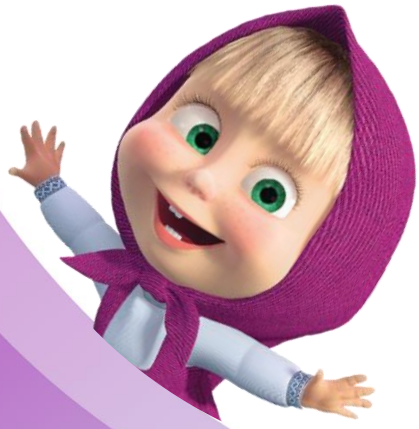
Computer Science Department



CPCS 302 -COMPILER CONSTRUCTION

MASHA LANGUAGE COMPILER

GROUP1- IAR:

Mona Hafez.

Bodor Alamri

Razan Aljuhani

Shuroog Alshaikh

Table of Contents

Member	Tasks Performed
Bodor Alamri	50% of Phase 1 + Video + 25% of Report
Mona Hafez	Phase 3 + 25% of Report
Razan Aljuhani	Phase 2 + 25% of Report
Shuroog Alshaikh	50% of Phase 1+ 25% of Report

1. PHASE1

1.1 INTRODUCTION

MASHA is a simple language with easy-to-understand codes that make learning programming more enjoyable for children, allowing programming to spread across Saudi Arabia in the next decade.

Source Sample:

```
:: Adding the first 10th number starting from 0

:: V1num is the summation

var V1num =0 .

:: V1counter is a counter

var V1counter =0 .

Until ( V1counter =? 10 ):

:: Summation result store in V1num

 [V1num + V1counter. =2

:: Update the counter

V1counter +1 .]
```

1.2 TOKENS

MASHA Token is a simple and easy-to-remember, the following is a regular expression for basic tokens of MASHA:

Table 1: Tokens Table

Token	Regular Expression
Arithmetic Operations	Addition: “+” Subtraction: “-” Multiplication: “*” Division: “/” Assignment: “=”
Comparison Operations	Equality: “=?” Inequality: “~=?” Less Than: “<?” Less Than or Equal: “<=?” Greater Than: “>?” Greater Than or Equal: “>=?”
Logical Operations	Logical AND: “&” Logical OR: “OR” Logical XOR: “XOR” Logical NOT: “!”
Constant Identifiers	“C” [“0”-“9”]+ [“A” - “Z” | “a”- “z”]+
Variable Identifiers	“V” [“0”-“9”]+ [“A” - “Z” | “a”- “z”]+
Static Identifiers	“S” [“0”-“9”]+ [“A” - “Z” | “a”- “z”]+
Integer Number	[“0”-“9”]+
Float Number	[“0”-“9”]+. [“0”-“9”]+
String	“#”[“A” - “Z” | “a”- “z”]+ [“0”-“9”]*
White Spaces	(“ ”)+ | (“\t”)+ |(“\n”)+
Keywords	[“if” | “else” | “Fun” | “Until” |"list"]
Single Line Comments	“::”
List	(Intger|String|float) **+ ,** (Intger|String|float)
Punctuation Marks	Left Parentheses: “(” Right Parentheses: “)” Left Square Brackets: “[” Right Square Brackets: “]” Double Quotes: “ " ” Comma: “,” Full Stop: “.” Colon: “:”

The language has three datatypes and one data structure:

- **Integer**
- **Float**
- **String**

- **List**

Conditional statement form :

- **if (Comparison_Stmt) : [Stmt+] .**
- **if (Comparison_Stmt) : [Stmt+] else [Stmt+]**

LOOP statement forms :

- **Fun(Integer : Integer : Integer) : [Stmt+]**
- **Until ((Comparison_Stmt | Logical_Stmt)) : [Stmt+]**

1.3 EXAMPLES

Examples of MASHA language statements that clarify the language's structure.

Table 2: Statements Table

Statement Type	Example (Code)
Arithmetic statements	• C1num + C2num . • 6 * 9.0 .
Comparison statements	• V1num <? V2num • 10 ~=? 50.0
Logical statements	(V1num <? V2num) & (10 ~=? 50.0)
Conditional statements	• if (V1num <? V2num): [V1num+1 .]. • if (V1num <? V2num): [V1num+1.] else [V2num+1.]
Iterative statement	• Fun (0: 1 :5): [var V1num= V2num.] • Until ((V1num <? 5) & (V1num~=? 1)): [var V1num= 5 .]

Variable Declaration	String datatype:	var V1name = #ALI .
	Integer datatype:	var V1count = 10 .
	Float datatype:	var V1count = 10.9 .
Constant variable declaration	var C3name = #Masha .	
Static variable declaration	var S8i =0 .	
Data structure: List	list V1Addresses=[#Jeddah,#Makkah] .	

Figure 2: Run examples in jj File

Figure 1: Run examples in jjt File

2. PHASE 2

2.1 BNF

1. **Start** → **Stmt**
2. **Stmt** → **Declaration| Logical_ Stmt| Conditional_Stmt| Loop| List**
3. **Declaration** → **(Declaration_Var| Arithmetic) .**
4. **Declaration_Var** → **var ID = (ID| num| String)**
5. **ID** → **Const| Static| Variable**
6. **num** → **Integer| Float**
7. **Static** → **S[0-9]+ [A – Z| a - z]+**
8. **Const** → **C[0-9]+ [A – Z| a- z]+**
9. **Comparison_Stmt** → **(ID| num) Comparison_op (ID| num)**

10. **Comparison_op** → **(= | ~= | < | <= | > | >=) ?**

11. **Logical_Stmt** → **Binary_Logical_Stmt| Unary_Logical_Stmt**

12. **Binary_Logical_Stmt → (Comparison_Stmt) (&| OR| XOR) (Comparison_Stmt)**

13. **Unary_Logical_Stmt → ! (Comparison_Stmt)**

14. **Conditional_ Stmt → if (Comparison_Stmt) : [Stmt+] . | if (Comparison_Stmt) : [Stmt+] else [Stmt+]**

15. **Arithmetic → Term Arithmetic_op Term**

16. **Arithmetic_ op → (+| -| *| /| =)**

17. **Term → ID| num**

18. **Integer → [0-9]+**

19. **Float → [0-9]+.[0-9]+**

20. **String → #[A - Z| a- z]+ [0-9]***

21. **Loop → Function| Until**

22. **Function → Fun(Integer : Integer : Integer) : [Stmt+.]**

23. **Until → Until ((Comparison_Stmt | Logical_Stmt)) : [Stmt+.]**

24. **List → list ID =[((String| num) (,))+ (String| num)].**

25. **Variable → V[0-9]+ [A – Z | a - z]+**

2.2 Explanation Grammar

Table 3: Explanation Grammar Table

#	Nonterminal	Production	Explanation
1	**Start**	**→ Stmt**	Should write a Stmt to start programming.
2	**Stmt**	**→ Declaration| Logical_ Stmt| Conditional_Stmt| Loop| List**	A Stmt can be one of the 5 types.
3	**Declaration**	**→ (Declaration_Var| Arithmetic) .**	A declaration may be for variable or Arithmetic, and must end with full stop **(.)** .

4	**Declaration_Var**	→ **var ID = (ID| num| String)**	The declaration of variable should start with var keyword followed by ID then assignment (=) followed by ID or num or String. ex. var S8i = 0.
5	**ID**	→ **Const| Static| Variable**	The ID is either Const or Static or variable.
6	**num**	→ **Integer| Float**	The num is either Integer or float
7	**Static**	→ **S[0-9]+ [A – Z| a - z]+**	The Static ID should start with S followed by one or more digits then end with one or more small and Capital letters. ex. S8i
8	**Const**	→ **C[0-9]+ [A – Z| a- z]+**	The Const ID should start with C followed by one or more digits then end with one or more small and Capital letters. ex. C2Masha
9	**Comparison_Stmt**	→ **(ID| num) Comparison_op (ID| num)**	The Comparison_Stmt should start with ID or num followed by Comparison_op then end with ID or num. ex. 8=?8
10	**Comparison_op**	→ **(= | ~= | < | <= | > | >=) ?**	Comparison_op can be one of those operators =?, ~=? , <?, <=? , >?, >=? .
11	**Logical_Stmt**	→ **Binary_Logical_Stmt| Unary_Logical_Stmt**	The Logical_Stmt can be either Binary or Unary.
12	**Binary_Logical_Stmt**	→ **(Comparison_Stmt) (&| OR| XOR) (Comparison_Stmt)**	Binary expression ex. (7>?3) & (3<?7)
13	**Unary_Logical_Stmt**	→ **!(Comparison_Stmt)**	Unary expression ex. !(7>?3)

