CS315 - Programming Languages

Project 1 Report

FLEX Language Design

Alper Kandemir 21703062 Section 1

Lamia Başak Amaç 21601930 Section 1

BNF:

1.Program

<program>::=<statements>

<statements>::=<statement>;

|<statement>;<statements>

<statement>::=<assignment>

|<expression>

| <declaration>

|<loop>

|<functions>

|<if_stmt>

|<set_op>

|<comment>

| <input>

| <output>

| <read_line>

| <write>

| <console_read>

| <primitive_func>

|<del_set> <identifier>

2.variable identifiers

<identifier> ::= <string>

| < string > <identifier>

| < string ><int>

|< string ><int> <identifier>

<declaration>::= <type_identifier> <identifier>

3.Assignment

<assignment>::= <identifier><assignment_operation><statement>

| <identifier><assignment_operation> <set_list>

| <identifier><assignment_operation> <type_identifier>

<assignment_operation> :: = | -= | += | /= | %= |*= | **=

<expression>::= < type_identifier > <relational> < type_identifier >

| < type_identifier> <arithmetic> < type_identifier>

| < identifier> <relational> < identifier>

| < identifier> <arithmetic> < identifier>

|<type_identifier>

<relational_operation> :: > | <= | >= | < | == | < >

<arithmetic_operation>::= - | +| / | * | %| **

<increment> ::= < identifier >++

<decrement> ::= < identifier > --

4.Sets

< set_list > ::= <num_set>

|<string_set>

| <float_set>

| <char_set>

| <digit_set>

|<all_set>

|<empty_set>

<num_set> ::= { <number> }

| { <number>, <num_set> }

<string_set> ::= { < strings > }

|{ < strings >, <string_set> }

<float_set> ::= { < float > }

|{ < float >, < float_set > }

<char_set>::= { < char> }

|{ < char>, < char_set > }

<digit_set>::= { < digit> }

|{ < digit>, < digit_set > }

<all_set> ::= {<type_identifier>, <all_set>}

<empty_set> ::= {}

//set relations

<set_relations> ::= <set_list> <relation_operator> <set_list>

< relation_operator > ::= <subset>

| <not_belongs>

| <belongs>

|<supset>

|<proper_subset>

|<universal_set>

| <not_subset>

|<not_supset>

|<symmetric_diff>

<subset> ::= Ω

<not_belongs> ::= @’

<belongs> ::= @

<supset> ::= ¬

<proper_subset>::=ß

<not_subset>::= Ω’

<universal_set>::= α

<not_proper_subset>::= ß’

<symmetric_diff>::= Δ

//set operators

<set_opts> ::= <union>

| <intersec>

| <setdiff>

|<complement>

<set_op>::= <identifier> <set_opts> <identifier>

<union> ::= #

<intersec> :: §

<setdiff> ::= ~

<complement>::= ^

5.Boolean type

<boolean> ::= <true>

|<false>

<true> ::= TRUE |true | 1

<false> ::= FALSE |false | 0

6.Types

<type_identifier>::= <int>

| <digit>

| <number>

| <float>

| <string>

| <char>

|<set>

<int>::= <digit>

|<digit><int>

| <sign> <int>

<sign>::= +

|-

<number>::= <digit>

| <digit><number>

| -<number>

<float>::= <number>.<int>

|<int>.<int>

<string> ::= ‘<chars>’

| ‘<chars> <string>’

<digits>::= <digit>

|<digit> <digits>

<char> ::= “<chars>”

<chars> ::= a|b|c|ç|d|e|f|g|ğ|h|ı|i|j|k|l|m|n|o|ö|p|r|s|ş|t|u|ü|v|y|z|!|#|£|$|%|&|_|

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

<new_line> ::= \n

//reserved keywords

<set>::= set

<del_set>::= del_set

<boolean>::= bool

<int>::= int

<float>::= fl

<string>::= str

<number>::=num

<char>::= ch

7. Loops

<loops>::= <while>

|<for>

|<do_while>

<while>::= while (<expression>) { <statements> }end

<for>::= for (<assignment> :: <expression> :: <expression>) { <statements> }end

<do_while>::= do { <statements> } while (<expression>)end

8.I/O:

<input>::= to <identifier>

<output>::= display <identifier>

| display <type_identifier>

<read_line> ::= read(<string>)

<write>::= write(<string> in <string>)

|write(<identifier> in <string>)

<console_read>::= console.read(<string>)

9.Conditional Statements

<if_stmt>::= <if>
|<switch>

<if>::= <matched>
|<unmatched>

<matched>::= if (<expression>) <matched> else <matched>

<unmatched>::= if (<expression>) <statement>
|if (<expression>) <matched> else <unmatched>

