Sharpie

Ali Babayev - 21600475 (section 1)

Nurlan Farzaliyev - 21503756 (section 1)

Mahammad Shirinov - 21603176 (section 2)

Sharpie

BNF

```
<set> -> [<set_elements>]
<set_elements> -> <set_element>
                | <set_elements>, <set_element>
<set_element> -> <identifier> | <data_type>
<identifier> -> <letter>
                | <identifier><letter>
                | <identifier><digit>
<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
            A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|K|R|S|T|U|V|W|X|Y|Z
<digit> -> 0|<positive_digit>
<positive_digit> -> 1|2|3|4|5|6|7|8|9
<num> -> <digit>|<num><digit>
<data_type> -> <primitive_data_type> | <set>
<primitive_data_type> -> <int>
                | <boolean>
<int> -> <sign>?<positive_digit>|<int><digit>
<boolean> -> true|false
<sign> -> -|+
<set_composition> -> <set><set_op><set>
<set_op> -> & | ^ | ~
<set_inclusion> -> <set><inclusion_symbol><set>
<inclusion_symbol> -> << | >>
<assignment_operator> -> =
<while_loop> -> while(<boolean_expression>)<code_block>
<code_block> -> {<statements>}
<boolean_expression> ->
<primitive_data_type><logical_operand><primitive_data_type>
                | <set_inclusion>
<logical_operand> -> <equals> | <nequals>
<statements> -> <statement>; | <statements><statement>
<statement> -> <identifier> <assignment_operator> <expression>
                | <def_var>
                | <oxu_statement>
                | <yaz_statement>
                | <function_def>
                | <function_call>
                | <while_loop>
                | <if_statement>
```

```
<type> -> int | boolean | set

<expression> -> <set>
               | <data_type>
               | <expression><assignment_operator><data_type>
               | <set_composition>
               | <set_inclusion>

<oxu_statement> -> oxu(<identifier>)
<yaz_statement> -> yaz(<identifier>)
<if_statement> -> if(<boolean_expression>)<code_block>
               |if(<boolean_expression>)<code_block>else<code_block>

<function_def> -> <return_type> <identifier> (<def_var_list>)
<code_block>
<return_type> -> <type> | void
<def_var_list> -> void
               | <def_var>
               | <def_var_list>, <def_var>
<def_var> -> <type> <identifier>
<function_call> -> <identifier>(<parameter_list>)
<parameter_list> -> <identifier>
               | <parameter_list>, <identifier>

<keyword> -> void | true | false | int | boolean | set | oxu | yaz
<comment> -> //<identifier> | <comment> <identifier>
```

The complete BNF description of the language

The Elements of our language:

<**set**>: a set is a collection of ‘things’, which can include integers, doubles, strings, booleans or other sets. In syntax, it is a comma separated list of identifiers that refer to these ‘data types’, enclosed in square brackets.

<**set_elements**>: collection of elements of a set

<**set_element**>: an element that a set contains.

<**identifier**>: name of variables, functions…

<**letter**>: letter characters that can be used in our language

<**digit**>: positive integers and zero

<**positive_digit**>: positive digits

<**num**>: numbers that contains several integers

<**data_type**>: data type can be “set” or primitive data types which are integer and boolean type

<**primitive_data_type**>: integer or Boolean

<**int**>: definition of integer

<**sign**>:integer numbers can be positive or negative

<**assignment_operator**>: =

<**while_loop**>: statement that process as a loop until its Boolean expression becomes false

<function_def>: definition of the functions while defining first time

<**code_block**>: contains data which are between brackets of function

<return_type>: return type of the functions

<**boolean_expression**>: expression that checks the case, such as setinclusion . It references to "condstmt" in yacc file. Boolean can be set to condstmt.

<**set_inclusion**>: the expression that process whether first set is super or subset of second set. this captures set inclusion statements, returning a boolean value of true or false. Set inclusion is denoted by characters '<<' and '>>', where X<<Y (same as Y>>X) means "X is a subset of Y". For example, if set A is a subset of B, the value of A<<B will be true.

<**set_composition**>: this is an operation between two or more sets, returning a set of intersection or union, or difference of the argument sets.

<**logical_operand**>: 2 operands that checks whether two value, variable are equal or not

<**statements**>: instruction to assign to identifier or just to implement as an action such as print or read

<identifier>: a sequence of characters that is used to identify variables, data types, or functions. They are basically used as names.

<**type**>: is one of the keywords int, boolean, set, that are used to define the types of data.

<**data_type**>: this is used to represent data; they can be primitive data types (integer, double and boolean values) or strings and sets. Unlike <type>s, these refer to the actual values of data_types, and not to their labels or locations.

<**comment**>: single line of characters ignored by the compiler, starting with '//'

<**keyword**>: words which can not be used by user for names of smth.

yaz and oxu keywords: names of functions for output and input, respectively.

void: type of function when function doesn't return anything

true: state of Boolean exp. or var.

false: state of Boolean exp. or var.

set: data type

yaz: for print statement

oxu: for read statement

Explanation for yacc file of our language

we have defined several tokens: they indicate …

%token **NEWLINE** – indicates new line

%token **SET BOOL INTEGER SETOFSETS -**

%token **TYPESET TYPEINTEGER TYPEBOOL** – primitive type of variable: integer, Boolean, and set

%token **SEMIC** – indicates semicolon(end of the line)

%token **ISSETTO**: indicates assignment operator

%token **EQUALS NEQUALS**: indicates equal and not equal symbols(= and !=)

%token **IDENTIFIER** = indicates name of variables, functions…

%token **PRINT READINPUT** – indicates print and read input operations(respectively "yaz" and "oxu")

%token **LP RP** – indicates left and right parentheses respectively

%token **IF WHILE** – indicates if and while statements

%token **LCB RCB** – indicate left and right curly brackets respectively

%token **SUBSETOF SUPERSETOF** – indicates subset and superset symbols for sets

%token **SETUNION SETINTERSECTION SETDIFF** – indicates "&" , " ^ ", "~" symbols respectively

$token **NOT** – indicates not symbol, means reverse of the operation

$token **COMMENT** – indicates given comment

$token **MAIN** – indicates main program with what every program should begin

%%

programs: matches the correct structure of program written in our language which should begin with “main” and should be enclosed with curly brackets and as a result of correctness of the structure, message is printed to indicate that structure is valid.

```
main {
…
}
```

lines – matches line of the given code which can be line of comment , assignment operation, print and read statements

ifstmt – matches structure of if statement

```
if (condition) {
…
}
```

whilestmt - matches structure of while statement

```
while (condition) {
…
}
```

condstmt: matches a statement which checks condition. reverse of this statement is also conditional statement. others can be conditions which compares identifiers, integers, sets with themselves and each other

intcomparator: matches equal and not equal symbols

setcomparator: matches subset and superset symbols

setop: matches union, intersection, and difference symbols for sets

declaration: matches structure of declarations of variables, sets(expressions)

assignment: matches structure of assignment expression

printstmt: matches structure of print statement

readstmt: matches structure of read statement

functiondef: matches structure of functions while defining. 4 different structure for the type of function

type: matches types of functions or variables. Boolean, integer, and set

parameterlist: matches parameters of functions while defining the functions

functioncall: matches structure of function while calling. Functions are called using the ‘do’ keyword.

```
do printSet( aSet);
```

parameter: `a parameter of function while calling the function.` Boolean, integer, and set

parameters: matches parameters of functions while calling the functions