14	**Conditional_ Stmt**	**→ if (Comparison_Stmt) : [Stmt+] . | if (Comparison_Stmt) : [Stmt+] else [Stmt+]**	The Conditional_ Stmt can be either just if statement that end with full stop (.) or if else statements. ex.1. if (V1num <? V2num): [V1num+1 .] ex.2. if (V1num <? V2num): [V1num+1] else [V2num+1]
15	**Arithmetic**	**→ Term Arithmetic_op Term**	This is Arithmetic statement. ex. 5*5
16	**Arithmetic_ op**	**→ (+| -| *| /| =)**	There are five Arithmetic operation that can be used +, -, *, /, =.
17	**Term**	**→ ID| num**	The Term can be either ID or num
18	**Integer**	**→ [0-9]+**	Any Integer ex. 100
19	**Float**	**→ [0-9]+.[0-9]+**	Any Float ex. 99.7
20	**String**	**→ #[A - Z| a- z]+ [0-9]***	Any String should start with #. ex. #Masha1
21	**Loop**	**→ Function| Until**	The Loop can be either Function or Until.
22	**Function**	**→ Fun(Integer : Integer : Integer) : [Stmt+.]**	Function is loop statement, it should be start with Fun. ex. Fun (0: 1 :5): [var V1num = V2num .]
23	**Until**	**→ Until ((Comparison_Stmt | Logical_Stmt)) : [Stmt+.]**	Until is loop statement , it should be start with Until. ex. Until ((V1num <? 5) & (V1num~=? 1)): [var V1num= 5 .]
24	**List**	**→ list ID =[((String| num) (,))+ (String| num)].**	List is data structure, it should be start with list and end with full stop(.) ex. list V1Addresses = [#Jeddah ,#Makkah].
25	**Variable**	**V[0-9]+ [A – Z | a - z]+**	The Variable ID should start with V followed by one or more digits then end with one or more small and Capital letters. ex. V2Masha

2.3 Screenshots of jj grammar run

- **Arithmetic statements**

```
Welcome to Masha language
--------------------------------
Please enter an expression:C1num + C2num .
You declare a constant varible.
You declare a constant varible.
You used an artimatic statment!

Good Job! Your statment is Correct :) .


Welcome to Masha language
--------------------------------
Please enter an expression:6 * 9.0 .
You used an artimatic statment!

Good Job! Your statment is Correct :) .
```

Figure 3: Screenshot (Arithmetic statements)

- **Comparison statements**

```
Welcome to Masha language
--------------------------------
Please enter an expression:V1num <? V2num
You declare a varible.
You declare a varible.
You used a comparison statment!

Good Job! Your statment is Correct :) .


Welcome to Masha language
--------------------------------
Please enter an expression:10 ~=? 50.0
You used a comparison statment!

Good Job! Your statment is Correct :) .
```

Figure 4: Screenshot (Comparison statements)

- **Logical statements**

```
Welcome to Masha language
--------------------------------
Please enter an expression:(V1num <? V2num) & (10 ~=? 50.0)
You declare a varible.
You declare a varible.
You used a comparison statment!
You used a comparison statment!
You used a logical statment!

Good Job! Your statment is Correct :) .
```

Figure 5: Screenshot (Logical statements)

- ## Conditional statements

```
Welcome to Masha language
--------------------------------
Please enter an expression:if (V1num <? V2num): [V1num+1 .].
You declare a varible.
You declare a varible.
You used a comparison statment!
You declare a varible.
You used an artimatic statment!
You used a conditional statment!

Good Job! Your statment is Correct :) .


Welcome to Masha language
--------------------------------
Please enter an expression:if (V1num <? V2num): [V1num+1.] else [V2num+1.]
You declare a varible.
You declare a varible.
You used a comparison statment!
You declare a varible.
You used an artimatic statment!
You declare a varible.
You used an artimatic statment!
You used a conditional statment!

Good Job! Your statment is Correct :) .
```

Figure 6: Screenshot (Conditional statements)

- ## Iterative statement

```
Welcome to Masha language
--------------------------------
Please enter an expression:Fun (0: 1 :5): [var V1num= V2num.]
You declare a varible.
You declare a varible.
You used a function!
You used an Iterative statement!

Good Job! Your statment is Correct :) .


Welcome to Masha language
--------------------------------
Please enter an expression:Until ((V1num <? 5) & (V1num~=? 1)): [var V1num= 5 .]
You declare a varible.
You used a comparison statment!
You declare a varible.
You used a comparison statment!
You used a logical statment!
You declare a varible.
You used an Iterative statement!

Good Job! Your statment is Correct :) .
```

Figure 7: Screenshot (Iterative statements)

- Variable Declaration

```
Welcome to Masha language
-------------------------------
Please enter an expression:var V1name = #ALI .
You declare a varible.

Good Job! Your statment is Correct :) .


Welcome to Masha language
-------------------------------
Please enter an expression:var V1count = 10 .
You declare a varible.

Good Job! Your statment is Correct :) .


Welcome to Masha language
-------------------------------
Please enter an expression:var V1count = 10.9 .
You declare a varible.

Good Job! Your statment is Correct :) .
```

Figure 8: Screenshot (Variable Declaration)

- Constant variable declaration

```
masha (4) [Java Application] /Users/bodoralharbi/.p2/pool/plugins/o
Welcome to Masha language
--------------------------------
Please enter an expression:var C3name = #Masha .
You declare a constant varible.

Good Job! Your statment is Correct :) .
```

Figure 9: Screenshot (Constant Variable Declaration)

- Static variable declaration

```
Welcome to Masha language
-------------------------------
Please enter an expression:var S8i =0 .
You declare a static varible.

Good Job! Your statment is Correct :) .
```

Figure 10: Screenshot (Static Variable Declaration)

- Data structure: List

```
Welcome to Masha language
-------------------------------
Please enter an expression:list V1Addresses=[#Jeddah,#Makkah] .
You declare a varible.
You used a list!

Good Job! Your statment is Correct :) .
```

Figure 11: Screenshot (List)

3. PHASE 3

Screenshots of jjt grammar run

- **Arithmetic statements**

```
masha (5) [Java Application] /Users/bodoramarbi/.p2/pool/plugins/org.ecl
   Welcome to Masha language
-------------------------------

Please enter an expression:C1num + C2num .
You declare a constant varible.
You declare a constant varible.
You used an artimatic statment!
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  decleration
Masha Tree >>   arithmetic_stmt
Masha Tree >>    ID
Masha Tree >>     CONSTANT_ID
Masha Tree >>    artimatic_operation
Masha Tree >>     PLUS
Masha Tree >>    ID
Masha Tree >>     CONSTANT_ID
Masha Tree >>   FULL_STOP
Good Job! Your statment is Correct :) .



Please enter an expression:6 * 9.0 .
You used an artimatic statment!
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  decleration
Masha Tree >>   arithmetic_stmt
Masha Tree >>    number
Masha Tree >>     INT
Masha Tree >>    artimatic_operation
Masha Tree >>     MULTIPLY
Masha Tree >>    number
Masha Tree >>     FLOAT
Masha Tree >>   FULL_STOP
Good Job! Your statment is Correct :) .
```

Figure 12: Screenshot (Arithmetic statements)

- **Comparison statements**

```
masha (5) [Java Application] /Users/bodoralharbi/.p2/pool/p
   Welcome to Masha language
--------------------------------

Please enter an expression:V1num <? V2num
You declare a varible.
You declare a varible.
You used a comparison statment!
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  comparison_stmet
Masha Tree >>   ID
Masha Tree >>    VARIABLE_ID
Masha Tree >>   comparison_operation
Masha Tree >>    LESS_THAN
Masha Tree >>   ID
Masha Tree >>    VARIABLE_ID
Good Job! Your statment is Correct :) .



Please enter an expression:10 ~=? 50.0
You used a comparison statment!
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  comparison_stmet
Masha Tree >>   number
Masha Tree >>    INT
Masha Tree >>   comparison_operation
Masha Tree >>    INEQUAL
Masha Tree >>   number
Masha Tree >>    FLOAT
Good Job! Your statment is Correct :) .
```

Figure 13: Screenshot (Comparison statements)

- **Logical statements**

```
masha (5) [Java Application] /Users/bodoralharbi/.p2/pool/plugins/org.eclipse.justj.openjdk
   Welcome to Masha language
--------------------------------

Please enter an expression:(V1num <? V2num) & (10 ~=? 50.0)
You declare a varible.
You declare a varible.
You used a comparison statment!
You used a comparison statment!
You used a logical statment!
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  logical_stmt
Masha Tree >>   Binary_logical_stmt
Masha Tree >>    LEFT_PAR
Masha Tree >>    comparison_stmet
Masha Tree >>     ID
Masha Tree >>      VARIABLE_ID
Masha Tree >>     comparison_operation
Masha Tree >>      LESS_THAN
Masha Tree >>     ID
Masha Tree >>      VARIABLE_ID
Masha Tree >>    RIGHT_PAR
Masha Tree >>    logical_operation
Masha Tree >>     AND
Masha Tree >>    LEFT_PAR
Masha Tree >>    comparison_stmet
Masha Tree >>     number
Masha Tree >>      INT
Masha Tree >>     comparison_operation
Masha Tree >>      INEQUAL
Masha Tree >>     number
Masha Tree >>      FLOAT
Masha Tree >>    RIGHT_PAR
Good Job! Your statment is Correct :) .
```

Figure 14: Screenshot (Logical statements)

