BNF Grammar

<trojan> → <statements>

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

<statement> → <assignment> ; | <if statement> | <for statement> |
<while statement> | <define function>| <functionNum>; | <return statement>; | <comment>

<assignment> → <variable name> = <typeExpr>
| <variableName> = <conditions>
| <variableName> = <setIntExpr>
| <variableName> = <setFloatExpr>
| <variableName> = <setStrExpr>
| <variableName> = <setBoolExpr>
| int <variable name> = <intExpr>
| int <variable name> = <noTypeExpr>
| float <variable name> = <floatExpr>
| float <variable name> = <noTypeExpr>
| str <variable name> = <strExpr>
| str <variable name> = <noTypeExpr>
| bool <variable name> = <boolExpr>
| bool <variable name> = <noTypeExpr>
| SET int <variable name> = <setIntExpr>
| SET int <variable name> = <noTypeExpr>
| SET float <variable name> = <setFloatExpr>
| SET float <variable name> = <noTypeExpr>
| SET str <variable name> = <setStrExpr>
| SET str <variable name> = <noTypeExpr>
| SET bool <variable name> = <setBoolExpr>
| SET bool <variable name> = <noTypeExpr>

<define_params> → <define params> | empty

<define params> → <define param> | <define param>,<define params>

<define param> → <type> <call function> | <type> <variable name>
| SET <type> <variable name>

<define function> → <function type> func $<variable name>(<define_params>){<statements> return <variable name>; } | func <variable name>(<call params>){<statements> return <call function>; } | func <variable name>(<call params>){<statements>}

<call_params> → <call params> | empty

<call params> → <call param> | <call param>,<call params>

<call param> → <call function> | <variable name> | <value>
<call function> → $<variable name>(<call_params>)

<parameters> → <call params> | <digits> | <float> | <letters> | <variable name>
| <setIntegerValue> | <setFloatValue> | <setBoolValue> | <setStrValue>
| <setVarValue>

<functionNum> → <set operators> | <set relations> | <read file> | <call function>

<set function> → <delete set> | <create set> | <set operators> | <set relations><setIntExpr>

<AND or OR> → && | ||

<condition> → <logical expressions> | <noTypeExpr> | <boolExpr>

<conditions> → <condtition> | <condition> <AND or OR> <conditions>

<return statement> → return | return <intExpr> | return <floatExpr> | return <strExpr>
| return <noTypeExpr> | return <setIntExpr> | return <setFloatExpr>
| return <setStrExpr> | return <setBoolExpr>

<delete set> → delete()

<create set> → create(<call params>)

<set operators> → union(<call params>) | intersection(<call_params>)
| addition(<call params>) | multiply(<call_params>)
| subtract(<call params>) | division(<call_params>)
| <variable name>.contain(<call_params>)
| <setValue>.contain(<call params>)
| <variable name>.add(<call_params>)
| <setValue>.add(<call_params>)
| <variable name>.pop(<call_params>)
| <setValue>.pop(<call_params>)

<set relations> → emptyset(<variable name>) | subset(<call params>)
| superset(<call params>)
| equalset(<call params>) | differenceset(<call params>)

<read file> → <variable name> = readFromFile(“<letters>”)

| <variable name> = readFromFile(“<variable name>”)

<write file> → <variable name>.writeToFile(“<letters>”)
| <variable name>.writeToFile(<variable name>)
| <setValue>.writeToFile(“<letters>”)
| <setValue>.writeToFile(<variable name>)

<function type> → void | <type>

<while statement> → while(<logical expressions>){<statements>}

<for statement> → for(<assignment>;<logical expression>;<assignment>){<statements>}

<if statement> → <matched> | <unmatched>

<matched> → if(<logical expression>){ <matched> } else { <matched> }
| any non-if statement

<unmatched>→ if(<logical expression>) {<statement>}
| if(<logical expression>){<matched>} else {<unmatched>}

<logical expressions> → <logical expression> | (<logical expression>) <and or> <logical expressions>

<logical expression> → <call function> | <variable name> | <logical expression vars> <comparison><logical expression vars>

<logical expression vars> → <variable name> | <number> | <call function>

<typeExpr> → <intExpr> | <floatExpr> | <strExpr>

<intExpr> → <intTerm> | <intExpr> + <intExpr> | <intExpr> - <intExpr>

<intTerm> → <intTerm> * <intFactor> | <intTerm> / <intFactor> | <intFactor>

<intFactor> → (<intExpr>) | <digits> | <noTypeExpr>

<setIntExpr> → <setIntTerm> | <setIntExpr> + <setIntExpr>
| <setIntExpr> - <setIntExpr>

