



**UNIVERSITY OF SOUTH BRITTANY**  
**FACULTY OF SCIENCES AND ENGINEERING**  
**MASTER 1 CYBERUS**

**Compiler Construction**  
**RM Programming Language**  
**Project Report**

**Contributors**

Mashal ZAINAB  
Ribiea RAMZAN

**Supervisor:**

Salah SADOU

31st March, 2023

## **Introduction:**

This report explains the compiler construction process of the RM programming language which was designed for the course. The zip files containing all the code files are provided with this report.

RM programming language is basic procedural programming language containing the basic functionalities of any common programming language such as variables, their assignment, initialization and declaration, arithmetic expressions, relational expressions, logical expressions, conditional expressions (if, if else and else), functions, procedures, arrays and loops (while and for each loop). All these statements combinatively make a program in RM programming language. It is a typed language, where all the variables or functions are defined with a type. The types include integers, floats, booleans and strings. The language also supports single line and multi line comments which are defined by \$ and \$\$ respectively.

## **Grammar:**

The first step in designing the compiler was to define the grammar for the language in Backus-Naur form (BNF). This grammar is used to define the rules of the language.

## **Compiler Design:**

The compiler for RM programming language is designed using JavaCC lexer and parser generator which is written in java. We use the above created BNF grammar and perform the following analyses for the construction of our compiler and the translation process.

### **1. Lexical Analysis:**

The lexical description of the language in JavaCC is specified in the jj file. We define tokens for lexical analysis in the statements.jj file. So the lexical analysis extracts individual words from our input statements and recognizes them as proper tokens.

The tokens defined in our jj file are the specifications for literals or variables i.e., what should be contained in strings, integers, floats and booleans, variable names in BNF Grammar. It also includes reserved keywords such as 'function', 'if', 'while', 'integer', 'float', 'boolean', 'string', 'return', 'void', 'foreach' and all the characters that are used to perform operations. The tokens also include whitespaces which should be skipped and how comments are defined.

statementsTokenManager.java file contains the lexical analyzer, which is a javaCC generated file.

### **2. Syntactic Analysis:**

The syntactic description of the language in JavaCC is specified in the jj file. We use the syntactic analysis to determine if the series of tokens passed as input statements are valid or not. During the syntactic analysis, our compiler examines the program input code with respect to the rules defined by us in the statements.jj file. Below the tokens in statements.jj file we have defined all the rules of the grammar that should be followed by our language. Basically we have defined the syntax of our language in this part.

statements.java file contains the syntactical analyzer, which is also a javaCC generated file.

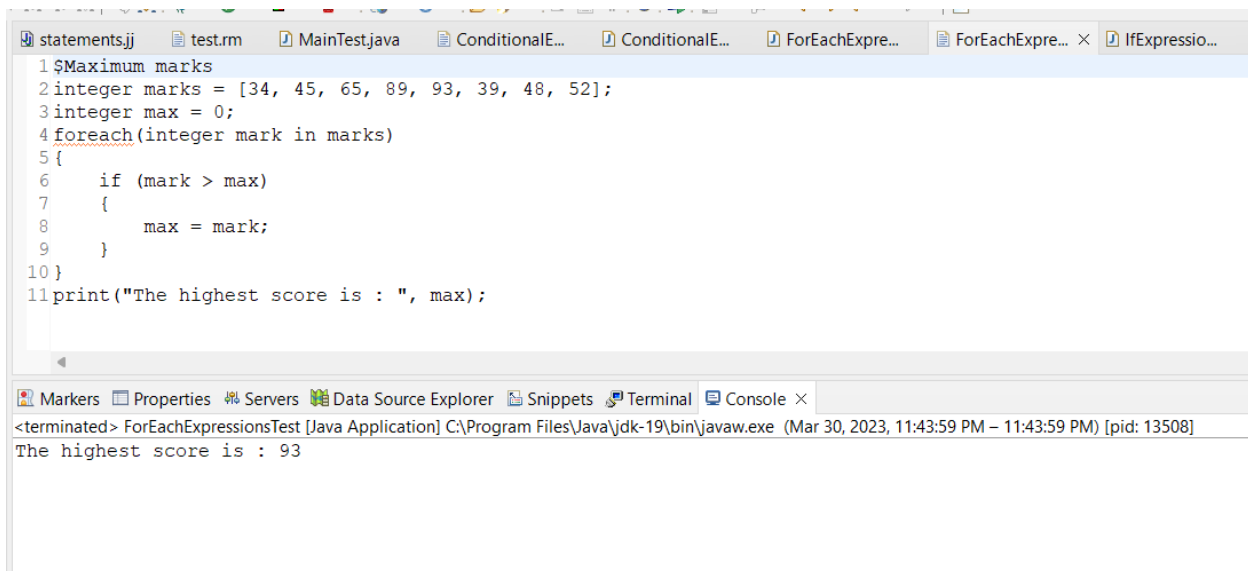
### 3. Semantic Analysis:

The semantic analysis checks the semantic correctness of our language. In order to do that we have created multiple extension classes and methods which are called in the jj file, as defined in our grammar. In the extension classes, we perform type checking of the input variables and operands. For example, the Declaration.java file in the statementparser package in our project contains type checks for the declaration of variables to be semantically consistent or not.