- **Conditional statements**

```
Please enter an expression:if (V1num <? V2num): [V1num+1.] else [V2num+1.]
You declare a varible.
You declare a varible.
You used a comparison statment!
You declare a varible.
You used an artimatic statment!
You declare a varible.
You used an artimatic statment!
You used a conditional statment!
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  conditional_stmt
Masha Tree >>   IF
Masha Tree >>   LEFT_PAR
Masha Tree >>   comparison_stmet
Masha Tree >>    ID
Masha Tree >>     VARIABLE_ID
Masha Tree >>    comparison_operation
Masha Tree >>     LESS_THAN
Masha Tree >>    ID
Masha Tree >>     VARIABLE_ID
Masha Tree >>   RIGHT_PAR
Masha Tree >>   COLON
Masha Tree >>   LEFT_SQR_BRACKT
Masha Tree >>   statement
Masha Tree >>    decleration
Masha Tree >>     arithmetic_stmt
Masha Tree >>      ID
Masha Tree >>       VARIABLE_ID
Masha Tree >>      artimatic_operation
Masha Tree >>       PLUS
Masha Tree >>      number
Masha Tree >>       INT
Masha Tree >>     FULL_STOP
Masha Tree >>   RIGHT_SQR_BRACKT
Masha Tree >>   else_stmt
Masha Tree >>    ELSE
Masha Tree >>    LEFT_SQR_BRACKT
Masha Tree >>    statement
Masha Tree >>     decleration
Masha Tree >>      arithmetic_stmt
Masha Tree >>       ID
Masha Tree >>        VARIABLE_ID
Masha Tree >>       artimatic_operation
Masha Tree >>        PLUS
Masha Tree >>       number
Masha Tree >>        INT
Masha Tree >>      FULL_STOP
Masha Tree >>    RIGHT_SQR_BRACKT
Good Job! Your statment is Correct :) .
```

Figure 15: Screenshot (Conditional statements)

- **Iterative statement**

```
Please enter an expression:Until ((V1num <? 5) & (V1num~=? 1)): [var V1num= 5 .]
You declare a varible.
You used a comparison statment!
You declare a varible.
You used a comparison statment!
You used a logical statment!
You declare a varible.
You used an Iterative statement!
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  loop
Masha Tree >>   until
Masha Tree >>    until
Masha Tree >>    LEFT_PAR
Masha Tree >>    logical_stmt
Masha Tree >>     Binary_logical_stmt
Masha Tree >>      LEFT_PAR
Masha Tree >>      comparison_stmet
Masha Tree >>       ID
Masha Tree >>        VARIABLE_ID
Masha Tree >>       comparison_operation
Masha Tree >>        LESS_THAN
Masha Tree >>       number
Masha Tree >>        INT
Masha Tree >>      RIGHT_PAR
Masha Tree >>      logical_operation
Masha Tree >>       AND
Masha Tree >>      LEFT_PAR
Masha Tree >>      comparison_stmet
Masha Tree >>       ID
Masha Tree >>        VARIABLE_ID
Masha Tree >>       comparison_operation
Masha Tree >>        INEQUAL
Masha Tree >>       number
Masha Tree >>        INT
Masha Tree >>      RIGHT_PAR
Masha Tree >>    RIGHT_PAR
Masha Tree >>    COLON
Masha Tree >>    LEFT_SQR_BRACKT
Masha Tree >>    statement
Masha Tree >>     decleration
Masha Tree >>      declere_VAR
Masha Tree >>       VAR
Masha Tree >>       ID
Masha Tree >>        VARIABLE_ID
Masha Tree >>       ASSIGNMENT
Masha Tree >>       number
Masha Tree >>        INT
Masha Tree >>      FULL_STOP
Masha Tree >>    RIGHT_SQR_BRACKT
Good Job! Your statment is Correct :) .
```

Figure 16: Screenshot ("Until " Iterative statement)

```
   Welcome to Masha language
-------------------------------

Please enter an expression:Fun (0: 1 :5): [var V1num= V2num.]
You declare a varible.
You declare a varible.
You used a function!
You used an Iterative statement!
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  loop
Masha Tree >>   fun
Masha Tree >>    fun
Masha Tree >>    LEFT_PAR
Masha Tree >>    INT
Masha Tree >>    COLON
Masha Tree >>    INT
Masha Tree >>    COLON
Masha Tree >>    INT
Masha Tree >>    RIGHT_PAR
Masha Tree >>    COLON
Masha Tree >>    LEFT_SQR_BRACKT
Masha Tree >>    statement
Masha Tree >>     decleration
Masha Tree >>      declere_VAR
Masha Tree >>       VAR
Masha Tree >>       ID
Masha Tree >>        VARIABLE_ID
Masha Tree >>       ASSIGNMENT
Masha Tree >>       ID
Masha Tree >>        VARIABLE_ID
Masha Tree >>      FULL_STOP
Masha Tree >>    RIGHT_SQR_BRACKT
Good Job! Your statment is Correct :) .
```

Figure 17: Screenshot ("Fun "Iterative statement)

- **Variable Declaration**

```
Please enter an expression:var V1name = #ALI .
You declare a varible.
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  decleration
Masha Tree >>   declere_VAR
Masha Tree >>    VAR
Masha Tree >>    ID
Masha Tree >>     VARIABLE_ID
Masha Tree >>    ASSIGNMENT
Masha Tree >>    STRING
Masha Tree >>   FULL_STOP
Good Job! Your statment is Correct :) .


Please enter an expression:var V1count = 10 .
You declare a varible.
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  decleration
Masha Tree >>   declere_VAR
Masha Tree >>    VAR
Masha Tree >>    ID
Masha Tree >>     VARIABLE_ID
Masha Tree >>    ASSIGNMENT
Masha Tree >>    number
Masha Tree >>     INT
Masha Tree >>   FULL_STOP
Good Job! Your statment is Correct :) .


Please enter an expression:var V1count = 10.9 .
You declare a varible.
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  decleration
Masha Tree >>   declere_VAR
Masha Tree >>    VAR
Masha Tree >>    ID
Masha Tree >>     VARIABLE_ID
Masha Tree >>    ASSIGNMENT
Masha Tree >>    number
Masha Tree >>     FLOAT
Masha Tree >>   FULL_STOP
Good Job! Your statment is Correct :) .
```

Figure 18: Screenshot (Variable Declaration)

- **Constant variable declaration**

```
masha (5) [Java Application] /Users/bodoralharbi/.p2/pool/plugins/c
   Welcome to Masha language
--------------------------------

Please enter an expression:var C3name = #Masha .
You declare a constant varible.
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  decleration
Masha Tree >>   declere_VAR
Masha Tree >>    VAR
Masha Tree >>    ID
Masha Tree >>     CONSTANT_ID
Masha Tree >>    ASSIGNMENT
Masha Tree >>    STRING
Masha Tree >>   FULL_STOP
Good Job! Your statment is Correct :) .
```

Figure 19: Screenshot (Constant Variable Declaration)

- **Static variable declaration**

```
masha (5) [Java Application] /Users/bodoralharbi/.p2/poo
   Welcome to Masha language
--------------------------------

Please enter an expression:var S8i =0 .
You declare a static varible.
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  decleration
Masha Tree >>   declere_VAR
Masha Tree >>    VAR
Masha Tree >>    ID
Masha Tree >>     STATIC_ID
Masha Tree >>    ASSIGNMENT
Masha Tree >>    number
Masha Tree >>     INT
Masha Tree >>   FULL_STOP
Good Job! Your statment is Correct :) .
```

Figure 20: Screenshot (Static Variable Declaration)

- **Data structure: List**

```
   Welcome to Masha language
--------------------------------

Please enter an expression:list V1Addresses=[#Jeddah,#Makkah] .
You declare a varible.
You used a list!
Masha Tree >>Start
Masha Tree >> statement
Masha Tree >>  list
Masha Tree >>   list
Masha Tree >>   ID
Masha Tree >>    VARIABLE_ID
Masha Tree >>   ASSIGNMENT
Masha Tree >>   LEFT_SQR_BRACKT
Masha Tree >>   LIST_ELEMENTS
Masha Tree >>   RIGHT_SQR_BRACKT
Masha Tree >>   FULL_STOP
Good Job! Your statment is Correct :) .
```

Figure 21: Screenshot (List)

