CS 315 Programming Languages

Project 2

Group 16

Name of the language:Bakusa

Batuhan Budak | 21704212 | Section 3

Kutsal Bekçi | 21602163| Section 3

Saidcan Alemdaroğlu | 21602521 | Section 3

Instructor: Karani Kardaş

Name of the Language: Bakusa

BNF Description of the Bakusa Language

1) Program

<program>::=<begin>|<begin><stmts>
<stmts>::=<stmt> | <stmts><stmt>
<stmt>::=<while_stmt>|<for_stmt>|<do_while_stmt>|<conditional_stmts>|<input_stmt>|<output_stmt>|<return_stmt>|<expression>|<declaration>|<initialization>|declaration_and_initialization>

2) Variable Identifiers

<type> ::= <var>
<var> ::= <boolean> | <char> | <integer> | <string> | <double> | <signedDouble> | <signedInteger>
<aritchmetic_var>::= <integer> | <double> | <signedInteger> | <signedDouble>
<boolean> ::= <true> | <false>
<char> ::= <char_identifier><letter><char_identifier>
<integer> ::= <digit> | <integer><digit>
<string> ::= <string identifier><sentence><string identifier>
<sentence> ::= <word> | <word><letter> | <word><digit> | <digit><word>
<word> ::= <letter> | <word><letter> | <digit><letter> | <digit><word>
<double> ::= <integer><dot><integer> | <integer><dot><digit> | <digit><dot><integer> | <digit><dot><digit>
<signedInteger>::= <sign><integer>
<signedDouble>::= <sign><double>

3) Symbols

<lowercase_char> ::=
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_char> ::=
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

<string_identifier> ::= “

<char_identifier> ::= ‘

<double_identifier> ::= .

<identifier> ::= <word> | <identifier> <digit>|<word><identifier>

<LP> ::= (

<RP> ::=)

<LB> ::= {

<RB> :: = }

<LSB> ::= [

<RSB> ::=]

<comma> ::= ,

<dot> ::= .

<sign> ::= + | -

<sl_comment_sign> ::= //

<semicolon> ::= ;

<underscore> ::= _

<equal_sign> ::= =

<space> ::= “ “

<degree> ::= °

<add_sign>::= +

<sub_sign>::= -

<mult_sign>::= *

<div_sign>::= /

<mod_sign>::= %

<qst_mark>::= ?

<exc_mark>::= !

<lst_sign>::= <

<gtt_sign>::= >

<and_sign>::= &&

<or_sign>::= ||

<return>::= return

<int_type>::=

4) Declaration and Initialization

<declaration>::=<type><space><identifier><semicolon>
<initialization>::=<boolean_initialization> | <char_initialization> | <double_initialization> | <signedDouble_initalization> | <signedInteger_initialization>

<boolean_initialization>::=<identifier><space><assign_op><boolean><semicolon>
<char_initialization>::=<identifier><space><assign_op><char><semicolon>
<integer_initialization>::=<identifier><space><assignment_operator><integer><semicolon>
<double_initialization>::=<identifier><space><assignment_operator><double><semicolon>
<signedDouble_initialization>::=<identifier><space><assignment_operator><signedDouble><semicolon>
<signedInteger_initialization>::=<identifier><space><assignment_operator><signedInteger><semicolon>

<declaration_and_initialization>::=<type><space><identifier><assignmen_operator><var><semicolon>

5) Truth Variables

<true> ::= “T” | ‘T’ | true
<false> ::= “F” | ‘F’ | false

6) Operators

a) Assignment Operators

<assignment_operator> ::= <assign_op> | <add_assignment_op> | <sub_assignment_op> | <mult_assignment_op> | <div_assignment_op>

<assign_op> :: = <equal_sign>
<add_assignment_op> :: = <add_sign><equal_sign>
<sub_assignment_op> :: = <sub_sign><equal_sign>
<mult_assignment_op> :: = <mult_sign><equal_sign>
<div_assignment_op> :: = <div_sign><equal_sign>

b) Arithmetic Operators

<arithmetic_operator> ::= <addition> | <subtraction> | <multiplication> | <division> | <mod>
<addition> ::= <add_sign>
<subtraction> ::= <sub_sign>
<multiplication> ::= <mult_sign>
<division> ::= <div_sign>
<mod> ::= <mod_sign>

c) Relational Operators

<relational_operator> ::= <equivalence_check> | <not_equal_check> | <less_than> | <greater_than> | <equal_and_less_than> | <equal_and_greater_than>