<setIntTerm> → <setIntTerm> * <setIntFactor> | <setInTerm> / <setIntFactor>
| <setIntFactor>

<setIntFactor> → (<setIntexpr>) | <setIntegerValue> | <noTypeExpr>

<floatExpr> → <floatTerm> | <floatExpr> + <floatExpr> | <floatExpr> - <floatExpr>
| <floatExpr> + <intExpr> | <intExpr> + <floatExpr> | <floatExpr> - <intExpr>
| <intExpr> - <floatExpr>

<floatTerm> → <floatTerm> * <floatFactor> | <floatTerm> / <floatFactor> | <floatFactor>
| <intTerm> / <floatFactor> | <floatTerm> / <intFactor>
| <intTerm> * <floatFactor> | <floatTerm> * <intFactor>

<floatFactor> → (<floatExpr>) | <float> | <noTypeExpr>

<setFloatExpr> → <setFloatTerm> | <setFloatExpr> + <setFloatExpr>
| <setFloatExpr> - <setIFloatExpr> | <setFloatExpr> + <setIntExpr>
| <setIntExpr> + <setFloatExpr> | <setFloatExpr> - <setIntExpr>
| <setIntExpr> - <setFloatExpr>

<setFloatTerm> → <setFloatTerm> * <setFloatFactor> | <setFloatTerm> / <setFloatFactor>
| <setFloatFactor> | <setIntTerm> * <setFloatFactor>

| <setFloatTerm> * <setIntFactor> | <setIntTerm> / <setFloatFactor>
| <setFloatTerm> / <setIntFactor>
<setFloatFactor> → (<setFloatexpr>) | <setFloatValue> | <noTypeExpr>

<strExpr> → <strTerm> | <strExpr> + <strExpr> | <strExpr> - <strExpr>

<strTerm> → <strTerm> * <strFactor> | <strTerm> / <strFactor> | <strFactor>

<strFactor> → (<strExpr>) | <letters> | <noTypeExpr>

<setStrExpr> → <setStrTerm> | <setStrExpr> + <setStrExpr>

<setStrFactor> → (<setStrExpr>) | <setStrValue> | <noTypeExpr>

<boolExpr> → <logical expression vars> | <noTypeExpr>

<setBoolExpr> → <setBooleans> | <noTypeExpr>

<setExpr> → <setIntExpr> | <setFloatExpr> | <setBoolExpr> | <setStrExpr>

<noTypeExpr> → <noTypeExprTerm>
| <noTypeExpr> + <noTypeExpr> | <noTypeExpr> - <noTypeExpr>

<noTypeExprTerm> → <noTypeExpr> * <noTypeExprFactor> | <noTypeExpr> /
<noTypeExprFactor> | <noTypeExprFactor>

<noTypeExprFactor> → (<noTypeExpr>) | <variable name> | <noTypeExpr> | <call function> | <boolExpr> | <functionNum>

<number> → <digits> | <float>

<addition> → +

<subtraction> → -

<multiplication> → *

<division> → /

<type> → int | float | str | bool | SET int | SET float | SET str | SET bool

<value> → <number> | <bool> | <letters> | <setValue>

<setValue> → <setIntegerValue> | <setFloatValue> | <setBoolValue> | <setStrValue> | <setVarValue>

<setIntegerValue> → { <setIntegers> <digit> }
<setIntegers> → <digit> , <setIntegers> | <digit> | empty

<setFloatValue> → { <setFloats> <float> }

<setFloats> → <float> , <setFloats> | <float> | empty

<setBoolValue> → { <setBooleans> <bool> }

<setBooleans> → <bool> , <setBooleans> | <bool> | empty

<setStrValue> → { <setStrings> <letters> }

<setStrings> → <letters> , <setStrings> | <letters> | empty

<setVarValue> → { <setVariables> <variable name> }

<setVariables> → <variable name> , <setVariables> | <variable name> | empty

<comment> → // <commentLines>

<commentLines> → <commentLine> | <commentLine> <commentLines>

<commentLine> → <letters> | <digits> | <float> | <signs> | <escapeChars> | <spaces>

<spaces> → <space> | <space> <spaces>

<space> → <empty> | null

<empty> →

<bool> → true | false

<variable name> → <letter><alphanumerics> | <letter>

<alphanumerics> → <alphanumerics> <digits> |
<alphanumerics><letters> | <letters> | <digits> |
<alphanumerics> _ | _

<letters> → <letter> | <letter> <letters>

<float> → <digits>.<digits>

<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 | 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
<escapeChar> → \n | \r | \t | \b | \f | \v | \0

1. Variables and sets

Variable names must start with a letter; the rest of it can be letters, digits or underscores. Also, just a single letter is accepted as a variable name.