<switch>::= switch (<ident>)
|<case>
|<case><switch>

<case>::= case<expression>:<statements> cut
|<default-case>

<default-case>::= default : <statements> cut

10.Primitive:

<primitive_func>::= <max>

|<min>

|<avg>

|<sum>

|<mul>

|< cardinality >

|< dot_product >

|< cross_product >

|<abs>

|<find>

|<findset>

| <flush>

| <venn>

| <insert>

| <del>

| <size>

<max> ::= < identifier >.max()

<min> ::= < identifier >.min()

<avg> ::= < identifier >.avg()

<sum> ::= < identifier >.sum()

<mul> ::= < identifier >.mul()

<cardinality> ::= < identifier >.cardinality()

< dot_product > ::= < identifier >. dot_product(< identifier >)

< cross_product > ::= < identifier >.cross_product(< identifier >)

<abs>::= < identifier >.abs()

<find>::= < identifier >.find(< identifier >)

<findset>::= < identifier >.findset(< identifier >)

<flush>::= < identifier >.flush()

<venn>::= < identifier >.venn(< identifier >)

<insert>::= < identifier >.insert(< identifier >)

<del>::= < identifier >.del(< identifier >)

<size>::= < identifier >.size()

11.Fucntion definitions and calls

<function>::= < type_identifier > < identifier >(<parameters>){statements}

| < identifier >(<parameters>) {statements}

| < identifier >(< identifier >):

| < identifier >():

| < identifier >(< identifier >()):

| < identifier >(< identifier >(<parameters>)):

<parameters>::= <parameter>

|<parameter>,<parameters>

<parameter> ::= < type_identifier > < identifier >

12.Comments

<comment>::= ^<string>^

Explanation of FLEX Language:

1.<program>::=<statements>

Program starts with this line which includes statements.

2.<statements>::=<statement>;

|<statement>;<statements>

Statements contain only one statement or several statements with recursive line.

3.<statement>::=<assignment>

|<expression>

| <declaration>

|<loop>

|<functions>

|<if_stmt>

|<set_op>

|<comment>

| <input>

| <output>

| <read_line>

| <write>

| <console_read>

| <primitive_func>

|<del_set> <identifier>

A type of statement can be as above. These form the main structure of FLEX language.

2.<identifier> ::= <string>

| < string > <identifier>

| < string ><int>

|< string ><int> <identifier>

Identifiers contains string only or string + int or string + int + string... However, cannot start with integers.

<declaration>::= <type_identifier> <identifier>

It's a basic variable declaration. First write the type of the variable and then write the variable name.

3.<assignment>::= <identifier><assignment_operation><statement>

| <identifier><assignment_operation> <set_list>

| <identifier><assignment_operation> <type_identifier>

Variables can be assigned to all context of statement or a set or integer, strings, all other types in type_identifier section.

<assignment_operation> :: = | -= | += | /= | %= |*= | **=

Different options for assigning the variables. % means modulo, * means multiplication, ** means power.

<expression>::= < type_identifier > <relational> < type_identifier >

| < type_identifier> <arithmetic> < type_identifier>

| < identifier> <relational> < identifier>

| < identifier> <arithmetic> < identifier>

|<type_identifier>

Variable or variable names can be used in expression. Then, relational or arithmetic operations can be used or expression can be just an integer, float, string... all types in <type_identifier>.

<relational_operation> :: > | <= | >= | < | == | < >

< > means not equals

<arithmetic_operation>::= - | +| / | * | %| **

<increment> ::= < identifier >++

Increase the variable

<decrement> ::= < identifier > --

Decrease the variable

4.< set_list > ::= <num_set>

|<string_set>

| <float_set>

| <char_set>

| <digit_set>

|<all_set>

|<empty_set>

<num_set> ::= { <number> }

| { <number>, <num_set> }

This set includes only number type.

<string_set> ::= { < strings > }

|{ < strings >, <string_set> }

This set includes only string type.

<float_set> ::= { < float > }

|{ < float >, < float_set > }

This set includes only float type.

<char_set>::= { < char> }

|{ < char>, < char_set > }

This set includes only char type.

<digit_set>::= { < digit> }

|{ < digit>, < digit_set > }

This set includes only digit type.

<all_set> ::= {<type_identifier>, <all_set>}

This set includes all types in <type_identifier>.

<empty_set> ::= {}

This is empty set.

//set relations

<set_relations> ::= <set_list> <relation_operator> <set_list>

This used for relation between two sets.

< relation_operator > ::= <subset>

| <not_belongs>

| <belongs>

|<supset>

|<proper_subset>

|<universal_set>

| <not_subset>

|<not_supset>

|<symmetric_diff>

<subset> ::= Ω

The symbol represents subset operator

<not_belongs> ::= @’

This symbol means does not belong to in math

<belongs> ::= @

This symbol means belongs to in math

<supset> ::= ¬

This demonstrates super set

<proper_subset>::=ß

This demonstrates proper subset

<not_subset>::= Ω’

This demonstrates not subset of some set.