<equivalence_check> ::= <equal_sign><qst_mark>
<not_equal_check> ::= <equal_sign><exc_mark>
<less_than> ::= <lst_sign>
<greater_than> ::= <gtt_sign>
<equal_and_less_than> ::= <lst_sign><equal_sign>
<equal_and_greater_than> ::= <gtt_sign><equal_sign>

d) Logical Operators

<logical_operator> ::= <and>|<or>
<and> ::= <and_sign>
<or> ::= <or_sign>

7) Precedence and Associativity of the Operators

Level	Operator	Description	Associativity
8	()	Parentheses	left to right
7	* / %	Multiplication Division Mod	left to right

6	+ -	Addition Subtraction	left to right
5	< <= > >=	Relational	not associative
4	=? =!	Equivalence check Not equal check	left to right
3	&&	Logical AND	left to right
2	||	Logical OR	left to right
1	= += -= /= *=	Assignment	right to left

- ❑ Level 8 has the highest precedence, level 1 has the lowest precedence. When level increases the precedence also increases.
- ❑ Parentheses have highest precedence.
- ❑ Multiplication, division and mod have second highest precedence.
- ❑ Addition and subtraction have the third highest precedence.
- ❑ Relational operators such as <, <=, >, >= have the fourth highest precedence.
- ❑ Logical AND has the third lowest precedence.
- ❑ Logical OR has the second lowest precedence.
- ❑ Assignment has the lowest precedence.

----------Missing precedence rules for logical operations----------

<assign>::= <id> = <expression>
<arithmetic_var>::=<integer> | <double> | <signedInteger> | <signedDouble>

<expression>::=<expression><addition><term> | <expression><subtraction><term> | <term>
<term>::= <term><multiplication><factor> | <term><division><factor> | <factor>
<factor>::= <LP><expr><RP> | <id>

8) Expressions

<arithmetic_var>::=<integer> | <double> | <signedInteger> | <signedDouble>
<expression> ::= <assignment_expression> | <arithmetic_expression> |
<logical_expression> | <relational_expression> | <logical_expression> |<function_call>|
<primitive_function>

a) Assignment Expressions

<assignment_expression>::= <assign_exp> | <add_assignment_exp> |
<sub_assignment_exp> | <mult_assignment_exp> | <div_assignment_exp>

<assign_exp>::= <identifier><assign_op><arithmetic_var> |
<identifier><assign_op><function_call>l <identifier><assign_op><logical_expression> |
<identifier><assign_op><relational_expression> |
<identifier><assign_op><primitive_function> |
<identifier><assign_op><arithmetic_expression>
<add_assignment_exp>::=<identifier><add_assignment_op><arithmetic_var>|<identifier><a
dd_assignment_op><identifier>
<sub_assignment_exp>::=<identifier><sub_assignment_op><arithmetic_var>|
<identifier><sub_assignment_op><identifier>
<mult_assignment_exp>::=<identifier><mult_assignment_op><arithmetic_var>|
<identifier><mult_assignment_op><identifier>
<div_assignment_exp>::=<identifier><div_assignment_op><arithmetic_var>|
<identifier><div_assignment_op><identifier>

b) Arithmetic Expressions

<arithmetic_expression>::=<addition_exp> | <subtraction_exp> | <multiplication_exp> |
<division_exp>

<addition_exp>::= <arithmetic_var><addition><arithmetic_var> | <identifier>
<addition><arithmetic_var> |<arithmetic_var><addition><identifier>|
<identifier><addition><identifier>
<subtraction_exp>::=<arithmetic_var><subtraction><arithmetic_var> |
<identifier><subtraction><arithmetic_var> |<arithmetic_var><subtraction><identifier>|
<identifier><subtraction><identifier>

<multiplication_exp>::=<arithmetic_var><multiplication><arithmetic_var>|<arithmetic_var><multiplication><identifier>|<identifier><multiplication><arithmetic_var>|<identifier><multiplication><identifier>

<division_exp>::=<arithmetic_var><division><arithmetic_var>|<arithmetic_var><division><identifier>|<identifier><division><arithmetic_var>|<identifier><division><identifier>