There are 4 types in our language. These are “int”, “str”, “float”, “bool”. They are recognizable and are essential to every language.

There will be four types of sets. These are “int”, “str”, “float”, “bool”. When a set with a specific type is declared, this sets’ elements should be the same type of declaration. If a variable is initializing with a function or a variable, since we can’t check the type with yacc it won’t be a syntax error.

```
int temp = $dummyFunction(); // correct syntax
int temp = var1; // correct syntaxt
```

Declaring variables type before their initialization increases reliability.

2. Operations for creating and deleting a set

Users can declare a set by typing “SET” then set’s type(“int”, “str”, “float”, “bool”) followed by the set’s name. After the declaration, the user can create the set by typing set’s name followed by a dot and “create” keyword. After the “create” keyword, the user should put the left and right brackets. Between these brackets, the user has the option to add parameter(s)

or not. If there are no parameters, set will be an empty set. If the user uses another set as a parameter, create function will act as a copy constructor and finally, if the user adds multiple parameters, these parameters will be added to the set. Users can delete a set by typing set's name, a dot and "delete" keyword followed by left and right brackets. There isn't more than one way to create and delete sets. It's simple, yet effective.

```
set1.create();
set1.create(set2);
SET int set1 = {1,2,3,4};
set1 = {1,2,3,4};
set1 = var1;
set1 = $dummyFunction;
set1 = set2;
// all of these statements are correct.
```

3. Set operators; e.g., union, intersection, …

There will be seven set operators in our language which are union, intersection, addition, subtraction, multiplication, division and difference. All of these operations will take one or multiple sets as a parameter. However, arithmetic set operators such as addition, subtraction, multiplication and division will only take integer or float sets as the parameter. Union operator unifies all of the given sets and returns a unified set. This unified set will include all of the elements of the given sets. However, intersections between sets will not be included more than once. Also there will be operations such as add and pop to add and delete elements from the set.

```
set1.create(1, 2, 3); // or SET int set1= {1,2,3};
set2.create(2, 3, 4);
set3 = union(set1, set2) //set3 has four elements which are 1, 2, 3, 4
```

Intersection operator detects intersections between sets and returns a new set from these intersection elements.

```
set1.create(1, 2, 3); SET int set1= {1,2,3};
set2.create(2, 3, 4);
set3 = intersection(set1, set2) //set3 has two elements which are 2, 3
```

Addition operator sums up sets elements index by index. Also, sets given as parameters should have the same size. This rule should apply to all arithmetic set operators.

```
set1.create(1, 2, 3); SET int set1= {1,2,3};
set2.create(2, 3, 4);
set3.create(3, 4, 5);
set4 = addition(set1, set2, set3); //set3 has three elements which are 6,9,12
```

Subtraction operator takes sets as parameter, subtract other sets from the first set then return this set.

```
set1.create(12,13,15,16);
set2.create(6,8,9,10);
set3.create(1,3,4,5);
set3 = subtraction(set1, set2); //set3 has four elements which are 6,5,6,6
set3 = subtraction(set1, set2, set3); //set3 has four elements which are 5,2,2,1
```

Differenceset takes two sets as parameter, detects the difference between first set and the second set, then returns a new set with these different elements.

```
set1.create(3,5,6,8);
set2.create(1,5,4,8);
set3 = differenceset(set1, set2); //set3 has two elements which are 3,6
set3 = differenceset(set2, set1); //set3 has two elements which are 1,4
```

add operator takes one parameter. It adds the parameter to end of the set.

```
SET int set1 = {1,2,3,4};
SET int set2 = {6,7};
set1.add(5); // set1 now will be equal to {1,2,3,4,5}
set1.add(set2); // set1 now will be equal to {1,2,3,4,5,6,7}
set1.add(var1);
```

pop operator either takes no parameter or one parameter. In the case of no parameter, the last element in the set will be deleted from the set. In the case of one parameter, the parameter will correspond to the index the user wants to delete.

```
SET int set1 = {1,2,3,4};
int var = 1;
set1.pop(); // set1 now will be equal to {1,2,3}
set1.pop(var); // set1 now will be equal to {1,3}
```

contain operator takes a single parameter and check whether that parameter is in the set or not. If it's in the set, it returns true. Returns false otherwise.

```
SET int set1= {1,2,3};
set1.contain(2) // returns true.
set1.contain(12); // returns false.
```