4. Appendix

4.1 jj Code

```
/**
 * JavaCC template file created by SF JavaCC plugin 1.5.28+ wizard for
JavaCC 1.5.0+
 */
/*GROUP1- IAR:
Mona Hafez.
Bodor Alamri.
Razan Aljuhani.
Shuroog Alshaikh.
 */
options
{
  static = true;
}

PARSER_BEGIN(masha)
package MASHA_Language;

public class masha
{
  public static void main(String args []) throws ParseException
  {
```

```
    masha parser = new masha(System.in);
    while (true)
    {
      System.out.println("Welcome to Masha language");
      System.out.print("-------------------------------\n");
      System.out.print("Please enter an expression:");
      try
      {
        masha.start();
        System.out.println();
        System.out.println("Good Job! Your statment is Correct :)
.\n\n");
      }
      catch (Exception e)
      {
        System.out.println();
        System.out.println("Oops.Wrong statement, Try again :( .\n\n");
        System.out.println(e.getMessage());
        masha.ReInit(System.in);
      }
      catch (Error e)
      {
        System.out.println("Oops. Not accepted statement, Try again :(
.\n\n");
        break;
      }
    }
  }
}

PARSER_END(masha)

/* White Spaces */
SKIP :
{
  " "
| "\r"
| "\t"
| "\n"
}

TOKEN : /* Identifiers and data type  */
{
  < STATIC_ID :
    "S" (< DIGIT >)+
    (
      (< SMALL_LETTER >)*
    | (< CAPITAL_LETTER >)*
    )+ >
  {
    System.out.println("You declare a static varible.");
  }
| < VARIABLE_ID :
```

```
      "V" (< DIGIT >)+
      (
        (< SMALL_LETTER >)*
      | (< CAPITAL_LETTER >)*
      )+ >
    {
      System.out.println("You declare a varible.");
    }
  | < CONSTANT_ID :
      "C" (< DIGIT >)+
      (
        (< SMALL_LETTER >)*
      | (< CAPITAL_LETTER >)*
      )+ >
    {
      System.out.println("You declare a constant varible.");
    }
  | < INT : (< DIGIT >)+ >
  | < FLOAT : (< DIGIT >)+ (".") (< DIGIT >)+ >
  | < STRING :
      "#"
      (
        (< SMALL_LETTER >)
      | (< CAPITAL_LETTER >)
      )+
      (< DIGIT >)* >
    //represent the list elements.
  | < LIST_ELEMENTS :
      (
        (
          (
            < INT >
          | < FLOAT >
          | < STRING >
          )
          (",")
        )+
      )
      (
        < INT >
      | < FLOAT >
      | < STRING >
      ) >
  | < #CAPITAL_LETTER : [ "A"-"Z" ] >
  | < #DIGIT : [ "0"-"9" ] >
  | < #SMALL_LETTER : [ "a"-"z" ] >
  }

  TOKEN : /* Keywords */
  {
    < IF : "if" > // (if) and (else) used with conditional statements.
  | < ELSE : "else" >
  | < FUN : "Fun" > // To define the function (method)
```

```
| < UNTIL : "Until" > // used to
| < VAR : "var" >
| < LIST : "list" >
}

TOKEN : /* Arithmetic Operations */
{
  < PLUS : "+" >
| < MINUS : "-" >
| < MULTIPLY : "*" >
| < DIVIDE : "/" >
| < ASSIGNMENT : "=" >
}

TOKEN : /* Relational Operations */
{
  < EQUAL : "=?" >
| < INEQUAL : "~=?" >
| < LESS_THAN : "<?" >
| < LESS_OR_EQUAL : "<=?" >
| < GREAT_THAN : ">?" >
| < GREAT_OR_EQUAL : ">=?" >
}

TOKEN : /* Logical Operators */
{
  < AND : "&" >
| < OR : "OR" >
| < XOR : "XOR" >
| < NOT : "!" >
}

TOKEN : /* Punctuation Marks */
{
  < LEFT_PAR : "(" > //Left Parentheses
| < RIGHT_PAR : ")" > //Right Parentheses
| < LEFT_SQR_BRACKT : "[" > // Left Square Brackets
| < RIGHT_SQR_BRACKT : "]" > //Right Square Brackets
| < DOUBLE_QUOTES : " \" " > //Double Quotes
| < COMMA : "," >
| < FULL_STOP : "." >
| < COLON : ":" >
}

// Discard the comments
SPECIAL_TOKEN :
{
  < SINGLE_LINE_COMMENT :
    "::" (~[ "\n", "\r" ])*
    (
      "\n"
    | "\r"
    | "\r\n"
```

```
    ) >
  {
    System.out.println("Your write a comment, it is discarded!");
  }
}

void start() :
{}
{
  statement()
}

void statement() :
{}
{
  (
    LOOKAHEAD(2) // look to the second position to identify the
operation.
    decleration()
  | logical_stmt()
  | conditional_stmt()
  | comparison_stmet()
  | loop()
  )
| list()
}

void decleration() :
{}
{
  // the statement should end with full stop (.)
  (
    LOOKAHEAD(2)
    declere_VAR()
  | arithmetic_stmt()
  )
  < FULL_STOP >
}

// used to write an arithmetic expression.
void arithmetic_stmt() :
{}
{
  // the structure could be: var C0num + 5 OR  6*6
  (
    (
      ID()
    | number()
    )
    artimatic_operation()
    (
      ID()
    | number()
```

```
    )
  )+
  {
    System.out.println("You used an artimatic statment!");
  }
}

// this method used to declare a variable
void declere_VAR() :
{}
{
  // the structure could be: var C0num = C1num  OR  var C0num = 5
  < VAR > ID() < ASSIGNMENT >
  (
    ID()
  | number()
  | < STRING >
  )
}

// work
void logical_stmt() :
{}
{
  (
    Binary_logical_stmt()
  | unary_logical_stmt()
  )
  // the structure could be: (V1num <? V2num) & (10 ~=? 50.0)
  {
    System.out.println("You used a logical statment!");
  }
}

void unary_logical_stmt() :
{}
{
  < NOT > < LEFT_PAR > comparison_stmet() < RIGHT_PAR >
}

void Binary_logical_stmt() :
{}
{
  < LEFT_PAR > comparison_stmet() < RIGHT_PAR > logical_operation() <
LEFT_PAR > comparison_stmet() < RIGHT_PAR >
}

void conditional_stmt() :
{}
{
  // the structure is: if (V1num <? V2num): [V1num+1.]  else [
V2num+1.]
  // OR: if (V1num <? V2num): [V1num+1.] .
```

```
  < IF > < LEFT_PAR >
  (
    comparison_stmet()
  )
  < RIGHT_PAR > < COLON >
  < LEFT_SQR_BRACKT >
  (
    statement()
  )+
  < RIGHT_SQR_BRACKT >
  (
    else_stmt()
  | < FULL_STOP >
  )
  {
    System.out.println("You used a conditional statment!");
  }
}

void else_stmt() :
{}
{
  < ELSE > < LEFT_SQR_BRACKT >
  (
    statement()
  )+
  < RIGHT_SQR_BRACKT >
}

// work
void comparison_stmet() :
{}
{
  // the structure could be: V1num <? V2num OR 10 ~=? 50.0
  (
    ID()
  | number()
  )
  comparison_operation()
  (
    ID()
  | number()
  )
  {
    System.out.println("You used a comparison statment!");
  }
}

// used to create a loop
void loop() :
{}
{
  (
```

```
    until()
  | fun()
  )
  {
    System.out.println("You used an Iterative statement!");
  }
}

void until() :
{}
{
  /* the structure is:
Until ((V1num <? 5) & (V1num~=? -1)):
[V1num+1.]
 */
  < UNTIL > < LEFT_PAR >
  (
    comparison_stmet()
  | logical_stmt()
  )
  < RIGHT_PAR > < COLON >
  < LEFT_SQR_BRACKT >
  (
    statement()
  )+
  < RIGHT_SQR_BRACKT >
}

void fun() :
{}
{
  /* the structure:
Fun (0: 1 :5):
[V1num+1.]
 */
  < FUN > < LEFT_PAR > < INT > < COLON > < INT > < COLON > < INT > <
RIGHT_PAR > < COLON >
  < LEFT_SQR_BRACKT >
  (
    statement()
  )+
  < RIGHT_SQR_BRACKT >
  {
    System.out.println("You used a function!");
  }
}

void list() :
{}
{
  //| < LIST_ELEMENTS :  >
  /* the structure:
list V1Addresses =[#Jeddah,#Makkah]. */
```

```
  (
    < LIST > ID() < ASSIGNMENT > < LEFT_SQR_BRACKT >
    // the elemnts in the list
    < LIST_ELEMENTS >
    // end of the list
    < RIGHT_SQR_BRACKT > < FULL_STOP >
  )
  {
    System.out.println("You used a list!");
  }
}

void ID() :
{}
{
  (
    < STATIC_ID >
  | < VARIABLE_ID >
  | < CONSTANT_ID >
  )
}

void number() :
{}
{
  (
    < INT >
  | < FLOAT >
  )
}

void artimatic_operation() :
{}
{
  (
    < PLUS >
  | < MINUS >
  | < MULTIPLY >
  | < DIVIDE >
  )
}

void logical_operation() :
{}
{
  (
    < AND >
  | < OR >
  | < XOR >
  | < NOT >
  )
}
```

```
void comparison_operation() :
{}
{
  (
    < EQUAL >
  | < INEQUAL >
  | < LESS_THAN >
  | < LESS_OR_EQUAL >
  | < GREAT_THAN >
  | < GREAT_OR_EQUAL >
  )
}
```

