2021–2022 SPRING SEMESTER

CS315 PROGRAMMING LANGUAGES

PROJECT 1 - REPORT

TEAM 13 - SETOSHI

A Programming Language for Sets and its Lexical Analyzer

COURSE: CS315 **SECTION:** 1

DATE: 25th February 2022

INSTRUCTOR: H. Altay Güvenir

TEACHING ASSISTANT: Irmak Türköz

TEAM MEMBERS:

NAME: Abdullah Riaz **STUDENT ID:** 22001296

NAME: Mohammed Sohail **STUDENT ID:** 22001513

CONTENTS

1. INTRODUCTION

The language **Setoshi** is designed to work with finite sets. The language's specification is included in BNF. The description of each language construct and the non-trivial token has been documented. The language has also been evaluated in terms of readability, writability, and reliability. At this stage, a set cannot have subsets or other sets as its elements. Additionally, a lex description file and three sample programs written in Setoshi have been attached to the report.

2. BNF DESCRIPTION OF THE LANGUAGE

2.1. Program

```
<program>    → <main>
<main>       → <statements>
<statements> → <statement>
             | <statement> <statements>
<statement>  → <comment>
             | <loop>
             | <expr>
             | <conditional>
             | <function>
```

2.2. Types

```
<form>       → <int>
             | <bool>
             | <char>
             | <float>
             | <string>
<bool>       → true | false
<char>       → <lowercase> | <uppercase>
<number>     → <int> | <int> <number>
<point>      → .
<float>      → <number> <point> <number>
<string>     → <char> | <char> <string>
<int>        → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<lowercase>  → 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
<uppercase>  → 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
<sign>       → + | -
<L_CB>       → {
<R_CB>       → }
<L_PR>       → (
<R_PR>       → )
<comma>      → ,
```

2.3. Variable Identifiers

```
<var_new>           → <form> <identifier>
<identifier>        → <string>
                    | <string> <identifier>
                    | <string> <number>
                    | <string> <number> <identifier>
```

2.4. Assignment Operators

```
<assign>            → <identifier> = <expr>
<assign_new>        → <var_new> = <expr>
<operator>          → <division_op>
                    | <multiplication_op>
                    | <addition_op>
                    | <subtraction_op>
<division_op>       → /
<multiplication_op> → *
<addition_op>       → +
<subtraction_op>    → -
```

2.5. Expressions

```
<expr>              → <identifier> <operator> <identifier>
                    | <identifier> <comparator> <identifier>
<comparators>       → <expr_and> | <expr_or> | <expr_lt> | <expr_gt>
                    | <expr_lte> | <expr_gte> | <expr_eq>
                    | <expr_eq_strict> | <expr_neq>
                    | <expr_neq_strict>
<expr_and>          → &&
<expr_or>           → ||
<expr_lt>           → <
<expr_gt>           → >
<expr_lte>          → <=
<expr_gte>          → >=
<expr_eq>           → ==
<expr_eq_strict>    → ===
<expr_neq>          → !=
<expr_neq_strict>   → !==
<foreach_expr>      → <identifier> as <var_new>
```

2.6. Conditional Statements

```
<conditional>       → <if>
<if>                → <matched_if> | <unmatched_if>
<matched_if>        → if <expr> <L_CB> <statements> <R_CB> else
<L_CB> <matched_if> <R_CB>
<unmatched_if>      → if <expr> <L_CB> <matched-if> <R_CB> else
<L_CB> <unmatched_if> <R_CB>
```

2.7. Loops

```
<loop>              → <while_loop> | <foreach_loop>
<while_loop>        → while <L_PR> <expr> <R_PR> <L_CB> <statements>
<R_CB>
<foreach_loop>      → foreach <L_PR> <foreach_expr> <R_PR> <L_CB>
<statements> <R_CB>
```

2.8. Functions

```
<function_name>     → <identifier>
<params>            → <var_new> | <params>
<function_new>      → function <function_name><L_PR><params>
<R_PR><L_CB><statements><R_CB>
<function_call>     → <function_name><L_PR><params><R_PR>
```