4. Set relations; e.g., subset, superset, ...

There will be six set relations in our program which are subset, superset, equalset, emptyset and differenceset.

Subset takes two sets as a parameter, then compares the first set to the second set to determine whether it is subset of the second set or not.

```
set1.create(1, 2, 3);
set2.create(1, 2, 3, 4, 5);
subset(set1, set2) // returns true

set1.create(1, 2, 3, 6);
set2.create(1, 2, 3, 4, 5);
subset(set1, set2) // returns false
```

Superset takes two sets as parameter, then compares the first set to the second set in order to determine whether it is superset of the second set or not.

```
set1.create(1, 2, 3, 4, 5);
set2.create(1, 2, 3);
superset(set1, set2); // returns true

set1.create(1, 2, 3, 4, 5);
set2.create(1, 2, 3, 6);
superset(set1, set2); // returns false
```

Equalset takes two sets as parameter, then compares these sets to determine whether they are the same or not. If sets' contents are the same, it will return true.

```
equalset(set1, set2) // returns true if set1 is equal to set2.
```

Emptyset takes a single set as a parameter, then determines whether this set is empty or not. It will return true if the given set is empty.

```
emptyset(set2); // if set2 is not null it returns true, else it returns false.
```

5. Arithmetic Operators and Assignment Operator

Our language supports arithmetic operators and the assignment operator. Arithmetic operators are consist of four elemental arithmetic operations: subtraction(-), addition(+), division(/) and multiplication(*). Assignment operator is used by typing "=". In our language we don't have increment or decrement operations (i++ , i--) since having those operations will make things more complicated, writability and readability will decrease.

```
set1 = set2;
count = count + 1;
```

6. Conditional statements; e.g., if-then, if-then-else

The user can use if statement in two ways. The first way is if-then statement which user types "if" keyword followed by left and right brackets. Between these brackets, there should be a logical expression. Statement in the body will be executed if this logical expression is true. Otherwise, nothing will happen. The second way is if-then-else statement which includes "else" keyword after the if part. Statement in the body of else will be executed if the logical expression belongs to the if part is false. Our language did not support "else if"

keyword. Not having else-if statements increments our language's readability since there are only if's and else's, it's simple.

```
if(subset(set1, set2)) { }
if(equalset(set1, set2)) { } else { }
```

7. Loops

The user has two alternatives for loops: while or for statement. While statement takes logical expressions as parameter, then loops the statements between the brackets until the logical statement becomes false. For statement is consists of three parts: control variable, logical expression variable update. Also, these three parts should be divided by a semicolon. For loop will execute statements in the body as long as the logical expression is true. The format of these loops is the same as other programming languages' formats like Java, C++ and Python. Since users are probably familiar to these languages, it will increase the readability and writability of our language.

```
for(int i = 0 ; i < 10 ; i++) { }
while(emptyset(set1)) { }
```

8. Structures for defining and calling functions

Function is defined by typing the type of function, then typing "func", followed by a variable name which starts with "$". Then, the user can put parameters between the left and right brackets. Multiple parameters should be separated by a comma. If a function has a single parameter, the user can directly put the parameter between brackets. If there are no parameters in a function, then the user shouldn't write anything between the brackets. While defining a function, at the end of the statements, if the function's type is not "void" there should be a return statement. Return statement in void type of functions are optional. Function is called by typing the function's name followed by the left and right brackets. Then, if there are any parameters, the user should add them between the brackets. Type checking while defining functions increases reliability since it removes the possibility of using the wrong type variable as a parameter.

```
// int func $myFunction(int param1, int param2) { // statements
    return 1; }
```

9. Comments

Users can add comments by writing the comment contents after the "//". Our program supports only single-line comments. In this way, our language intents to increase readability and writability because the user that reads and writes comments only need to know the behavior of "//".

```
// this is a comment line.
```