4.2 jjt Code

- **jj file**

```
/*@bgen(jjtree) Generated By:JJTree: Do not edit this line. masha.jj */
/*@egen*//**
 * JJTree template file created by SF JavaCC plugin 1.5.28+ wizard for
JavaCC 1.5.0+
 */
/* GROUP1- IAR:
Mona Hafez.
Bodor Alamri.
Razan Aljuhani.
Shuroog Alshaikh.
 */
options
{
  static = true;
}

PARSER_BEGIN(masha)
package MASHA_Language_Tree;

public class masha/*@bgen(jjtree)*/implements
mashaTreeConstants/*@egen*/
{/*@bgen(jjtree)*/
  protected static JJTmashaState jjtree = new JJTmashaState();

/*@egen*/
  public static void main(String args [])
  {
    System.out.println("   Welcome to Masha language");
    System.out.print("------------------------------\n");
    masha parser = new masha(System.in);
    while (true)
    {
```

```
      System.out.print("\nPlease enter an expression:");
      try
      {
        SimpleNode n = masha.Start();
        n.dump("Masha Tree >>");
        System.out.println("Good Job! Your statment is Correct :)
.\n\n");
      }
      catch (Exception e)
      {
        System.out.println();
        System.out.println("Oops.Wrong statement, Try again :( \n\n");
        System.out.println(e.getMessage());
        masha.ReInit(System.in);
      }
      catch (Error e)
      {
        System.out.println("Oops. Not accepted statement, Try again :(
.\n\n");
        break;
      }
    }
  }
}

PARSER_END(masha)

/* White Spaces */
SKIP :
{
  " "
| "\r"
| "\t"
| "\n"
}

TOKEN : /* Identifiers and data type  */
{
  < STATIC_ID :
    "S" (< DIGIT >)+
    (
      (< SMALL_LETTER >)*
    | (< CAPITAL_LETTER >)*
    )+ >
  {
    System.out.println("You declare a static varible.");
  }
| < VARIABLE_ID :
    "V" (< DIGIT >)+
    (
      (< SMALL_LETTER >)*
    | (< CAPITAL_LETTER >)*
    )+ >
```

```
  {
    System.out.println("You declare a varible.");
  }
| < CONSTANT_ID :
    "C" (< DIGIT >)+
    (
      (< SMALL_LETTER >)*
    | (< CAPITAL_LETTER >)*
    )+ >
  {
    System.out.println("You declare a constant varible.");
  }
| < INT : (< DIGIT >)+ >
| < FLOAT : (< DIGIT >)+ (".") (< DIGIT >)+ >
| < STRING :
    "#"
    (
      (< SMALL_LETTER >)
    | (< CAPITAL_LETTER >)
    )+
    (< DIGIT >)* >
  //represent the list elements.
| < LIST_ELEMENTS :
    (
      (
        (
          < INT >
        | < FLOAT >
        | < STRING >
        )
        (",")
      )+
    )
    (
      < INT >
    | < FLOAT >
    | < STRING >
    ) >
| < #CAPITAL_LETTER : [ "A"-"Z" ] >
| < #DIGIT : [ "0"-"9" ] >
| < #SMALL_LETTER : [ "a"-"z" ] >
}

TOKEN : /* Keywords */
{
  < IF : "if" > // (if) and (else) used with conditional statements.
| < ELSE : "else" >
| < FUN : "Fun" > // To define the function (method)
| < UNTIL : "Until" > // used to
| < VAR : "var" >
| < LIST : "list" >
}
```

```
TOKEN : /* Arithmetic Operations */
{
  < PLUS : "+" >
| < MINUS : "-" >
| < MULTIPLY : "*" >
| < DIVIDE : "/" >
| < ASSIGNMENT : "=" >
}

TOKEN : /* Relational Operations */
{
  < EQUAL : "=?" >
| < INEQUAL : "~=?" >
| < LESS_THAN : "<?" >
| < LESS_OR_EQUAL : "<=?" >
| < GREAT_THAN : ">?" >
| < GREAT_OR_EQUAL : ">=?" >
}

TOKEN : /* Logical Operators */
{
  < AND : "&" >
| < OR : "OR" >
| < XOR : "XOR" >
| < NOT : "!" >
}

TOKEN : /* Punctuation Marks */
{
  < LEFT_PAR : "(" > //Left Parentheses
| < RIGHT_PAR : ")" > //Right Parentheses
| < LEFT_SQR_BRACKT : "[" > // Left Square Brackets
| < RIGHT_SQR_BRACKT : "]" > //Right Square Brackets
| < DOUBLE_QUOTES : " \" " > //Double Quotes
| < COMMA : "," >
| < FULL_STOP : "." >
| < COLON : ":" >
}

// Discard the comments
SPECIAL_TOKEN :
{
  < SINGLE_LINE_COMMENT :
    "::" (~[ "\n", "\r" ])*
    (
      "\n"
    | "\r"
    | "\r\n"
    ) >
  {
    System.out.println("Your write a comment, it is discarded!");
  }
}
```

```
SimpleNode Start() :
{/*@bgen(jjtree) Start */
  SimpleNode jjtn000 = new SimpleNode(JJTSTART);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) Start */
  try {
/*@egen*/
  statement()/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    return jjtn000;
  }/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void statement() :
{/*@bgen(jjtree) statement */
  SimpleNode jjtn000 = new SimpleNode(JJTSTATEMENT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) statement */
  try {
/*@egen*/
  (
    LOOKAHEAD(2) // look to the second position to identify the
operation.
```

```
    decleration()
  | logical_stmt()
  | conditional_stmt()
  | comparison_stmet()
  | loop()
  )
| list()/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void decleration() :
{/*@bgen(jjtree) decleration */
  SimpleNode jjtn000 = new SimpleNode(JJTDECLERATION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) decleration */
  try {
/*@egen*/
  // the statement should end with full stop (.)
  ///////////
  (
    LOOKAHEAD(2)
    declere_VAR()
  | arithmetic_stmt()
  )
  FULL_STOP()/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
```

```
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

// used to write an arithmetic expression.
void arithmetic_stmt() :
{/*@bgen(jjtree) arithmetic_stmt */
  SimpleNode jjtn000 = new SimpleNode(JJTARITHMETIC_STMT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) arithmetic_stmt */
  try {
/*@egen*/
  // the structure could be: var C0num + 5 OR  6*6
  (
    (
      ID()
    | number()
    )
    artimatic_operation()
    (
      ID()
    | number()
    )
  )+/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    System.out.println("You used an artimatic statment!");
  }/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
```

```
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

// this method used to declare a variable
void declere_VAR() :
{/*@bgen(jjtree) declere_VAR */
  SimpleNode jjtn000 = new SimpleNode(JJTDECLERE_VAR);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) declere_VAR */
  try {
/*@egen*/
  // the structure could be: var C0num = C1num  OR  var C0num = 5
  VAR() ID() ASSIGNMENT()
  (
    ID()
  | number()
  | STRING()
  )/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}
```

```
// work
void logical_stmt() :
{/*@bgen(jjtree) logical_stmt */
  SimpleNode jjtn000 = new SimpleNode(JJTLOGICAL_STMT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) logical_stmt */
  try {
/*@egen*/
  (
    Binary_logical_stmt()
  | unary_logical_stmt()
  )/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  // the structure could be: (V1num <? V2num) & (10 ~=? 50.0)
  {
    System.out.println("You used a logical statment!");
  }/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void unary_logical_stmt() :
{/*@bgen(jjtree) unary_logical_stmt */
  SimpleNode jjtn000 = new SimpleNode(JJTUNARY_LOGICAL_STMT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) unary_logical_stmt */
  try {
```

```
/*@egen*/
  NOT() LEFT_PAR() comparison_stmet() RIGHT_PAR()/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void Binary_logical_stmt() :
{/*@bgen(jjtree) Binary_logical_stmt */
  SimpleNode jjtn000 = new SimpleNode(JJTBINARY_LOGICAL_STMT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) Binary_logical_stmt */
  try {
/*@egen*/
  LEFT_PAR() comparison_stmet() RIGHT_PAR() logical_operation()
LEFT_PAR() comparison_stmet() RIGHT_PAR()/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
```

```
  }
/*@egen*/
}

void conditional_stmt() :
{/*@bgen(jjtree) conditional_stmt */
  SimpleNode jjtn000 = new SimpleNode(JJTCONDITIONAL_STMT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) conditional_stmt */
  try {
/*@egen*/
  // the structure is: if (V1num <? V2num): [V1num+1.]  else [
V2num+1.]
  // OR: if (V1num <? V2num): [V1num+1.] .
  IF() LEFT_PAR()
  (
    comparison_stmet()