### Example breakdown:

In this section we provide an example program in the RM programming language, and also explain the implementation details for it.

In the following example we are checking the maximum number in an array. We are using arrays, integer variable declaration and assignment, foreach loop, if condition expression, and print statement.



```
1 $Maximum marks
2 integer marks = [34, 45, 65, 89, 93, 39, 48, 52];
3 integer max = 0;
4 foreach(integer mark in marks)
5 {
6     if (mark > max)
7     {
8         max = mark;
9     }
10 }
11 print("The highest score is : ", max);
```

The screenshot shows an IDE with several tabs open: statements.jj, test.rm, MainTest.java, ConditionalE..., ConditionalE..., ForEachExpre..., ForEachExpre..., and IfExpressio... The code in statements.jj is as follows:

```
1 $Maximum marks
2 integer marks = [34, 45, 65, 89, 93, 39, 48, 52];
3 integer max = 0;
4 foreach(integer mark in marks)
5 {
6     if (mark > max)
7     {
8         max = mark;
9     }
10 }
11 print("The highest score is : ", max);
```

The bottom of the IDE shows a console window with the output: "The highest score is : 93". The console title is "<terminated> ForEachExpressionsTest [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (Mar 30, 2023, 11:43:59 PM - 11:43:59 PM) [pid: 13508]".

### 1. Array Declaration:

We have implemented following grammar for array in the statements.jj file:

```

//Arrays
ArrayStatement arrayStatementParser() throws NumberFormatException:
{
    Identifier id = null;
    Token t = null;
    Literal l = null;
    List<Expression> literalList = null;
}
{
    t = < OBJECTTYPE > id=identifierParser() < EQUALSTO > < LSP >
        { literalList = new ArrayList<Expression>(); }
        (
            l = literalParser()
            { literalList.add(l); }
            (
                <COMMA>
                l = literalParser()
                { literalList.add(l); }
            )*
        )
        < RSP > ";"
}
{
    return new ArrayStatement(t.image, id, literalList);
}
}

```

Here, the literalParser() is another function based on the class Literal, which is used for parsing types like integer, string, float and boolean.

For implementation of array grammar, we are using list and Array list functions of java to store the values of literalParser(), and then in return we return a new object of ArrayStatement class with passing objecttype, identifier, and list in constructor.

For storing variables, we are using hashmap. In the ArrayStatement class, we are adding the list in a hashmap with an identifier name (variable name) by calling the function of class Context. Context class is used for handling declaration and storing information of variables, functions, and procedures.

## 2. Integer Declaration:

The second statement is the variable declaration. In the same manner as array declaration, we are using the declaration class to add integer max = 0; which calls the instance/object of Context class to store variable name and value.

## 3. Foreach Loop:

Then we are using a foreach loop to traverse through the array.  
Grammar for foreach defined in statements.jj file:

```

//Foreach Expression
@ForeachExpression foreachParser():
{
    Identifier variablename;
    Identifier arrayname;
    Statement statement = new EmptyStatement();
    Token type;
}
{
    (
        <FOREACH> <LP> type= < OBJECTTYPE > variablename=identifierParser() < IN > arrayname=identifierParser()
        <RP> <LGP>
        statement=multiStatement()
        <RGP> )
    {
        return new ForEachExpression(type.image, variablename, arrayname, statement);
    }
}

```

First variable name token is for the variable we will use to traverse the array, and array name is the name of array. Here, identifierParser() function returns the string of <VARIABLENAME> token image.

In the ForEachExpression class, we get the array value from hashmap and run for loop of java on the statements provided in the constructor.

#### 4. If expression:

Then inside the foreach loop we are using the if expression.

```

IfExpression ifParser():
{
    Expression condition = null;
    Statement thenStatement = new EmptyStatement();
    Statement elseStatement = new EmptyStatement();
}
{
    (
        <IF> <LP> condition=expressionParser() <RP><LGP>
        thenStatement=multiStatement()
        <RGP> [ <ELSE>
        <LGP>
        elseStatement=multiStatement()
        <RGP>]
    )
    {
        return new IfExpression(condition, thenStatement, elseStatement);
    }
}

```

We are passing three arguments to the constructor, condition for if, statements to be executed in case condition is true, and statement for else. In the class, condition is evaluated and if else is implemented in java. Here in this example we are passing a condition if (mark > max) to check this logical expression is called inside expressionParser().

expressionParser() function is responsible for handling all arithmetic, logical, and relational expressions.

**5. Assignment Statement:**

We have assignment statements inside of if condition, which work in the same way as declaration using instance/object of Context class.

**6. Print Statement:**

In print statement class, we have used the System.out.print() function of java.