CS 315 - 1

PROJECT PART 2

LANGUAGE DESIGN REPORT

SSL LANGUAGE

GROUP 14

Muhammed Ensar Kaya: 21502089

Elif Özdabak: 21101938

Islomiddin Sodiqov: 21503402

SSL

Description:

SSL(Simple Set Language) is a programming language that deals with variables and expressions of the **Set** type.

BNF

<program> => begin <stmt_list> end

<stmt_list> => <stmt> | <stmt_list> <stmt>

<stmt> => <assignment_stmt> | <if_stmt> | <while_stmt> | <return_stmt> | <read_stmt> | <write_stmt> |
<break_stmt> | <continue_stmt> | <dec_stmt> | <call_stmt> | <comment_stmt>

<assignment_stmt> => <var> = <expr> | <var> = <lbr>(<elements>)<rbr>

<if_stmt> => if <lp> <boolean_expr> <rp><lbr> <stmt_list> <rbr> |
if <lp> <boolean_expr > <rp> <lbr> <stmt_list> <rbr> else <lbr> <stmt_list> <rbr>

<while_stmt> => while <lp> <boolean_expr > <rp> <lbr> <stmt_list> <rbr>

<return_stmt> => return <lp> <expr> <rp> | return <lp> <boolean_expr> <rp>

<read_stmt> => read <lp> <var> <rp>

<write_stmt> => write <lp> <expr> <rp> | write <lp> <element> <rp>

<break_stmt> => break

<continue_stmt> => continue

<dec_stmt> => <function_declaration> | <array_ declaration > | <var_ declaration >

<call_stmt> => <function _call> | <array _call> | <var _call>

<comment_stmt> => <comment> | <block_comment>

<expr> => <var> <operator> <var> | <lp> <expr> <rp> | <function_call> | <variables>

<boolean_expr> => <var> <boolean_operator> <var>|<boolean_expr> <boolean_operator> <boolean_expr>
true | false

<array_ declaration > => <List> <gts> <id> =<lcb> <variables> <rbc>

<variables> => <var> | <variables> , <var>

<var> => <id>

<var_ declaration > => <var> = <expr> | <var> = <elements> |const <var> = <expr> |const<var>=<elements>

<function_declaration> => <function _header> <lbr> <function _body> <rbr>

<function _header> => <result_type> <function _name> <lp> <parameters> <rp>

<function _body> => <stmt_list> <return_stmt>

<result_type> => <type>

<function _name> => <string>

<parameters> => <parameter> | <parameter> , <paremeter>

<parameter> => <var>

<args_list> => <expr> | <args_list> , <expr>

<type> => Set

<function _call> => <function _name> <lp> <args_list> <rp>

<array _call> => <List> = <variables>

<var _call> => <var>

<elements> => <element> | <elements>, <element>

<element> => <id> | <digits> |<chars>

<id> => <string>

<string> => <string_chars>

<string_chars> => <char> | <char> (<chars> | <string_chars>)

<chars> => <char> | <chars> (<char> | <digits>)

<char> => ‘A’ | ‘B’ | ... |’Z’| ‘a’ | ‘b’| ... |’z’| ‘_’ | ‘$’

<digits> => <digit> | <digits> <digit>

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

<operator> => + | ^ | - | * | ‘

<boolean_operator> => ‘&’ | ‘|’ | ‘!’ | ‘==’ | ‘[’ | ‘]’ | ‘!=’ | ‘![’ | ‘!]’

<lp> => ‘(’

<rp> => ‘)’

<lbr> => ‘{’

<rbr> => ‘}’

<gts> => ‘>’

<keywords> => break | continue | else | if | read | return | Set | true | false while | write

Properties:

a) SSL will have a variable of only “**Set**” type.
b) Constant can be defined using the keyword “**const**”

c) Let's assume we have two sets: **A = {1, 2, 3, 4, 5}** and **B = {2, 4, 6}**. The set operators of our language are the following:

- **"+" Union: A + B** will give the union of both sets: **A + B = {1, 2, 3, 4, 5, 6}**
- **"^" Intersection: A ^ B** wil give the intersection of both sets: **A ^ B = {2, 4}**
- **"-" Difference (subtraction): A – B** will give us the set difference, that is, it will give us the set of all elements that are in **A** but not in **B**: **A – B = {1, 3, 5}** and **B – A = {6}**
- **"*" Cartesian product: A * B** will give us a set that contains ordered pairs from **A** and **B:**
- **A * B =** {(1,2), (1,4), (1,6), (2,2), (2,4), (2,6), (3,2), (3,4), (3,6), (4,2), (4,4), (4,6), (5,2), (5,4), (5,6)}
- **" ' " Complement:** Let's say set **U = {1, 2, 3, 4, 5, 6}** is the universal set and **B** is a subset of it. **B'** will give us the set of elements in the universal set that are not in **B**: **B' = {1, 3, 5}**

d) There are 2 set relations in this language. Assume that we have two sets **A={1, 2, 3, 4, 5, 6}** and **B={2, 4, 6}**

- **" [" Subset:** The expression **B [A** will be true if **B** contains some elements of **A.** In our case **B [A** is true, since it contains {2, 4, 6} which are all present in set **A**.
- **"] " Superset:** The expression **A] B** will be true if set **A** contains all elements of set **B** or more. In our case **A] B** is true, since it contains all elements(2,4,6) of **B** and has more elements than that(1,2,3,4,5,6).