2.9. Comments

```
<comment_start>     → /*
<comment_end>       → */
<comment>           → <comment_start> <string> <comment_end>
```

2.10. Sets

2.10.1. Representing Sets and Set Elements

```
<set>               → <L_CB> <set_elements> <R_CB>
<set_elements>      → <set_element> | <set_element> <comma>
<set_elements>
<set_element>       → <int> | <char> | <string>
```

2.10.2. Creating and Deleting Sets

```
<set_new>           → <identifier> = <set>
<set_del>           → delete_set<L_PR><identifier><R_PR>
```

2.10.3. Set Operators

```
<set_add>           → add_to_set<L_PR><set_elements><comma>
<identifier><R_PR>
<set_remove>        → remove_from_set<L_PR><set_elements><comma>
<identifier><R_PR>
<set_union>         → union<L_PR><identifier>
<comma><identifier><R_PR>
<set_intersection>  → intersection<L_PR><identifier>
<comma><identifier><R_PR>
```

2.10.4. Set Relations

```
<is_superset>       → superset<L_PR><identifier><R_PR>
<is_subset>         → subset<L_PR><identifier><R_PR>
<is_emptyset>       → emptyset<L_PR><identifier><R_PR>
<is_equalset>       → equalset<L_RP><identifier><R_PR>
```

2.11. Input and Output Statements

```
<file_path>         → <string>
<read_file>         → read<L_PR><file_path><R_PR>
<write_file>        → write<L_PR><file_path<R_PR>
<input>             → in <read_file>
<output>            → out <identifier> | out <string> | out <int>
                    | out <set>
```

3. DESCRIPTION OF LANGUAGE CONSTRUCTS

3.1. Program

```
<program>     → <main>
<main>        → <statements>
```

program represents the scope of the program. It contains the ***main***, which contains all the statements that need to be executed and start the program

```
<statements> → <statement>
             | <statement> <statements>
```

It consists of either a single statement or multiple statements.

```
<statement>   → <comment>
             | <loop>
             | <expr>
             | <conditional>
             | <function>
```

Statement consists of loops, expressions, conditional statements and function. They are the main building blocks of the program.

3.2. Types

```
<form>        → <int>
             | <bool>
             | <char>
             | <float>
             | <string>
```

Represents the type of a variable, which can be of five types: *float*, *bool* , *char*, *int* and *string*

```
<bool>        → true | false
<char>        → <lowercase> | <uppercase>
<number>      → <int> | <int> <number>
<point>       → .
```

<point> represents the decimal point

```
<float>       → <number> <point> <number>
```

These are different *types* for a variable to be declared in.

```
<string>      → <char> | <char> <string>
<lowercase>   → 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
<uppercase>   → 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 string is made up of single character or combinations of both uppercase and lowercase characters that range from a-z .

```
<int>         → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Consists of all single-digit numbers

```
<sign>        → + | -
```

Contains signs to symbolize if a number is negative or positive

```
<L_CB>        → {
<R_CB>        → }
<L_PR>        → (
<R_PR>        → )
```

Represents the opening and closing tags of curly brackets and parenthesis.

3.3. Variable Identifiers

```
<var_new>           → <form> <identifier>
<identifier>        → <string>
                    | <string> <identifier>
                    | <string> <number>
                    | <string> <number> <identifier>
```

These variable identifiers can be used to make a variable which can be a string of characters or a mixture of characters and integers.

3.4. Assignment Operators

```
<assign>             → <identifier> = <expr>
```

This assignment operator is used to reassign a variable.

```
<assign_new>         → <var_new> = <expr>
```

This assignment operator is used to initialize and assign a new variable.

```
<operator>           → <division_op>
                     | <multiplication_op>
                     | <addition_op>
                     | <subtraction_op>
<division_op>        → /
<multiplication_op>  → *
<addition_op>        → +
<subtraction_op>     → -
```

These are all the possible arithmetic operators used in Setoshi and include all the common operators such as addition, division, multiplication, and subtraction.

