier( <func_params> ){ <stmts> }
<assign_stmt> ::=  <variable> <assign_op> <expr> ;
             | <variable> <assign_op> <func_call_stmt>;
             | <truth_var> <assign_op> <true_or_false>  ;
             |<set_variable> <assign_op> <set_operations>;
             |<set_variable><assign_op> <set>;
<return_stmt> ::=  return <bool_expr> ;
<while_stmt> ::=  while ( <bool_expr> ) { <stmts> } ;
<do_while_stmt> ::=  do{ <stmts> } while ( <bool_expr> );
<dec_var_stmt> ::=  var<space><variable>;
<set_stmt>::= <create_set> | <delete_set> | <set_operation>;
<create_set>::=  create<space><set_variable>;
<delete_set>::=  delete<space><set_variable>;
<print_set>::=  print<space><set_variable>;
<add_to_set> ::=  <set_variable> <add_to_set_op> <variable>
<remove_from_set> ::=  <set_variable> <remove_from_sets_op> <variable>
<set_addition> ::=  <set_variable> <set_addition_op> <set_variable>
             |  <set_addition> <set_addition_op> <set_variable>
<set_subtraction> ::=  <set_variable> <set_subtraction_op> <set_variable>
<set_difference> ::=  <set_variable> <set_difference_op> <set_variable>
<set_union> ::=  <set_variable> <set_union_op> <set_variable>
             | <set_union><set_union_op> <set_variable>
<set_intersection>::=  <set_variable> <set_intersection_op> <set_variable>
<sub_set> ::=  <set_variable> <subset_op> <set_variable>
<super_set> ::=  <set_variable> <superset_op> <set_variable>
<set_relations> ::=  <sub_set> | <super_set>
<set_operations> ::=  <add_to_set>;
             | <remove_from_set>
```

```
            | <set_addition>
            | <set_subtraction>
            | <set_difference>
            | <set_union>
            | <set_intersection>
<set_elements> ::=  <set_element> |  <set_elements>, <set_element>
<set_element> ::=  <no_space_string> | <set> | <set_variable>
<empty_set> ::=  {}
<set> ::=  <empty_set> | {<set_elements>}
<comment> ::=  ## <string> | <comment> <string> ##
<variable> ::=  &<no_space_string>
<set_variable> ::=  $<no_space_string>
<func_params> ::= <func_param> | <func_params>, <func_param>
<func_param> ::=  var <whole_variables>
<truth_var> ::=  £<no_space_string>
<whole_variables> ::=  <variable> | <set_variable> | <truth_var>
<letter> ::=  <letter_lower> | <letter_upper>
<letter_lower> ::=  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_upper> ::=  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
<digit> ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<integer> ::=  <digit> | <integer> <digit>
<float> ::=  <integer> <dot> <integer>
<real_num> ::=  <integer> | <float>
<true_or_false> ::=  true | false
<constant> ::=  cons <space> <variable> | cons <space> <truth_var>
<dot> ::= .
<comma> ::= ,
<string> ::=  <letter> | <digit>
            | < string><digit>
            | <string><letter>
            | <spaces><string>
<no_space_string> ::=  <letter> | <digit>
            | < string><digit>
            | <string><letter>
<spaces> ::=  <space>| <tab> | <new_line>
<space> ::=
<tab> ::=
<new_line> ::=
<expr> ::=  <expr> <ar_op> <real_num>
            | <real_num>
            | <expr> <ar_op> <variable>
            | <variable>
            |  <true_or_false>
            | <set_variable>
            | <set_operations>
            | <not_op> <true_or_false>
            | ( <bool_expr> )
            | ( <expr> <bool_op> <expr> )
            | <truth_var>
            | <not_op> <truth_var>
```

```
<bool_expr> ::=  <expr> <bool_op> <expr>
            | <true_or_false>
<bool_op> ::=  <eq_op>
            | <great_op>
            | <less_op>
            | <great_eq_op>
            | <less_eq_op>
            | <and_op>
            | <or_op>
            |<sub_set_op>
            |<super_set_op>
<subset_op> ::=  <<
<superset_op> ::=  >>
<add_to_set_op> ::=  [+]
<remove_from_Set_op> ::=  [-]
<set_addition_op> ::=  [++]
<set_subtraction_op> ::=  [--]
<set_difference_op> ::=  [//]
<set_union_op> ::=  [U]
<set_intersection_op>::=  [%%]
<ar_op> ::=  + | - | * | /
<eq_op> ::=  ==
<assign_op> ::=  =
<great_op> ::=  >
<less_op> ::=  <
<great_eq_op> ::=  >=
<less_eq_op> ::=  <=
<and_op> ::= aNd
<or_op> ::=  oR
<not_op> ::=  ~
```

Explanation of Nonterminals of splang:

- **<program>** Program consists of functions which are called <func_defs> in BNF.