<universal_set>::= α

This represents universal set symbol.

<not_proper_subset>::= ß’

This represents not proper subset symbol.

<symmetric_diff>::= Δ

This demonstrates symmetrical differential.

//set operators

<set_opts> ::= <union>

| <intersec>

| <setdiff>

|<complement>

<set_op>::= <identifier> <set_opts> <identifier>

Set operation can be defined as a variable name, an operator and a variable name.

<union> ::= #

This represents union operator symbol.

<intersec> :: §

This represents intersection operator symbol.

<setdiff> ::= ~

This represents set difference operator symbol.

<complement>::= ^

This represents complement operator symbol.

5.<boolean> ::= <true>

|<false>

<true> ::= TRUE |true | 1

<false> ::= FALSE |false | 0

Boolean type includes true and false values that are not case sensitive. Also, true and false can be represented as 1 and 0.

6.<type_identifier>::= <int>

| <digit>

| <number>

| <float>

| <string>

| <char>

|<set>

<int>::= <digit>

|<digit><int>

| <sign> <int>

Integers can be positive or negative.

<sign>::= +

|-

<number>::= <digit>

| <digit><number>

| -<number>

Number type does not have plus sign.

<float>::= <number>.<int>

|<int>.<int>

float can only be positive

<string> ::= ‘<chars>’

| ‘<chars> <string>’

String starts with the symbol ‘ and end with the symbol ’ .

<char> ::= “<chars>”

<chars> ::= a|b|c|ç|d|e|f|g|ğ|h|ı|i|j|k|l|m|n|o|ö|p|r|s|ş|t|u|ü|v|y|z|!|#|£|$|%|&|_|

All characters in the language

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

<new_line> ::= \n

new line character

//reserved keywords

<set>::= set

<del_set>::= del_set

To delete the set completely

<boolean>::= bool

<int>::= int

<float>::= fl
<string>::= str

<number>::=num

<char>::= ch

7. <loops>::= <while>
|<for>
|<do_while>

<while>::= while (<expression>) { <statements> }end

While loop includes expression in the parenthesises the statement starts with curly left bracket and finishes with right curly bracket. While loop ends with the keyword “end”.

<for>::= for (<assignment> :: <expression> :: <expression>) { <statements> }end

“::” is used for separation and the loop ends with the keyword “end”.

<do_while>::= do { <statements> } while (<expression>)end

Statement are in the curl brackets and the expression in parentheses. the loop ends with the keyword “end”.

8.<input>::= to <identifier>

To get input from keyboard to a variable.

<output>::= display <identifier>

| display <type_identifier>

To print variables to console also print integers, strings, chars also set.

<read_line> ::= read(<string>)

To read sets from file. File name should be string type like read('test.txt').

<write>::= write(<string> in <string>)

|write(<identifier> in <string>)

To write file to file or write variables to a file. Example, write ('test1.txt' in 'test2.txt'). This write tes1.txt to end of test2.txt. File name should be string type and indicate with "in" keyword. Example write(set1 in 'test1.txt').

<console_read>::= console.read(<string>)

To read from console that indicates with string.

9.<if_stmt>::= <if>
|<switch>

<if>::= <matched>
|<unmatched>

If is conditional statement and it has two types marched and unmatched to prevent ambiguity in the language

<matched>::= if (<expression>) <matched> else <matched>

If the expression in the parenthesises is correct, matched will be taken otherwise else part will be taken

<unmatched>::= if (<expression>) <statement>
|if (<expression>) <matched> else <unmatched>

<switch>::= switch (<identifier>)
|<case>
|<case><switch>

Switch takes identifier and do the cases.

<case>::= case<expression>:<statements> cut
|<default-case>

Case evaluates the expression and compare with identifier in the statements. Proper statement will be executed.

<default-case>::= default : <statements> cut

Execute the default case in no case scenario.

10.<primitive_func>::=<max>
|<min>
|<avg>
|<sum>
|<mul>
|< cardinality >
|< dot_product >
|< cross_product >
|<abs>
|<find>
|<findset>
| <flush>
| <venn>
| <insert>
| <del>
| <size>

<max> ::= < identifier >.max()

To find max element in the set.

<min> ::= < identifier >.min()

To find min element in the set.

<avg> ::= < identifier >.avg()

To find average of elements in the set.

<sum> ::= < identifier >.sum()

To find sum of all elements in the set.

<mul> ::= < identifier >.mul()

To multiply of all elements in the set.

<cardinality> ::= < identifier >.cardinality()

To find number of elements in the set.

< dot_product > ::= < identifier >. dot_product(< identifier >)

To find dot product of two sets.