<mod_exp>::=<integer><mod><integer> | <signedInteger><mod><integer>
|<integer><mod><identifier>|<identifier><mod><integer> |
<signedInteger><mod><identifier>|<identifier><mod><identifier>

c) Relational Expressions

<relational_expression>::=<equivalence_check_exp> | <not_equal_check_exp> |
<less_than_exp> | <greater_than_exp> | <equal_and_less_than_exp> |
<equal_and_greater_than_exp>

<equivalence_check_exp>::= <var><equivalence_check><var>|
<identifier><equivalence_check><identifier> |
<var><equivalence_check><identifier>|<identifier><equivalence_check><var>

<not_equal_check_exp>::=<var><not_equal_check><var><identifier><not_equal_check><identifier> | <var><not_equal_check><identifier>|<identifier><not_equal_check><var>

<less_than_exp>::=<arithmetic_var><less_than><arithmetic_var>|<identifier><less_than><arithmetic_var>|<arithmetic_var><less_than><identifier>|<identifier><less_than><identifier>

<greater_than_exp>::=<arithmetic_var><greater_than><arithmetic_var>|<identifier><greater_than><arithmetic_var>|<arithmetic_var><greater_than><identifier>|<identifier><greater_than><identifier>

<equal_and_less_than_exp>::=<arithmetic_var><equal_and_less_than><arithmetic_var>|<identifier><equal_and_less_than><arithmetic_var>|<arithmetic_var><equal_and_less_than>
<identifier>|<identifier><equal_and_less_than><identifier>

<equal_and_greater_than_exp>::=<arithmetic_var><equal_and_greater_than><arithmetic_var>|<identifier><equal_and_greater_than><arithmetic_var>|<arithmetic_var><equal_and_greater_than><identifier>|<identifier><equal_and_greater_than><identifier>

d) Logical Expressions

<logical_expression>::= <and_exp> | <or_exp>

<and_exp> ::= <identifier> <and><identifier> | <boolean><and><identifier>
|<identifier><and><boolean> | < boolean><and><boolean>
<or_exp> ::= <identifier> <or><identifier>
|<boolean><or><identifier>|<identifier><or><boolean>|<boolean><or><boolean>

9) Loops

<while_stmt> ::= while<LP><expression> <RP> <LB> <stmts> <RB>
<for_stmt> ::=
for<LP><initialization><semicolon><expression><semicolon><expression><semicolon><RP
><LB><stmts><RB>
<do_while_stmt> ::= do<LB> <stmts> <RB> while <LP> <expression><RP><semicolon>

10) Conditional Statements

<conditional_stmts> ::= <if_stmt> | <if_stmt><else_stmt> | <if_stmt><else_it_stmt_ctn> |
<if_stmt><else_it_stmt_ctn><else_stmt>

<if_stmt> ::= if <LP><logical_expression><RP><LB><statement><RB>
<else_stmt> ::= else<LB><statement><RB>
<else_if_stmt> ::= else if<LP><logical_expression><RP><LB><statement><RB>
<else_it_stmt_ctn> ::= <else_if_stmt> | <else_if_stmt><else_it_stmt_ctn>
<if_else_stmt> ::= <if_stmt><else_stmt>

11) Comments

<single_line_comment> ::= <sl_comment_sign><sentence>

12) Statements for Input / Output

<input>::= input
<input_stmt>::=<type><identifier><assignment_operator>bksin<LP><RP>|<identifier><assig
nment_operator><bksin><LP><RP>
<input>::= <boolean> | <char> | <integer> | <string> | <double> | <signedDouble> |
<signedInteger>
<output_stmt>::=bksout<LP><output><RP>

<output>::=<var><underline><output> | <var>| <function_call> | <primitive_function>

13) Function Definitions and Function Calls

<return_stmt>::= <return><space><identifier><semicolon>
<parameter> ::= <type><identifier> | <type><identifier><coma><space><parameter>

<function_definition>::= <returned_function> | <void_function> | <paraVoid_function> |
<paraReturn_function>

<returned_function> ::= <type><identifier><LP><RP><LB><stmts><return_stmt><RB>
<void_function> ::= void <identifier><LP><RP><LB><stmt><RB>
<paraVoid_function> ::= void <identifier><LP><parameter><RP><LB><stmts><RB>
<paraReturn_function> ::=
<type><identifier><LP><parameter><RP><LB><stmts><return_stmt><RB>

<function_call>::=<identifier><LP><RP><semicolon> |
<identifier><LP><parameter><RP><semicolon>