- **<stmts**> This nonterminal represents multiple usage of statements which are called <stmt> and defined by left recursion.
- **<stmt>**This nonterminal represents defining a statement. statement types are given.
- **<matched>** This nonterminal represents matched if statements which have an equal number of if and else and other statements.
- **<unmatched>** This nonterminal represents unmatched if statements which has unmatched number of if and else.
- **<assign_stmt>** This nonterminal represents assigning values to variables or truth variables.
- **<return_stmt>** This nonterminal represents a function's return variable.
- **<func_call_stmt>** This nonterminal represents function calls in the program.
- **<func_imp>** This nonterminal implements functions.
- **<while_stmt>** This nonterminal represents the structure of while statements which consists of "while", a boolean expression and statements.
- **<do_while_stmt>** This nonterminal represents the structure of do-while statements which consists of "do", statements, "while" and boolean expressions.
- **<dec_var_stmt>** This nonterminal represents variable decleration statements.
- **<set_stmt>** This nonterminal represents set creation and deletion statements.
- **<create_set**> This nonterminal creates a new set.
- **<delete_set>** This nonterminal deletes a set.
- **<print_set>** This nonterminal prints the set.
- <**add_to_set>** This nonterminal adds a variable to a set.
- **<remove_from_set>** This nonterminal removes a variable from a set.
- **<set_addition>** This nonterminal makes set addition.
- **<set_subtraction>** This nonterminal makes set subtraction.
- **<set_difference>** This nonterminal takes the difference of two sets.
- **<set_union>** This nonterminal unites the two or more sets in a new set.
- **<set_intersection>** This nonterminal finds the intersection of two or more sets in a new set.
- **<sub_set>** This nonterminal checks whether a set is a subset of another set.
- **<super_set>** This nonterminal checks whether a set is a superset of another set.
- **<set_relations>** This nonterminal represents the subset and superset relations.
- **<set_operations>** This nonterminal represents the set operations.
- **<set_elements>** This nonterminal represents the combination of set elements which are <set_element>
- **<set_element>** This nonterminal represents the set elements.
- **<empty_set>** This nonterminal represents an empty set
- **<set>** This nonterminal represents a set.
- **<comment>** This nonterminal represents the comments that a developer might add to the code. Comments do not affect the execution of the program. They increase the readability.
- **<variable>** This nonterminal represents variables except set and boolean variables.
- **<set_variable>** This nonterminal represents set variables.

- **<func_params> ::=** This nonterminal represents collection function parameters.
- **<func_param> ::=** This nonterminal represents function parameters.
- **<truth_var>** This nonterminal represents boolean variables.
- **<whole_variables>** This nonterminal represents whole variables.
- **<letter>** This nonterminal represents the lower or upper case characters in the English alphabet.
- **<letter_lower>** This nonterminal represents the lower case letters in the English alphabet.
- **<letter_upper>** This nonterminal represents the uppercase letters in the English alphabet.
- **<digit>** This nonterminal represents the digits in base 10.
- **<integer>** This nonterminal represents integers.
- **<float>** This nonterminal represents floating point numbers.
- **<real_num>** This nonterminal represents floating point numbers or integers.
- **<true_or_false>** This nonterminal represents boolean values: true or false
- **<constant>** This nonterminal represents variables and boolean variables which are constant values so that these values cannot be changed.
- **<dot>** This nonterminal represents "." symbol.
- **<comma>** This nonterminal represents "," symbol.
- **<string>** This nonterminal represents possible alphanumeric strings.
- **<no_space_string>** This nonterminal represents string without space character.
- **<spaces>** This nonterminal represents space, tab and new line characters.
- **<space>** This nonterminal represents " " character.
- **<tab>** This nonterminal represents "\t" character
- **<expr>** This nonterminal represents expressions.