< cross_product > ::= < identifier >.cross_product(< identifier >)

To find cross product of two sets.

<abs>::= < identifier >.abs()

Converting the negative elements to the positive elements in the set.

<find>::= < identifier >.find(< identifier >)

Searching the variable in the set.

<findset>::= < identifier >.findset(< identifier >)

Searching in the set for a certain subset.

<flush>::= < identifier >.flush()

To clear all values of the set but not delete the set itself.

<venn>::= < identifier >.venn(< identifier >)

Creating a venn diagram for two set.

<insert>::= < identifier >.insert(< identifier >)

Inserting a variable to a set.

<del>::= < identifier >.del(< identifier >)

Deleting a variable from the set.

<size>::= < identifier >.size()

Find size of the set.

11.<function>::= < type_identifier > < identifier >(<parameters>){statements}

| < identifier >(<parameters>) {statements}

| < identifier >(< identifier >)

| < identifier >()

| < identifier >(< identifier >())

| < identifier >(< identifier >(<parameters>))

Functions declarations starts with its return type then function name. Parameters are in the parenthesises and separated with comma as below. Function calls start with function name, and function calls can be nested like foo1(foo2()).

<parameters>::= <parameter>

|<parameter>, <parameters>

<parameter> ::= < type_identifier > < identifier >

Parameter has its type and name.

12.<comment>::= ^<string>^

Comments are starts with ^ symbol and ends with ^ symbol again. Comment text will be the inside of the two symbols.

Reserved Tokens:

Set: Reserved for stating set data type.

del_set: Reserved for deleting set data type.

bool: Reserved for stating Boolean data type.

int: Reserved for stating integer data type.

fl: Reserved for stating floating points number data type.

str: Reserved for stating string data type.

num: Reserved for stating special number data type.

ch: Reserved for stating character data type.

while: Reserved for stating while loops.

end: Reserved for loops.

for: Reserved for stating for loops.

do: Reserved for stating do while loops.

display: Reserved for calling display function.

to: Reserved for taking input from keyboard.

read: Reserved for calling read function.

in: Reserved for using in write function.

console.read: Reserved for console readings.

if: Reserved for stating if statements.

else: Reserved for stating else statements.

switch: Reserved for stating switch statements.

case: Reserved for stating cases.

cut: Reserved for switch-case statements.

default: Reserved for switch-case statements.

max: Reserved for calling max primitive function.

min: Reserved for calling min primitive function.

avg: Reserved for calling avg primitive function.

sum: Reserved for calling sum primitive function.

mul: Reserved for calling mul primitive function.

cardinality: Reserved for calling cardinality primitive function.

dot_product: Reserved for calling dot_product primitive function.

cross_product: Reserved for calling cross_product primitive function.

abs: Reserved for calling abs primitive function.

find: Reserved for calling find primitive function.

findset: Reserved for calling findset primitive function.

flush: Reserved for calling flush primitive function.

venn: Reserved for calling venn primitive function.

insert: Reserved for calling insert primitive function.

del: Reserved for calling del primitive function.

size: Reserved for calling size primitive function.

True: Reserved for when Boolean is true.

true: Reserved for when Boolean is true.

False: Reserved for when Boolean is false.

false: Reserved for when Boolean is false.

Language Evaluation

Readability:

The Flex language can be traced easily while reading the program. It has understandable variable names, understandable function ends and so on. While loop, for loop, do-while loop statements have a proper ending as “end”. Moreover, switch case has also other ending as “cut”. Our language has “for, while, if” statements for understandability. The reason why Flex uses that idea is that programmers use different languages which have various syntax properties. We didn't want to create a different program because we want users to adapt to Flex program without a prior knowledge of the language. We want to increase the readability without using any extraordinary tokens or symbols to use. If someone is not really familiar with these kinds of languages, even they can easily learn our language because of its convenience.

Writability:

User can experience flexibility with the Flex language while they are writing codes. Also, our language is created to easily adapt and be used fast without a misunderstanding. Moreover, it is created with a different assigning order. It can be used with right associativity when assigning or expressions. We think that Flex language can be used all around the world. Also, we expanded it by adding Turkish letters in our language. This can increase the desire to use our program language. Inversely, Flex language has similar structures with popular programs all around the world.

Reliability:

The Flex language offers users reliability. Through both readability and writability, tokens are created for increasing reliability. For instance, we designed the while loop with "while (some expression)", and after writing the calculation that the user wants, we created a token "end" to understand that user comes to ending the while loop. It increases understandability and it becomes clear that users can use our language easily. Another example, we have implemented tokens as "to" and "display". These are used for input/output operations that increase reliability in giving a more accurate usability without a misunderstanding. Finally, in Flex language, the user can write variables, functions etc. that are very understandable by the user and also shows the expected outputs.