14) Primitive Functions

<primitive_functions>::=<inclinometer> | <barometer> | <thermometer> | < accelerometer> |
<videoCamera> | <timer> | <host> | <port>
<inclinometer> ::= INCLINATION
<altimeter>::=ALTITUDE
<thermometer>::=TEMPERATURE
<accelerometer>::=ACCELERATION
<videoCamera>::= ON | OFF | TakePicture
<timer>::=TIMESTAMP
<host> ::= HOST
<port> ::= PORT

Explanation for Bakusa Language Constructions

1. **<program>::=<begin><LP><RP><LB><stmts><RB>**

This non-terminal shows that our language is consisting of statements. It begins with the begin command and left curly bracelet and ends with the right curly bracelet and end command.

2. **<stmts>::=<stmt> | <stmts><stmt>**

This non-terminal shows that statements in the language are a list of the statements.

3. **<var> ::= <boolean> | <char> | <integer> | <string> | <double> | <signedDouble> | <signedInteger>**

This terminal shows the existing variables in Bakusa language.

4. **<aritchmetic_var>::= <integer> | <double> | <signedInteger> | <signedDouble>**

This terminal shows the variables that can be used in arithmetic operations.

5. **<boolean> ::= <true> | <false>**

This terminal is created to show types of boolean variables.

6. **<char> ::= <char_identifier><letter><char_identifier>**

This terminal shows how Bakusa language distinguishes char variables.

7. **<integer> ::= <digit> | <integer><digit>**

This terminal shows integers consisting from digits.

8. **<string> ::= <string identifier><sentence><string identifier>**

This terminal shows how Bakusa language distinguishes string variables.

9. **<sentence> ::= <word> | <word><letter> | <word><digit> | <digit><word>**

This terminal shows sentences in Bakusa language consisting of words, words and letters or words and digits.

10. **<word> ::= <letter> | <word><letter> | <digit><letter> | <digit><word>**

This terminal shows words in Bakusa language consisting of letters or letters and digits.

11. **<double> ::= <integer><dot><integer> | <integer><dot><digit>|<digit><dot><integer> | <digit><dot><digit>**

This terminal shows double variables in Bakusa language are a combination of integers, digits and a dot.

12. **<signedInteger>::=<sign><integer>**

This terminal shows integers can be signed.

13. **<signedDouble>::=<sign><double>**

This terminal shows doubles can be signed.

14. **<lowercase_char> ::= 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**

This terminal represents lowercase characters in Bakusa language.

15. **<uppercase_char> ::= 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**

This terminal represents uppercase characters in Bakusa language.

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

This terminal represents digits in Bakusa language.

17. **<declaration>::=<type><space><identifier><semicolon>**

This non-terminal shows the grammar of declaring a new variable in Bakusa language.

18. **<initialization>::=<boolean_initialization> | <char_initialization> | <double_initialization> | <signedDouble_initalization> | <signedInteger_initialization>**

This non-terminal shows the types of initializations in Bakusa language. In Bakusa language users can initialize listed variables through using defined grammars.

19. **<declaration_and_initialization>::=<type><space><identifier><assignmen_operator><var><semicolon>**

This non-terminal shows the grammar of declaring and initializing a variable at the same line. In Bakusa language users have a choice to declare and initialize a variable in the same line.

20. **<operators>::=<assignment_operator> | <arithmetic_operator> | <relational_operator> | <relational_operator> | <logical_operator>**

This non-terminal lists the operator types in Bakusa language.

21. **<assignment_operator> ::= <assign_op> | <add_assignment_op> | <sub_assignment_op> | <mult_assignment_op> | <div_assignment_op>**

This non-terminal lists the assignment operators in Bakusa language. Users can assign a new variable via using a variety of assignment operators in Bakusa language.

22. **<arithmetic_operator> ::= <addition> | <subtraction> | <multiplication> | <division> | <mod>**

This non-terminal lists the arithmetic operators in Bakusa language. Users can use these operators for their mathematical operations.

23. **<relational_operator> ::= <equivalence_check> | <not_equal_check> | <less_than> | <greater_than> | <equal_and_less_than> | <equal_and_greater_than>**

This non-terminal represents the relational operators in Bakusa language. Relational operators in Bakusa language can be used to compare variables.

24. **<logical_operator> ::= <and>|<or>**

This non-terminal shows the logical operators in Bakusa language. Logical operators might be used to connect two or more expressions.