Apart from those two relations our language will have boolean expressions as well in order to have support for conditions in <if_stmt> and <while_stmt>. Those are:

- "**&**" **and.** The & of two boolean expressions will be true if both of them are true
- " | " **or.** The | of two boolean expressions will be true if at least one of them is true
- **" ! " not.** ! of a boolean expression will be true if the expression itself is false.
- **" == " equals. A == B** will be true if set A equals to set B.
- **" != " not equal to. A != B** will be true if set A is not equals to set B.

The "&" and "|" will be used between two boolean expressions, meanwhile **"==", "!="** can be used between two sets, and **"!"** can be used to negate the set relations. Examples:

A == B A equals to B

A != B A is not equal to B

A ![B A is not a subset of B

A !] B A is not the superset of B

e) Variables and constants can be initialized or assigned supported values(**set**) using the assignment operator **"=".** Example:

```
Set <a = {a, b, c};

const Set <b = <a;
```

Variables that are declared using the keyword **const** can be initialized or assigned any value only once.

In this language the set elements can only be **numbers**, **characters** and **strings**.

f) Input can be scanned using the keyword **read(var)** in which we should pass a variable as an argument to scan.
Example: Set <a; read(<a); Input example: 1 2 3. After the input the set should be {1, 2, 3}

We can print using the keyword **write()** which also accepts a variable name or any other expression as an argument. It can also print out strings, chars and numbers if passed as arguments.
Example: Set <a = {1, 2, 3};
Set <b = {4,5};
write(<a + <b); #prints {1, 2, 3, 4, 5}

g) We have the **if(condition){<stmt_list>} / else {<stmt_list>}** control flow statement that executes the certain statement(s) if the given condition holds or executes the other given statement(s) otherwise.

Example: **Set <a = {1,2};**
Set <b = {1,2};
if (<a == <b) {
write("equal"); #prints "equal"
}

This language has only **while(condition) {<stmt_list>}** loop which executes the given statements repeatedly as long as the condition holds true.
Example: **Set <a = {1,2}**
Set <b = {1};
while (<b != <a) {
write("not equal"); #prints "not equal" infinitely many times since b is not equal to a
}
The **break** and **continue** statements are also available in this language.

h) **Functions** can be defined and used as shown in BNF form. First we state the return type and after that the name of the function and inside parentheses we should state what parameters does this fuction accept and and use curly braces to open and close the function body. Inside the function body we can have the statements or not but the return statement is required. To call the function we can just write the name of the function and pass the arguments.
Example: **Set <a = {1,2};**
Set <b = {3,4};
write(union(<a, < b)); #prints the union of a and b which is {1,2,3,4}

Set union(Set <a, Set <b) {
return <a + <b;
}

i) There are two types of **comments** in this language: **Line comments** and **block comments**.

The **"#"** sign is used to write line comments. That is everything after this sign on the line is ignored.
Example: **Set <a; #we declared a set**

To write block comments one should state the start and end of the block. In this language to start the block comment we use **"/*"** combination of symbols and to close the block we use **"*/"**
Example: **/*Here goes the comment on the first line**
and the second line*/

To state the end of a statement the semicolon **";"** should be used (as shown in examples above).

j) The language supports arrays (lists) that can be composed of several variables of the type set. To declare and initialize an array we should first write the keyword List and then ">" sign followed by identifier. See the example code below:

Set <d = {1,2,3};
Set <c ={4,5};
List >a = {<d, <c};

k) The **keywords** of this language are: **break, continue, else, if, read, return, Set, while, write.**
You can choose any string as **identifier** as long as they are not the keywords above and don't start with digits and don't contain any unnecessary symbols(Look at bnf for more information). Moreover, each identifier for variables should start with **"<"** symbol as shown in above code examples.

Our motivation to choose those reserved words, symbols, literals and identifiers is that those symols and combinations of characters that make reserved words, symbols, literals and identifiers are easy to learn, since most of them are very similar to human language. Thus it makes **reading** and **writing** in this language easier, thus providing **readability** and **writability**.
For example when talking in human language(English) we often say if this then that, else that. That was our motivation to choose if/else for control flow statements. Same was applied for while loops, break and continue statements, return statement and for read and write and Set.

For those who know set algebra, we tried to make sybols and operators between set variables as similar as possible to set algebra symbols and more meaningful.
For instance, we used **"+"** symbol for set union. Since **set union** gives us a set that includes all elements from both sets without duplicates, one could think of set union as set addition, that is what would the resulting set be if we added two sets. Since sets can't have duplicates, neither can't their addition. That was our motivation to choose **"+"** as the set union.
For intersection we used **" ^ ".** In set algebra it is more of a upside down **"u".** We considered **"^"** as the only symbol that is similar to corresponding symbol in set algebra. For set product and complement we chose " ***** " and " **'** ". The "*****" is the product sign in most languages and people are more familiar with it. In set algebra the set complement is " **'** ", same as in our language.

If one follows all language rules and conventions, there won't be any problems regarding the program reliability and the program will perform as supposed to do. Hence, our language is **reliable**, since there are no internal factors that could lead to program malfunction(as stated before, if you follow the language rules and conventions).