- **<bool_expr>** This nonterminal represents boolean expressions.
- **<bool_op>** This nonterminal represents boolean operators.
- **<subset_op>** This nonterminal represents a subset (<<) operator.
- **<superset_op>** This nonterminal represents a superset (>>) operator.
- **<add_to_set_op>** This nonterminal represents variable addition to a set ([+]) op.
- **<remove_from_set_op>** This nonterminal represents variable removal from a set ([-]) operator.
- **<set_addition_op>** This nonterminal represents a set addition ([++]) operator.
- **<set_subtraction_op>** This nonterminal represents a set subtraction ([--]) operator.
- **<set_difference_op>** This nonterminal represents a set difference ([//]) operator.
- **<set_union_op>** This nonterminal represents a set union (U) operator.
- **<set_intersection_op>** This nonterminal represents a set intersection (%%) operator.
- **<ar_op>** This nonterminal represents arithmetic operators.
- **<eq_op>** This nonterminal represents equality operators.
- **<assign_op>** This nonterminal represents assignment (=) operator.
- **<great_op>** This nonterminal represents a greater (>) operator.
- **<less_op>** This nonterminal represents a less than (<) operator.
- **<great_eq_op>** This nonterminal represents a greater than or equal to (>=) operator.
- **<less_eq_op>** This nonterminal represents a less than or equal to (<=) operator.
- **<and_op>** This nonterminal represents a and (aNd) operator.
- **<or_op>** This nonterminal represents a or(oR) operator.
- **<not_op>** This nonterminal represents a not (~) operator.

Description of Nontrivial Tokens of splang:

1) Comments: These are used for explanation of a specific part of the code. They are made out of ## <string> ##. They are used in only a line. They increase the readability by allowing developers to put notes in the code.

2) Identifiers:

- Function Identifiers: These identifiers are used to distinguish the functions.
- Variable Identifiers: These identifiers are used to distinguish the regular variables by using "&" symbol reduces writability.
- Truth Variable Identifiers: These identifiers are used to distinguish the truth variables by using "£" symbol reduces writability.
- Set Variable Identifiers: These identifiers are used to distinguish the set variables by using "$" symbol reduces writability.

3) Literals:

- Float literals: Float literals are defined in the bnf in the form of '3.21'.
- Integer literals: These literals are defined in the bnf in the form of '21'.
- Real literals: These literals are defined in the bnf in the form of '21' or '3.21'. Real literals are either integer literals or float literals.

4) Reserved keywords:

- return: This reserved word is used only in functions. It will return the written statement and returns it after the function is used.
- while: This reserved word is used for a typical while loop.
- do: This reserved word is used for a typical do-while loop.
- if and else: These reserved words are used for a typical "if, else" situation.
- true and false: These reserved words are used for boolean expressions.

5) Operators:

- = : This operator is used for assignments.
- + : This operator represents plus (+).
- - : This operator represents minus (-).
- *: This operator represents multiplication (*).
- /: This operator represents division (/).
- == : This operator represents the equality operator.
- < : This operator represents the less than operator.
- > : This operator represents the greater than operator.
- <= : This operator represents the less than or equal to operator.
- >= : This operator represents the greater than or equal to operator.
- aNd : This operator represents the logic "and" operator.
- oR : This operator represents the logic "or" operator.
- ~ : This operator represents the "negation" operator.
- << : This operator represents the subset operator.

- >> : This operator represents the superset operator.
- [+] : This operator represents the add to set operator.
- [-] : This operator represents the removal from the set operator.
- [++] : This operator represents the set addition operator.
- [--] : This operator represents the set subtraction operator.
- [//] : This operator represents the set difference operator.
- [U] : This operator represents the set union operator.
- [%%] : This operator represents the set intersection operator.

6) Separators: The following separators are used to increase the readability by

separating a piece of code from others so that the borders easily distinguishable.

- ## ##: These separators are used by the compiler to distinguish executable code from developer comments.
- ( ): These separators are used by the compiler to identify boolean expressions and arguments.
- { }: These separators are used by the compiler to identify when an if/else statement starts and ends and is used for sets brackets.
- , : This separator is used in a function's parameter part to distinguish the different parameters and is used for distinguish set elements .
- ; : This separator indicates that the given statement is over.

Conflicts:

No conflicts.