  )
  RIGHT_PAR() COLON()
  LEFT_SQR_BRACKT()
  (
    statement()
  )+
  RIGHT_SQR_BRACKT()
  (
    else_stmt()
  | FULL_STOP()
  )/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    System.out.println("You used a conditional statment!");
  }/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
```

```
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void else_stmt() :
{/*@bgen(jjtree) else_stmt */
  SimpleNode jjtn000 = new SimpleNode(JJTELSE_STMT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) else_stmt */
  try {
/*@egen*/
  ELSE() LEFT_SQR_BRACKT()
  (
    statement()
  )+
  RIGHT_SQR_BRACKT()/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

// work
void comparison_stmet() :
{/*@bgen(jjtree) comparison_stmet */
  SimpleNode jjtn000 = new SimpleNode(JJTCOMPARISON_STMET);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) comparison_stmet */
  try {
/*@egen*/
```

```
  // the structure could be: V1num <? V2num OR 10 ~=? 50.0
  (
    ID()
  | number()
  )
  comparison_operation()
  (
    ID()
  | number()
  )/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    System.out.println("You used a comparison statment!");
  }/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

// used to create a loop
void loop() :
{/*@bgen(jjtree) loop */
  SimpleNode jjtn000 = new SimpleNode(JJTLOOP);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) loop */
  try {
/*@egen*/
  (
    until()
  | fun()
```

```
  )/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    System.out.println("You used an Iterative statement!");
  }/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void until() :
{/*@bgen(jjtree) until */
  SimpleNode jjtn000 = new SimpleNode(JJTUNTIL);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) until */
  try {
/*@egen*/
  /* the structure is:
Until ((V1num <? 5) & (V1num~=? -1)):
[V1num+1.]
 */
  UNTIL() LEFT_PAR()
  (
    comparison_stmet()
  | logical_stmt()
  )
  RIGHT_PAR() COLON()
  LEFT_SQR_BRACKT()
  (
    statement()
```

```
  )+
  RIGHT_SQR_BRACKT()/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void fun() :
{/*@bgen(jjtree) fun */
  SimpleNode jjtn000 = new SimpleNode(JJTFUN);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) fun */
  try {
/*@egen*/
  /* the structure:
Fun (0: 1 :5):
[V1num+1.]
 */
  FUN() LEFT_PAR() INT() COLON() INT() COLON() INT() RIGHT_PAR()
COLON()
  LEFT_SQR_BRACKT()
  (
    statement()
  )+
  RIGHT_SQR_BRACKT()/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    System.out.println("You used a function!");
  }/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
```

```
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void list() :
{/*@bgen(jjtree) list */
  SimpleNode jjtn000 = new SimpleNode(JJTLIST);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) list */
  try {
/*@egen*/
  //| < LIST_ELEMENTS :  >
  /* the structure:
list V1Addresses =[#Jeddah,#Makkah]. */

  (
    LIST() ID() ASSIGNMENT() LEFT_SQR_BRACKT()
    // the elements in the list
    LIST_ELEMENTS()
    // end of the list
    RIGHT_SQR_BRACKT() FULL_STOP()
  )/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    System.out.println("You used a list!");
  }/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
```

```
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void ID() :
{/*@bgen(jjtree) ID */
  SimpleNode jjtn000 = new SimpleNode(JJTID);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) ID */
  try {
/*@egen*/
  (
    STATIC_ID()
  | VARIABLE_ID()
  | CONSTANT_ID()
  )/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}
```

```
void number() :
{/*@bgen(jjtree) number */
  SimpleNode jjtn000 = new SimpleNode(JJTNUMBER);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) number */
  try {
/*@egen*/
  (
    INT()
  | FLOAT()
  )/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void artimatic_operation() :
{/*@bgen(jjtree) artimatic_operation */
  SimpleNode jjtn000 = new SimpleNode(JJTARTIMATIC_OPERATION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) artimatic_operation */
  try {
/*@egen*/
  (
    PLUS()
  | MINUS()
  | MULTIPLY()
  | DIVIDE()
  )/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
```

```
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void logical_operation() :
{/*@bgen(jjtree) logical_operation */
  SimpleNode jjtn000 = new SimpleNode(JJTLOGICAL_OPERATION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) logical_operation */
  try {
/*@egen*/
  (
    AND()
  | OR()
  | XOR()
  | NOT()
  )/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
```

```
  }
/*@egen*/
}

void comparison_operation() :
{/*@bgen(jjtree) comparison_operation */
  SimpleNode jjtn000 = new SimpleNode(JJTCOMPARISON_OPERATION);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/}
{/*@bgen(jjtree) comparison_operation */
  try {
/*@egen*/
  (
    EQUAL()
  | INEQUAL()
  | LESS_THAN()
  | LESS_OR_EQUAL()
  | GREAT_THAN()
  | GREAT_OR_EQUAL()
  )/*@bgen(jjtree)*/
  } catch (Throwable jjte000) {
    if (jjtc000) {
      jjtree.clearNodeScope(jjtn000);
      jjtc000 = false;
    } else {
      jjtree.popNode();
    }
    if (jjte000 instanceof RuntimeException) {
      throw (RuntimeException)jjte000;
    }
    if (jjte000 instanceof ParseException) {
      throw (ParseException)jjte000;
    }
    throw (Error)jjte000;
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

//terminals-------------
void FULL_STOP() :
{/*@bgen(jjtree) FULL_STOP */
  SimpleNode jjtn000 = new SimpleNode(JJTFULL_STOP);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) FULL_STOP */
```

```
  try {
/*@egen*/
  t0 = < FULL_STOP >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void ASSIGNMENT() :
{/*@bgen(jjtree) ASSIGNMENT */
  SimpleNode jjtn000 = new SimpleNode(JJTASSIGNMENT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) ASSIGNMENT */
  try {
/*@egen*/
  t0 = < ASSIGNMENT >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void STRING() :
{/*@bgen(jjtree) STRING */
  SimpleNode jjtn000 = new SimpleNode(JJTSTRING);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
```

```
  Token t0;
}
{/*@bgen(jjtree) STRING */
  try {
/*@egen*/
  t0 = < STRING >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void LIST() :
{/*@bgen(jjtree) LIST */
  SimpleNode jjtn000 = new SimpleNode(JJTLIST);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) LIST */
  try {
/*@egen*/
  t0 = < LIST >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void PLUS() :
{/*@bgen(jjtree) PLUS */
  SimpleNode jjtn000 = new SimpleNode(JJTPLUS);
```

```
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) PLUS */
  try {
/*@egen*/
  t0 = < PLUS >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void MINUS() :
{/*@bgen(jjtree) MINUS */
  SimpleNode jjtn000 = new SimpleNode(JJTMINUS);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) MINUS */
  try {
/*@egen*/
  t0 = < MINUS >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}
```

```
void MULTIPLY() :
{/*@bgen(jjtree) MULTIPLY */
  SimpleNode jjtn000 = new SimpleNode(JJTMULTIPLY);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) MULTIPLY */
  try {
/*@egen*/
  t0 = < MULTIPLY >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void DIVIDE() :
{/*@bgen(jjtree) DIVIDE */
  SimpleNode jjtn000 = new SimpleNode(JJTDIVIDE);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) DIVIDE */
  try {
/*@egen*/
  t0 = < DIVIDE >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
```

```
/*@egen*/
}

void INT() :
{/*@bgen(jjtree) INT */
  SimpleNode jjtn000 = new SimpleNode(JJTINT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) INT */
  try {
/*@egen*/