3.5. Expressions

```
<expr>               → <identifier> <operator> <identifier>
                     | <identifier> <comparator> <identifier>
```

expr is used to perform arithmetic and comparison operations between two variables.

```
<comparators>        → <expr_and> | <expr_or> | <expr_lt> | <expr_gt>
                     | <expr_lte> | <expr_gte> | <expr_eq>
                     | <expr_eq_strict> | <expr_neq>
                     | <expr_neq_strict>
<expr_and>           → &&
<expr_or>            → ||
```

Logical operators thats can be used alongside other operators to chain multiple true/false conditions

```
<expr_lt>           → <
<expr_gt>           → >
<expr_lte>          → <=
<expr_gte>          → >=
<expr_eq>           → ==
<expr_eq_strict>    → ===
<expr_neq>          → !=
<expr_neq_strict>   → !==
```

General comparison operators that can be used on integers and sets. It is helpful in sets to figure out subsets and supersets

```
<foreach_expr>      → <identifier> as <var_new>
```

This loops through each element of the set

3.6. Conditional Statements

```
<conditional>       → <if>
<if>                → <matched_if> | <unmatched_if>
<matched_if>        → if <expr> <L_CB> <statements> <R_CB> else
<L_CB> <matched_if> <R_CB>
<unmatched_if>      → if <expr> <L_CB> <matched-if> <R_CB> else
<L_CB> <unmatched_if> <R_CB>
```

Conditional statements are used to set a condition for specific statements to be executed based on the state of the condition. It consists of *if* and *else* parts.

3.7. Loops

```
<loop>              → <while_loop> | <foreach_loop>
<while_loop>        → while <L_PR> <expr> <R_PR> <L_CB> <statements>
<R_CB>
```

While loops take an expression with its associated condition in parenthesis and executes the statements inside the loop until the expression condition is no longer true.

```
<foreach_loop>      → foreach <L_PR> <foreach_expr> <R_PR> <L_CB>
<statements> <R_CB>
```

3.8. Functions

```
<function_name>     → <identifier>
<params>            → <var_new> | <params>
<function_new>      → function <function_name><L_PR><params>
<R_PR><L_CB><statements><R_CB>
<function_call>     → <function_name><L_PR><params><R_PR>
```

The function is declared starting with a function word followed by its function name and parameters in parentheses. The statements to be executed from the function are stored within curly brackets

3.9. Comments

```
<comment_start>     → /*
<comment_end>       → */
<comment>           → <comment_start> <string> <comment_end>
```

Comments start with /* and end */. It allows users to write explanations of their code without mixing it up as program code. It increases the readability of Setoshi as a language

3.10. Sets

3.10.1. Representing Sets and Set Elements

```
<set>               → <L_CB> <set_elements> <R_CB>
<set_elements>      → <set_element> | <set_element> <set_elements>
<set_element>       → <int> | <char> | <string>
```

The *set* keyword allows for a user to initialize a set. All elements between the curly brackets are considered as part of the set.

A set element can be one or more and can be of type integer, character and string.

3.10.2. Creating and Deleting Sets

```
<set_new>            → <identifier> = <set>
```

This allows the initialization of a new set.

```
<set_del>            → delete_set<L_PR><identifier><R_PR>
```

This is used to delete an existing set that is no longer required.

3.10.3. Set Operators

```
<set_add>            → add_to_set<L_PR><set_elements>,
<identifier><R_PR>
<set_remove>         → remove_from_set<L_PR><set_elements>,
<identifier><R_PR>
```

These allow the addition and removal of elements from a *set*.

```
<set_union>          → union<L_PR><identifier>,<identifier><R_PR>
<set_intersection>  →
intersection<L_PR><identifier>,<identifier><R_PR>
```

union keyword is used before a parenthesis consisting of two sets to indicate the union operation. It merges both the sets and returns the final set after the union.

intersection keyword follows the same pattern but instead is used for the intersection operation.

