RSS Programming Language

Reference Manual

Team 24:

Richa Shastri - 1209458663

Sarvesh Kulkarni - 1210322864

Subhankar Mishra - 1209249583

Contents

* Introduction
* Tools Used
* Language Inspired From
* Syntax
* Features
* How it works
* Code Transition
* Grammar
* Parse Tree
* RSS Compiler
* Intermediate Language Syntax
* RSS Runtime
* Symbol Table
* Challenges
* References.

Introduction:

The new language we have designed is called RSS. It’s a high level, user friendly and procedural language. It is designed for users who are using programming languages for the first time as it is very user friendly and its syntax is very simple and easy to understand. The language is extremely powerful in terms of functionality.

It supports operators, conditional statements, data types, a data structure such as array, functions with return declarations and argument calls, iterations and recursive calls. It is enclosed in class declarations to enable object oriented approach in the future. The compiler of our language is extremely intelligent and does maximum processing, syntax checking, and function call handling. This helps the runtime to be very fast and efficient. RSS is scalable enough and versatile that any runtime can be built upon it.

RSS file is stored as .rss format. The compiler generates intermediate code with format .rssc which is then run on the runtime interpreter.

Tools Used:

The tools that were used to build the compiler and runtime interpreter are

* Lexical Analysis and Parsing : ANTLR V4
* Intermediate Code Generation : ANTLR V4 and Java
* Runtime Environment : Java

Languages Inspired from:

RSS is inspired from various languages like C, C++, Java and Tiny ADA.

* It is static, procedural and strongly typed like C++.
* The condition check in loops and language functional handling part is inspired from Java.
* It deals with dangling else problem like ADA by using elseif and endif statements.
* Writing structure of our language is inspired from Visual BASIC..

Syntax:

The keywords of our high level language are given below

|  |  |
| --- | --- |
| **Keywords** | **Meaning** |
| class | Beginning of a Class Declaration |
| func | Declaring a function |
| returns | Defining the date type of the value the function returns |
| return | The return statement |
| for | For loop |
| till | Limiter of the for loop |
| if | conditional statement |
| else | Conditional statement |
| end | Terminator for a class, function, for loop and if and if else |
| print | Printing a value |
| \*,/,%,+,- | Arithmetic operators |
| >=,<=,>,<,==,!= | Conditional operators |
| &&, || | Logical operators |
| :: | Scope resolution operator |

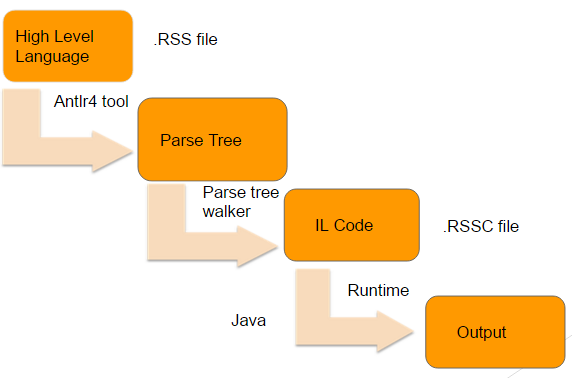
Features:

Our language contains the following features

* Local and global variables: Our language supports local and global variables and treat them differently. To access a global variable with the same name as local variable you need to use a scope resolution operator. Our compiler is smart enough to handle different variables with same name but with different scope and data types.
* Data types: The following data types are supported for local, global and arrays.
  + - Int
    - String
    - Boolean
* Data structure:
  + - We have implement a sequential data structure - Array.
    - Array can be declared with n size.
    - Both array and its members can be passed as an argument in function call.
* Operators: Following operators are supported in the order of the priority
  + - Arithmetic : \*,/,%,+,-,
    - Conditional: >=,<=,>,<,==,!=
    - Logical: &&, ||
* Complex expressions: Handles complex expressions such as shown below a=func1(func1(x+2)))+a\*b + a/b
* Conditional operations
  + - Implements condition statement such as If , else if , else
    - Handles dangling else problem
    - Else Statement is not mandatory
    - Condition checks in IF can be expressions, variables or boolean literals like true and false
* Iterative operations
  + - Implements a loop similar to ‘For’ loop in C++.
* Function declaration
  + - Declare a function with arguments and return value data type.
    - Arguments are passed by value.
    - Functions can be called from anywhere in the program.
* Function call
  + - Before calling a function it is not necessary that the function be declared or defined.
    - Return statements are optional.
* Recursive Function call
  + - A function can be called to itself any number of times.