  t0 = < INT >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void FLOAT() :
{/*@bgen(jjtree) FLOAT */
  SimpleNode jjtn000 = new SimpleNode(JJTFLOAT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) FLOAT */
  try {
/*@egen*/
  t0 = < FLOAT >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
```

```
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void LEFT_PAR() :
{/*@bgen(jjtree) LEFT_PAR */
  SimpleNode jjtn000 = new SimpleNode(JJTLEFT_PAR);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) LEFT_PAR */
  try {
/*@egen*/
  t0 = < LEFT_PAR >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void RIGHT_PAR() :
{/*@bgen(jjtree) RIGHT_PAR */
  SimpleNode jjtn000 = new SimpleNode(JJTRIGHT_PAR);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) RIGHT_PAR */
  try {
/*@egen*/
  t0 = < RIGHT_PAR >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
```

```
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void IF() :
{/*@bgen(jjtree) IF */
  SimpleNode jjtn000 = new SimpleNode(JJTIF);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) IF */
  try {
/*@egen*/
  t0 = < IF >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void COLON() :
{/*@bgen(jjtree) COLON */
  SimpleNode jjtn000 = new SimpleNode(JJTCOLON);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) COLON */
  try {
/*@egen*/
  t0 = < COLON >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
```

```
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void ELSE() :
{/*@bgen(jjtree) ELSE */
  SimpleNode jjtn000 = new SimpleNode(JJTELSE);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) ELSE */
  try {
/*@egen*/
  t0 = < ELSE >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void LEFT_SQR_BRACKT() :
{/*@bgen(jjtree) LEFT_SQR_BRACKT */
  SimpleNode jjtn000 = new SimpleNode(JJTLEFT_SQR_BRACKT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) LEFT_SQR_BRACKT */
  try {
/*@egen*/
  t0 = < LEFT_SQR_BRACKT >/*@bgen(jjtree)*/
  {
```

```
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void RIGHT_SQR_BRACKT() :
{/*@bgen(jjtree) RIGHT_SQR_BRACKT */
  SimpleNode jjtn000 = new SimpleNode(JJTRIGHT_SQR_BRACKT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) RIGHT_SQR_BRACKT */
  try {
/*@egen*/
  t0 = < RIGHT_SQR_BRACKT >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void UNTIL() :
{/*@bgen(jjtree) UNTIL */
  SimpleNode jjtn000 = new SimpleNode(JJTUNTIL);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) UNTIL */
  try {
```

```
/*@egen*/
  t0 = < UNTIL >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void FUN() :
{/*@bgen(jjtree) FUN */
  SimpleNode jjtn000 = new SimpleNode(JJTFUN);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) FUN */
  try {
/*@egen*/
  t0 = < FUN >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void STATIC_ID() :
{/*@bgen(jjtree) STATIC_ID */
  SimpleNode jjtn000 = new SimpleNode(JJTSTATIC_ID);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
```

```
}
{/*@bgen(jjtree) STATIC_ID */
  try {
/*@egen*/
  t0 = < STATIC_ID >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void VARIABLE_ID() :
{/*@bgen(jjtree) VARIABLE_ID */
  SimpleNode jjtn000 = new SimpleNode(JJTVARIABLE_ID);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) VARIABLE_ID */
  try {
/*@egen*/
  t0 = < VARIABLE_ID >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void CONSTANT_ID() :
{/*@bgen(jjtree) CONSTANT_ID */
  SimpleNode jjtn000 = new SimpleNode(JJTCONSTANT_ID);
  boolean jjtc000 = true;
```

```
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) CONSTANT_ID */
  try {
/*@egen*/
  t0 = < CONSTANT_ID >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void AND() :
{/*@bgen(jjtree) AND */
  SimpleNode jjtn000 = new SimpleNode(JJTAND);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) AND */
  try {
/*@egen*/
  t0 = < AND >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void OR() :
```

```
{/*@bgen(jjtree) OR */
  SimpleNode jjtn000 = new SimpleNode(JJTOR);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) OR */
  try {
/*@egen*/
  t0 = < OR >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void XOR() :
{/*@bgen(jjtree) XOR */
  SimpleNode jjtn000 = new SimpleNode(JJTXOR);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) XOR */
  try {
/*@egen*/
  t0 = < XOR >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
```

```
}

void NOT() :
{/*@bgen(jjtree) NOT */
  SimpleNode jjtn000 = new SimpleNode(JJTNOT);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) NOT */
  try {
/*@egen*/
  t0 = < NOT >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void EQUAL() :
{/*@bgen(jjtree) EQUAL */
  SimpleNode jjtn000 = new SimpleNode(JJTEQUAL);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) EQUAL */
  try {
/*@egen*/
  t0 = < EQUAL >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
```

```
    }
  }
/*@egen*/
}

void INEQUAL() :
{/*@bgen(jjtree) INEQUAL */
  SimpleNode jjtn000 = new SimpleNode(JJTINEQUAL);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) INEQUAL */
  try {
/*@egen*/
  t0 = < INEQUAL >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void LESS_THAN() :
{/*@bgen(jjtree) LESS_THAN */
  SimpleNode jjtn000 = new SimpleNode(JJTLESS_THAN);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) LESS_THAN */
  try {
/*@egen*/
  t0 = < LESS_THAN >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
```

```
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void LESS_OR_EQUAL() :
{/*@bgen(jjtree) LESS_OR_EQUAL */
  SimpleNode jjtn000 = new SimpleNode(JJTLESS_OR_EQUAL);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) LESS_OR_EQUAL */
  try {
/*@egen*/
  t0 = < LESS_OR_EQUAL >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void GREAT_THAN() :
{/*@bgen(jjtree) GREAT_THAN */
  SimpleNode jjtn000 = new SimpleNode(JJTGREAT_THAN);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) GREAT_THAN */
  try {
/*@egen*/
  t0 = < GREAT_THAN >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
```

```
    {
      jjtn000.jjtSetValue(t0.image);
    }/*@bgen(jjtree)*/
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  /*@egen*/
  }

  void GREAT_OR_EQUAL() :
  {/*@bgen(jjtree) GREAT_OR_EQUAL */
    SimpleNode jjtn000 = new SimpleNode(JJTGREAT_OR_EQUAL);
    boolean jjtc000 = true;
    jjtree.openNodeScope(jjtn000);
  /*@egen*/
    Token t0;
  }
  {/*@bgen(jjtree) GREAT_OR_EQUAL */
    try {
  /*@egen*/
    t0 = < GREAT_OR_EQUAL >/*@bgen(jjtree)*/
    {
      jjtree.closeNodeScope(jjtn000, true);
      jjtc000 = false;
    }
  /*@egen*/
    {
      jjtn000.jjtSetValue(t0.image);
    }/*@bgen(jjtree)*/
    } finally {
      if (jjtc000) {
        jjtree.closeNodeScope(jjtn000, true);
      }
    }
  /*@egen*/
  }

  void LIST_ELEMENTS() :
  {/*@bgen(jjtree) LIST_ELEMENTS */
    SimpleNode jjtn000 = new SimpleNode(JJTLIST_ELEMENTS);
    boolean jjtc000 = true;
    jjtree.openNodeScope(jjtn000);
  /*@egen*/
    Token t0;
  }
  {/*@bgen(jjtree) LIST_ELEMENTS */
    try {
  /*@egen*/
    t0 = < LIST_ELEMENTS >/*@bgen(jjtree)*/
    {
      jjtree.closeNodeScope(jjtn000, true);
```

```
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}

void VAR() :
{/*@bgen(jjtree) VAR */
  SimpleNode jjtn000 = new SimpleNode(JJTVAR);
  boolean jjtc000 = true;
  jjtree.openNodeScope(jjtn000);
/*@egen*/
  Token t0;
}
{/*@bgen(jjtree) VAR */
  try {
/*@egen*/
  t0 = < VAR >/*@bgen(jjtree)*/
  {
    jjtree.closeNodeScope(jjtn000, true);
    jjtc000 = false;
  }
/*@egen*/
  {
    jjtn000.jjtSetValue(t0.image);
  }/*@bgen(jjtree)*/
  } finally {
    if (jjtc000) {
      jjtree.closeNodeScope(jjtn000, true);
    }
  }
/*@egen*/
}
```