3.10.4. Set Relations

```
<is_superset>        → superset<L_PR><identifier><R_PR>
<is_subset>          → subset<L_PR><identifier><R_PR>
<is_emptyset>        → emptyset<L_PR><identifier><R_PR>
<is_equalset>        → equalset<L_RP><identifier><R_PR>
```

These set relations return whether a set is either a superset, subset, or equal to another set. It also returns whether the set is empty or not.

3.11. Input and Output Statements

```
<file_path>          → <string>
<read_file>          → read<L_PR><file_path><R_PR>
<write_file>         → write<L_PR><file_path<R_PR>
<input>              → in <read_file>
<output>             → out <identifier> | out <string> | out <int>
                     | out <set>
```

These are used for I/O operations such as reading or writing to files. A file path is indicated using **file_path**.

4. DESCRIPTION OF NON-TRIVIAL TOKENS

comment:
Comments start with ‘/*’ and end ‘*/’. It allows users to write explanations of their code without mixing it up as program code. It increases the readability of Setoshi as a language as a comment can quickly summarize the function of a statement. It also helps the writability of the code as it helps distinguish between code and explanations.

var_new:
This identifier is used to indicate the type of a variable. e.g. bool, string, integer.

identifier:
A variable identifier that can be string or alphanumeric.

if:
Keyword used in if/else loop statements.

else:
Keyword used in if/else loop statements.

while:
Keyword used to denote the start of a *while* loop statement.

function:
Keyword used to declare functions and marks the start of a function code scope.

delete_set:
A token for deleting a set.

add_to_set:
A token to add to a set.

remove_from_set:
A token to remove an element from a set.

union:

This is a token used to indicate a union/merger of sets.

intersection:

A token similar to the *union* token but used for intersection of sets.

superset:

This is a token to symbolize if a set is a superset of another set.

subset:

This is a token to symbolize if a set is a superset of another set.

emptyset:

This token is used to check if a set is empty.

equalset:

This token is used to check if a set is equal.

read:

This token indicates getting data from a file in the system.

write:

This token is used to denote writing I/O operation to file.

in:

This is used to read from a file.

out:

This token is used to output to the console.

5. EVALUATION OF THE LANGUAGE

5.1. Readability

Setoshi is designed to be readable as it has a manageable set of features. During the design of the language in BNF form, a minimalistic approach was taken. All constructs are given simple, memorable names in English and the constructs are self-descriptive. So, programmers who have prior knowledge of C or Java can easily code in Setoshi. Most of Setoshi's design is orthogonal except for some built-in functions for set operations that require the set parameter to be provided at the end. Data types have been included in the design so variables are explicitly declared to avoid any casting errors. To improve the readability of comments, a starting and ending point has to be specifically provided. This increases the uniformity of the code and improves the readability of Setoshi.

5.2. Writability

Setoshi's orthogonality and simplicity allow for greater writability. A minimal number of built-in functions have been included to avoid any conflicts. Also, only a limited number of words have been reserved in the language. Constructs that can be derived using available constructs have been removed. The set of features included in the language is restricted to make the language manageable and easier to extend in the future. However, Setoshi trades expressiveness for precision. Moreover, its support for abstraction is on the lower end.

5.3. Reliability

Setoshi does not include type-checking despite having data types included in its design. It means data types are not enforced and the language will not throw an exception in case of a data type mismatch. Setoshi has a high level of readability and writability which leads to higher reliability. Exception handling can be added to the language using its parses which will further increase its reliability. Setoshi meets the expectations of the programmer by avoiding ambiguous grammar.

6. CONCLUSION

Setoshi’s readability is the strongest point followed by its writability and reliability factors. The language’s simple design was possible due to its focus on set and set-related operations. Syntax considerations were taken when writing the BNF grammar to use meaningful keywords.