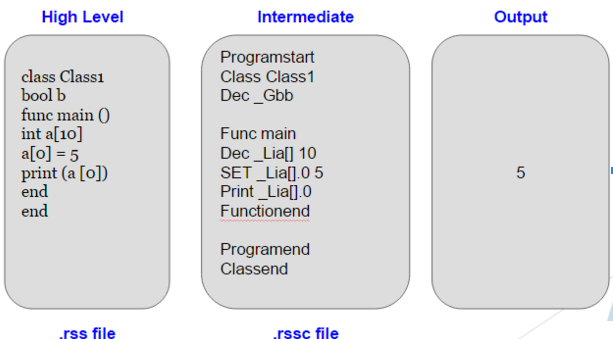
How it works:

This workflow of our language is shown below

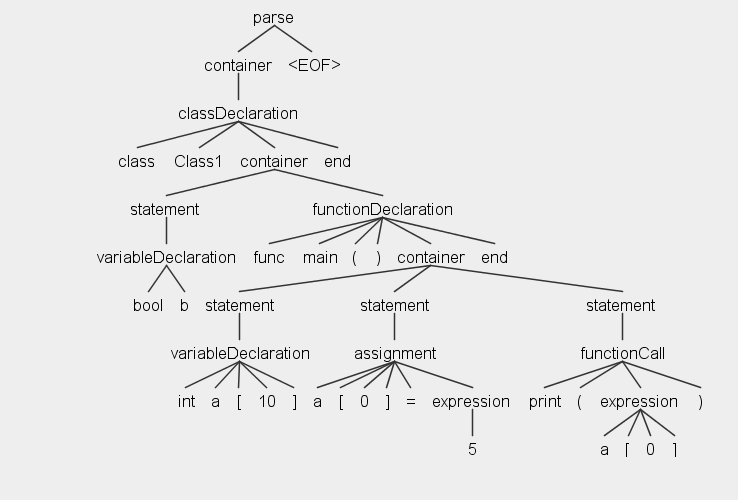


The .rss file is fed into the ANTLR tool to perform scanning and parsing. The ANTLR parser and scanner class helps us to generate the parse tree. We use the walker class to traverse through the parse tree. Creating an instance of the walker class invokes the functions of the base listener class. Baselistener class is generated by ANTLR that contains all the entry and exit functions of the rules stated in grammar. We override each of these functions to generate the intermediate code. Once all the nodes are traversed our intermediate file is generated with .rssc extension. The intermediate file with .rssc extension is fed into the runtime which is created by Java. The runtime environment processes the .rssc file and gives the output.

Code transition:



Its corresponding parse tree is shown below



Grammar:

The Syntactic grammar of our language is given below:

*parse : container EOF ;*

*container : (statement | functionDeclaration | classDeclaration)\* ;*

*statement : assignment | functionCall | ifStatement | forStatement | variableDeclaration | returnDeclaration ;*

*assignment : Scope? Identifier ('[' (Number | Identifier)? ']')? Assign expression ;*

*functionCall : Identifier '(' exprList? ')' #identifierFunctionCall*

*| Print '(' expression ')' #printFunctionCall ;*

*ifStatement : ifStat elseIfStat\* elseStat? End ;*

*ifStat : If expression container ;*

*elseIfStat : Else If expression container ;*

*elseStat : Else container ;*

*functionDeclaration : Func Identifier '(' argList? ')' (Returns Identifier)? container (returnDeclaration)? End ;*

*returnDeclaration : Return expression ;*

*argList : variableDeclaration (variableDeclaration)\* ;*

*classDeclaration : Class Identifier container End ;*

*forStatement : For Identifier Assign expression Till expression container End ;*

*variableDeclaration : Identifier Identifier ('[' Number? ']')? ;*

*exprList : expression (',' expression)\* ;*

*expression :*

*expression '\*' expression #multiplyExpression*

*| expression '/' expression #divideExpression*

*| expression '%' expression #modulusExpression*

*| expression '+' expression #addExpression*

*| expression '-' expression #subtractExpression*

*| expression '>=' expression #gtEqExpression*

*| expression '<=' expression #ltEqExpression*

*| expression '>' expression #gtExpression*

*| expression '<' expression #ltExpression*

*| expression '==' expression #eqExpression*

*| expression '!=' expression #notEqExpression*

*| expression '&&' expression #andExpression*

*| expression '||' expression #orExpression*

*| Number #numberExpression*

*| Scope? Identifier ('[' (Number | Identifier)? ']')? #identifierExpression*

*| String #stringExpression*

*| functionCall #functionCallExpression*

*| '(' expression ')' #expressionExpression*

*;*

Intermediate Language Syntax:

|  |  |
| --- | --- |
| Keywords | Meaning |
| Programstart | Starting of Program |
| Programend | End of Program |
| Class classname | Starting of Class having name as classname |
| Classend | End of Class. It pops out all the entries of latest class |
| Func fname | Starting of function having name as fname |
| Functionend | End of Function. It pops out all the entries of latest function |
| dec \_Giabc | Declaration of global integer variable having name "abc" |
| dec \_Gsabc | Declaration of global string variable having name "abc" |
| dec\_Gbabc | Declaring of global boolean variable having name “abc” |
| dec\_Giabc 10 | Declaration of a Global integer array of size 10 with name “abc” |
| dec\_Gsabc 10 | Declaration of a Global string array of size 10 with name “abc” |
| dec\_Gbabc 10 | Declaration of a Global boolean array of size 10 with name “abc” |
| dec \_Liabc | Declaration of local integer variable having name "abc" |
| dec \_Lsabc | Declaration of local string variable having name "abc" |
| dec \_Lbabc | Declaration of local boolean variable having name "abc" |
| dec\_Liabc 10 | Declaration of a local integer array of size 10 with name “abc” |
| dec\_Lsabc 10 | Declaration of a Global string array of size 10 with name “abc” |
| dec\_Lbabc 10 | Declaration of a Global boolean array of size 10 with name “abc” |
| Set | Set a value to a variable. Ex: Set \_Lsabc 5 |
| Pop | Pop the latest entry |
| push | Push the entry in the stack |
| Print | Print the text |
| If | Start of if statement |
| else if | Start else if part |
| Endif | End of if block |
| endeLse | End of else block |
| Loop | Beginning of loop |
| Endloop | End of loop |
| @Reg1 | Address of register 1 |
| \_GReturn | Global Return variable |
| @Pointerstack | Address of the pointer stack |
| currentline | Current control line |
| goto | Reset the programme control line |
| ADD | Perform Addition |
| SUM | Perform Subtraction |
| MUL | Perform Multiplication |
| DIV | Perform Division |
| MOD | Performs modulus |
| GT | > logical operation |
| LT | < logical operation |
| EQ | == logical operation |
| NQ | != logical operation |
| GTEQ | >= logical operation |
| LTEQ | <= logical operation |
| OR | || logical operation |
| AND | && logical operation |

Complex function calls and expression:

The below is example of complex function calls and expression supported

by our language

*class Class1*

*int a*

*func main ()*

*::a = 5*

*print("Output : " + (square(square(::a)) + 3\*5))*

*print("Global Variable 'a' value : " + ::a)*

*end*

*func square(int a) returns int*

*a = a\*a*

*return a*

*end*

*end*

In this example, the program is having complex expression and function call. Here we have multiple function call and arithmetic operation and usage of variables and literals in a single line. The compiler is able to differentiate identifier having same name with respect to scope.

The grammar breaks the complex expression into simpler modules by the usage for temporary variables.

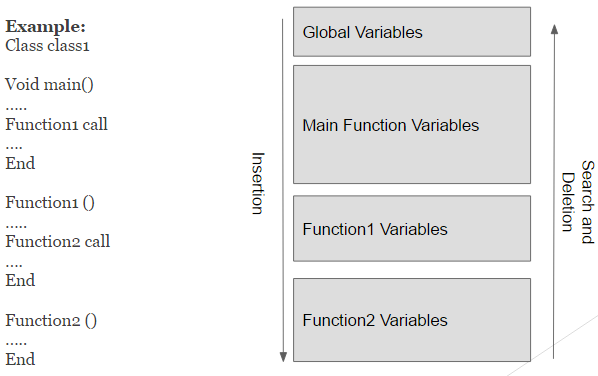
During a function call the the argument that is to be passed is set into a register and the current line number is pushed into a stacked named Pointerstack. The pointer stack is used as the return point after any function execution. Then the control line is jumped to the activation record of the desired function. After that function execution the control line is again shifted back to the calling environment by popping out the line number from pointerstack.

RSS Runtime:

* Runtime is written in Java
* .rssc intermediate file is given as input which generates the final output.
* The intermediate language is read line by line.
* Each line of intermediate code is divided into lexical tokens.
* The first token in a line determines the actions to be performed.
* Based on it, appropriate function call is made.
* Symbol table is used for value storage.

Symbol Table:

The below is the example of our symbol table.



We have used a stack based symbol table for storage of values and scope handling.

In the above example, we have main function calling Function1 which in turn calls function2. In our symbol table first the global variables gets pushed first then the main function variables followed by function 1 and then followed by function2. When the function2 finishes execution, all the new entry made by the function2 in the symbol table gets popped out. Now this approach has helped us in maintaining scope of a variable.

Challenges:

* Learning Antlr4 tool was one of the major challenge we faced.
* Another major challenge was to build the entire runtime from scratch using stack based approach.
* Implementation of function activation and argument handling
* Implementation of arrays and handling recursive function in runtime was another major hurdle.

References:

* <http://www.antlr.org/>
* <http://stackoverflow.com/questions/tagged/antlr>
* <https://www.javacodegeeks.com/2012/04/antlr-tutorial-hello-word.html>
* <http://www.oursland.net/tutorials/antlr/AntlrEclipse.html>
* <http://www.antlr2.org/doc/sor.html>