- **jjt file**

```
/**
 * JJTree template file created by SF JavaCC plugin 1.5.28+ wizard for
JavaCC 1.5.0+
 */
/*GROUP1- IAR:
Mona Hafez.
```

```
Bodor Alamri
Razan Aljuhani
Shuroog Alshaikh
 */
options
{
  static = true;
}

PARSER_BEGIN(masha)
package MASHA_Language;

public class masha
{
  public static void main(String args [])
  {
    System.out.println("   Welcome to Masha language");
    System.out.print("------------------------------\n");
    masha parser = new masha(System.in);

    while(true)
    {
    System.out.println("\nPlease enter an expression:");
    try
    {
      SimpleNode n = masha.Start();
      n.dump("Masha Tree >>");
      System.out.println("Good Job! Your statment is Correct :)
.\n\n");
    }
    catch (Exception e)
    {
      System.out.println();
        System.out.println("Oops.Wrong statement, Try again :( \n\n");
        System.out.println(e.getMessage());
        masha.ReInit(System.in);
      }
    }
    catch (Error e)
    {
      System.out.println("Oops. Not accepted statement, Try again :(
.\n\n");
      break;
     }

  }
}
PARSER_END(masha)

/* White Spaces */
SKIP :
{
  " "
```

```
| "\r"
| "\t"
| "\n"
}

TOKEN :/* Identifiers and data type  */

 {
  < STATIC_ID :"S"(< DIGIT >)+ ((<SMALL_LETTER>)* | (<
CAPITAL_LETTER>)*)+  >
  { System.out.println("You declare a static varible.");

  }
| < VARIABLE_ID :"V"(< DIGIT >)+ ((<SMALL_LETTER>)* | (<
CAPITAL_LETTER>)*)+ >
{ System.out.println("You declare a varible.");
}
| < CONSTANT_ID :"C"(< DIGIT >)+ ((<SMALL_LETTER>)* | (<
CAPITAL_LETTER>)*)+  >
{
  System.out.println("You declare a constant varible.");
}

| < INT: (< DIGIT >)+ >
| < FLOAT : (< DIGIT >)+ (".") (< DIGIT >)+ >
| < STRING :  "#" ((<SMALL_LETTER>) | (< CAPITAL_LETTER>))+ (< DIGIT
>)* >

//represent the list elements.
| < LIST_ELEMENTS :  (((< INT >|< FLOAT >|< STRING >)(","))+ ) (< INT
>|<FLOAT >|< STRING >) >
| < #CAPITAL_LETTER: ["A"-"Z"] >
| < #DIGIT: ["0"-"9"] >
| < #SMALL_LETTER: ["a"-"z"] >
}

TOKEN : /* Keywords */
{
  < IF : "if" >  // (if) and (else) used with conditional statements.
| < ELSE : "else" >
| < FUN  : "Fun" > // To define the function (method)
| < UNTIL: "Until" > // used to
| < VAR: "var" >
| < LIST: "list" >
}


TOKEN : /* Arithmetic Operations */

{
  < PLUS : "+" >
| < MINUS : "-" >
| < MULTIPLY : "*" >
```

```
| < DIVIDE : "/" >
| < ASSIGNMENT : "=" >

}


TOKEN : /* Relational Operations */
{
  < EQUAL : "=?" >
| < INEQUAL : "~=?" >

| < LESS_THAN : "<?" >
| < LESS_OR_EQUAL : "<=?" >

| < GREAT_THAN : ">?" >
| < GREAT_OR_EQUAL : ">=?" >
}


TOKEN : /* Logical Operators */
{
  < AND : "&" >
| < OR : "OR" >
| < XOR : "XOR" >
| < NOT : "!" >
}


TOKEN : /* Punctuation Marks */
{
  < LEFT_PAR : "(" >   //Left Parentheses
| < RIGHT_PAR : ")" >  //Right Parentheses
| < LEFT_SQR_BRACKT : "[" > // Left Square Brackets
| < RIGHT_SQR_BRACKT  : "]" > //Right Square Brackets
| < DOUBLE_QUOTES: " \" " > //Double Quotes
| < COMMA: "," >
| < FULL_STOP: "." >
| < COLON: ":" >
}



// Discard the comments
SPECIAL_TOKEN : {
 < SINGLE_LINE_COMMENT: "::" (~["\n","\r"])* ("\n"|"\r" |"\r\n") >
{
  System.out.println("Your write a comment, it is discarded!");
}
}


SimpleNode Start() :
{}
```

```
{
statement()
{
  return jjtThis;
}
}

void statement() :
{}
{

  ( LOOKAHEAD(2) // look to the second position to identify the
operation.

 decleration()
| logical_stmt()
| conditional_stmt()
| comparison_stmet()
| loop()


)|list()

}


void decleration() :
{}
{
   // the statement should end with full stop (.)
   ///////////
  ( LOOKAHEAD(2) declere_VAR() | arithmetic_stmt()) FULL_STOP()

}

// used to write an arithmetic expression.
void arithmetic_stmt() :
{}
{
// the structure could be: var C0num + 5 OR  6*6
(( ID() | number()) artimatic_operation() ( ID() | number()) )+

{
    System.out.println("You used an artimatic statment!");
}
}

// this method used to declare a variable
void declere_VAR() :
{}
{
// the structure could be: var C0num = C1num  OR  var C0num = 5
 VAR() ID() ASSIGNMENT() ( ID() | number() | STRING() )
```

```
}

// work
void logical_stmt() :
{}
{
  ( Binary_logical_stmt() | unary_logical_stmt())
  // the structure could be: (V1num <? V2num) & (10 ~=? 50.0)
{
    System.out.println("You used a logical statment!");

}
}

void unary_logical_stmt() :
{}
{
 NOT() LEFT_PAR() comparison_stmet() RIGHT_PAR()

}
void Binary_logical_stmt() :
{}
{

  LEFT_PAR() comparison_stmet() RIGHT_PAR() logical_operation()
LEFT_PAR() comparison_stmet()  RIGHT_PAR()

}


void conditional_stmt() :
{}
{
  // the structure is: if (V1num <? V2num): [V1num+1.]  else [
V2num+1.]
  // OR: if (V1num <? V2num): [V1num+1.] .
IF() LEFT_PAR() (comparison_stmet()) RIGHT_PAR() COLON()
LEFT_SQR_BRACKT()  (statement())+ RIGHT_SQR_BRACKT() (else_stmt() |
FULL_STOP())
{
    System.out.println("You used a conditional statment!");

}
}

void else_stmt() :
{}
{

ELSE() LEFT_SQR_BRACKT()  (statement())+ RIGHT_SQR_BRACKT()

}
```

```
// work
void comparison_stmet():
{}
{
  // the structure could be: V1num <? V2num OR 10 ~=? 50.0

 (ID() | number()) comparison_operation()  (ID() | number())

{
    System.out.println("You used a comparison statment!");

}
}

// used to create a loop
void loop() :
{}
{
 (until()| fun())

{
  System.out.println("You used an Iterative statement!");

}
}

void until() :
{}
{
 /* the structure is:
Until ((V1num <? 5) & (V1num~=? -1)):
[V1num+1.]
 */
 UNTIL() LEFT_PAR() (comparison_stmet()| logical_stmt()) RIGHT_PAR()
COLON()
 LEFT_SQR_BRACKT()  (statement())+  RIGHT_SQR_BRACKT()
 }


void fun() :
{}
{
  /* the structure:
Fun (0: 1 :5):
[V1num+1.]
 */
 FUN() LEFT_PAR() INT()  COLON()  INT() COLON()  INT() RIGHT_PAR()
COLON()
 LEFT_SQR_BRACKT()   (statement())+  RIGHT_SQR_BRACKT()
 {
    System.out.println("You used a function!");
}
```

```
}

void list() :
{}
{
//| < LIST_ELEMENTS :  >

  /* the structure:
list V1Addresses =[#Jeddah,#Makkah]. */

 ( LIST() ID() ASSIGNMENT() LEFT_SQR_BRACKT()
 // the elements in the list
 LIST_ELEMENTS()
  // end of the list
  RIGHT_SQR_BRACKT() FULL_STOP() )
 {
    System.out.println("You used a list!");
}

}


void ID() :
{}
{
(
  STATIC_ID()
| VARIABLE_ID()
| CONSTANT_ID()  )
}


void number() :
{}
{
(
 INT()| FLOAT()
)

}

void artimatic_operation() :
{}
{
(
 PLUS()
| MINUS()
| MULTIPLY()
| DIVIDE()
)
}
```

```
void logical_operation() :
{}
{
(
  AND()
| OR()
| XOR()
| NOT()
)
}


void comparison_operation() :
{}
{
(
  EQUAL()
| INEQUAL()
| LESS_THAN()
| LESS_OR_EQUAL()
| GREAT_THAN()
| GREAT_OR_EQUAL()
)

}

//terminals-------------
void FULL_STOP(): {Token t0; }
{
  t0= < FULL_STOP >{ jjtThis.jjtSetValue(t0.image); }
}

void ASSIGNMENT(): {Token t0; }
{
  t0= < ASSIGNMENT >{ jjtThis.jjtSetValue(t0.image); }
}
void STRING(): {Token t0; }
{
  t0=< STRING >{ jjtThis.jjtSetValue(t0.image); }
}
void LIST(): {Token t0; }
{
  t0= < LIST>{ jjtThis.jjtSetValue(t0.image); }
}

void PLUS(): {Token t0; }
{
  t0= < PLUS >{ jjtThis.jjtSetValue(t0.image); }
}
void MINUS(): {Token t0; }
{
  t0= < MINUS >{ jjtThis.jjtSetValue(t0.image); }
```

```
}
void MULTIPLY(): {Token t0; }
{
  t0= < MULTIPLY >{ jjtThis.jjtSetValue(t0.image); }
}
void DIVIDE(): {Token t0; }
{
  t0= < DIVIDE >{ jjtThis.jjtSetValue(t0.image); }
}

void INT(): {Token t0 ;}
{
  t0= < INT > {jjtThis.jjtSetValue(t0.image);}

}
void FLOAT():{Token t0; }
{
  t0=< FLOAT >{jjtThis.jjtSetValue(t0.image);}
}

void LEFT_PAR(): {Token t0; }
{
  t0=< LEFT_PAR > { jjtThis.jjtSetValue(t0.image); }
}

void RIGHT_PAR(): {Token t0; }
{
  t0=< RIGHT_PAR > { jjtThis.jjtSetValue(t0.image); }
}

void IF(): {Token t0; }
{
  t0= < IF>{ jjtThis.jjtSetValue(t0.image); }
}
void COLON(): {Token t0; }
{
  t0=< COLON > { jjtThis.jjtSetValue(t0.image); }
}
void ELSE(): {Token t0; }
{
  t0= < ELSE>{ jjtThis.jjtSetValue(t0.image); }
}
void LEFT_SQR_BRACKT(): {Token t0; }
{
  t0=< LEFT_SQR_BRACKT > { jjtThis.jjtSetValue(t0.image); }
}
void RIGHT_SQR_BRACKT(): {Token t0; }
{
  t0=< RIGHT_SQR_BRACKT > { jjtThis.jjtSetValue(t0.image); }
}

void UNTIL(): {Token t0; }
{
```

```
  t0=< UNTIL > { jjtThis.jjtSetValue(t0.image); }
}

void FUN(): {Token t0; }
{
  t0=< FUN > { jjtThis.jjtSetValue(t0.image); }
}

void STATIC_ID(): {Token t0; }
{
  t0=< STATIC_ID > { jjtThis.jjtSetValue(t0.image); }
}
void VARIABLE_ID(): {Token t0; }
{
  t0=< VARIABLE_ID > { jjtThis.jjtSetValue(t0.image); }
}
void CONSTANT_ID(): {Token t0; }
{
  t0=< CONSTANT_ID > { jjtThis.jjtSetValue(t0.image); }
}

void AND(): {Token t0; }
{
  t0=< AND> { jjtThis.jjtSetValue(t0.image); }
}
void OR(): {Token t0; }
{
  t0=< OR> { jjtThis.jjtSetValue(t0.image); }
}
void XOR(): {Token t0; }
{
  t0=< XOR> { jjtThis.jjtSetValue(t0.image); }
}
void NOT(): {Token t0; }
{
  t0=< NOT> { jjtThis.jjtSetValue(t0.image); }
}

 void EQUAL():{Token t0;}
{
  t0=< EQUAL >{ jjtThis.jjtSetValue(t0.image); }
}
void INEQUAL():{Token t0;}
{
  t0=< INEQUAL >{ jjtThis.jjtSetValue(t0.image); }
}
void LESS_THAN():{Token t0;}
{
  t0=< LESS_THAN >{ jjtThis.jjtSetValue(t0.image); }
}
void LESS_OR_EQUAL():{Token t0;}
{
  t0=< LESS_OR_EQUAL >{ jjtThis.jjtSetValue(t0.image); }
```

```
}
void GREAT_THAN():{Token t0;}
{
  t0=< GREAT_THAN >{ jjtThis.jjtSetValue(t0.image); }
}
void GREAT_OR_EQUAL():{Token t0;}
{
  t0=< GREAT_OR_EQUAL >{ jjtThis.jjtSetValue(t0.image); }
}

void LIST_ELEMENTS():{Token t0;}
{
  t0=< LIST_ELEMENTS >{ jjtThis.jjtSetValue(t0.image); }
}

void VAR():{Token t0;}
{
  t0=< VAR >{ jjtThis.jjtSetValue(t0.image); }
}
```