25. **<expression> ::= <assignment_expression> | <arithmetic_expression> | <logical_expression> | <relational_expression> | <logical_expression>**

This non-terminal lists the expressions in Bakusa language.

26. **<assignment_expression>::= <assign_exp> | <add_assignment_exp> | <sub_assignment_exp> | <mult_assignment_exp> | <div_assignment_exp>**

This non-terminal lists the assignment expressions in Bakusa language. Expressions have their defined grammar to assign variables in Bakusa language. Users can assign variables through sticking to defined grammar for assignment.

27. **<arithmetic_expression>::=<addition_exp> | <subtraction_exp> | <multiplication_exp> | <division_exp>**

This non-terminal shows the arithmetic expressions in Bakusa language. Users can use these expressions for mathematical calculations.

28. **<relational_expression>::=<equivalence_check_exp> | <not_equal_check_exp> | <less_than_exp> | <greater_than_exp> | <equal_and_less_than_exp> | <equal_and_greater_than_exp>**

This non-terminal lists the relational expressions in our language. Users have a chance to choose the related expression for their code. These expressions have a determined grammar.

29. **<logical_expression>::= <and_exp> | <or_exp>**

This non-terminal shows the logical expressions in Bakusa language. Also, it shows the usage of logical expressions.

30. **<while_stmt> ::= while<LP><logical_expression> <RP> <LB> <statements> <RB>**
31. **<for_stmt>::= for<LP><initialization><semicolon><logical_expression><RP><LB><statements><RB>**
32. **<do_while_stmt> ::= do<LB> <statements> <RB> while <LP> <logical_expression><RP>**

These non-terminals (30, 31, 32) are created to show the looping syntax in Bakusa language. In Bakusa language there are three loop options which are while, for and do-while.

33. **<conditional_stmts> ::= <if_stmt> | <if_stmt><else_stmt> | <if_stmt><else_it_stmt_ctn> | <if_stmt><else_it_stmt_ctn><else_stmt>**

This non-terminal shows the possible “if - else” statements and their syntax.

34. **<single_line_comment> ::= <sl_comment_sign><sentence>**

This non-terminal was created to show how to add comments in Bakusa language.

35. **<input_stmt>::=bksin<LP><input><RP>**

This non-terminal indicates the input statement of Bakusa language. It is used to accept the input from the standard input device.

36. **<output_stmt>::=bksout<LP><output><RP>**

This non-terminal indicates the output statement of Bakusa language. It is used to display output to the standard output device.

37. **<function_definition>::= <returned_function> | <void_function> | <paraVoid_function> | <paraReturn_function>**

This non-terminal lists the function types in Bakusa language. Also these non-terminals define the syntax for function definitions.

38. **<function_call>::=<identifier><LP><RP><semicolon>**

This non-terminal indicates the way of calling a function in the Bakusa language.

Non-Trivial Tokens of Bakusa Language

- **bksin:** This is the token that user can take input with using this in Bakusa Language
- **bksout:** This is the token that gives output to user in Bakusa Language
- **if:** This token is for if statements in Bakusa Language
- **else if:** This token is for else if statements in Bakusa language
- **else:** This token is for else statements in Bakusa language
- **for:** This token is for “for” statements in Bakusa language
- **do:** This token is for do-while statements in Bakusa language
- **while:** This token is for while statements in Bakusa language
- **integer:** This token is for integer type data in Bakusa language
- **double:** This token is for double type data in Bakusa language
- **signedInteger:** This token is for signed integer type data in Bakusa language
- **signedDouble:** This token is for signed double type data in Bakusa language
- **boolean:** This token is for boolean type data in Bakusa language
- **char:** This token is for char type data in Bakusa language
- **string:** This token is for string type data in Bakusa language
- **return:** This token is for return statements in Bakusa language. Return is used in non-void functions.
- **begin:** This token is for indicating the start of the main program.

Evaluation

a) Readability

In order to improve readability in Bakusa language, K&R Indentation Style is used. Variables' and functions' names are meaningful. It is simple to read.

b) Writablity

In order to improve writability Bakusa is using most common loop, if-else structures. Their syntax is the same with Java. Input and output statements are inspired from C which is easier than Java. In Bakusa input statement is "bksin" which is similar to "cin" and output statement is "bksout" which is also similar to "cout". Moreover, initialization and declaration have the same structure as Java.

c) Reliability

In the tests that have been performed the Bakusa language always gives the right outputs which demonstrates the Bakusa language is reliable.