SER 502 PROJECT

MADS

Team 12

Madhu Madhavan

Ashwin Srinivasan

Deepti Paul

Srinivasan Sundar

Overview

- Introduction
- Grammar of the Language
- Tools Used over the course of the project
- Lexical Analyser and Parser
- Intermediate Code Generation
- Runtime
- Sample Programs

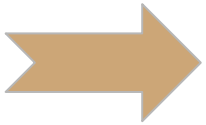
INTRODUCTION

- Designed Language Name: **MADS**
- Tools:
  - Compiler: Java SE 1.8 and ANTLR 4.8
  - Runtime: Python 3.7
- Basic Features:
  - Supports data types: integer, float, boolean and string
  - Supports arithmetic operations: addition, subtraction, Multiplication, Division
  - Supports traditional if-then-else statement, for loops, while loops
  - Supports standard output: print statement
  - Extension: .mads for source code and .imc for intermediate code

GRAMMAR

Original grammar rules:

Arithmetic Expression:
Expr ::= Expr '+' Expr
| Expr '-' Expr
| Expr '*' Expr
| Expr '/' Expr
| Expr '%' Expr
| (Expr)
| Identifier
| Digit



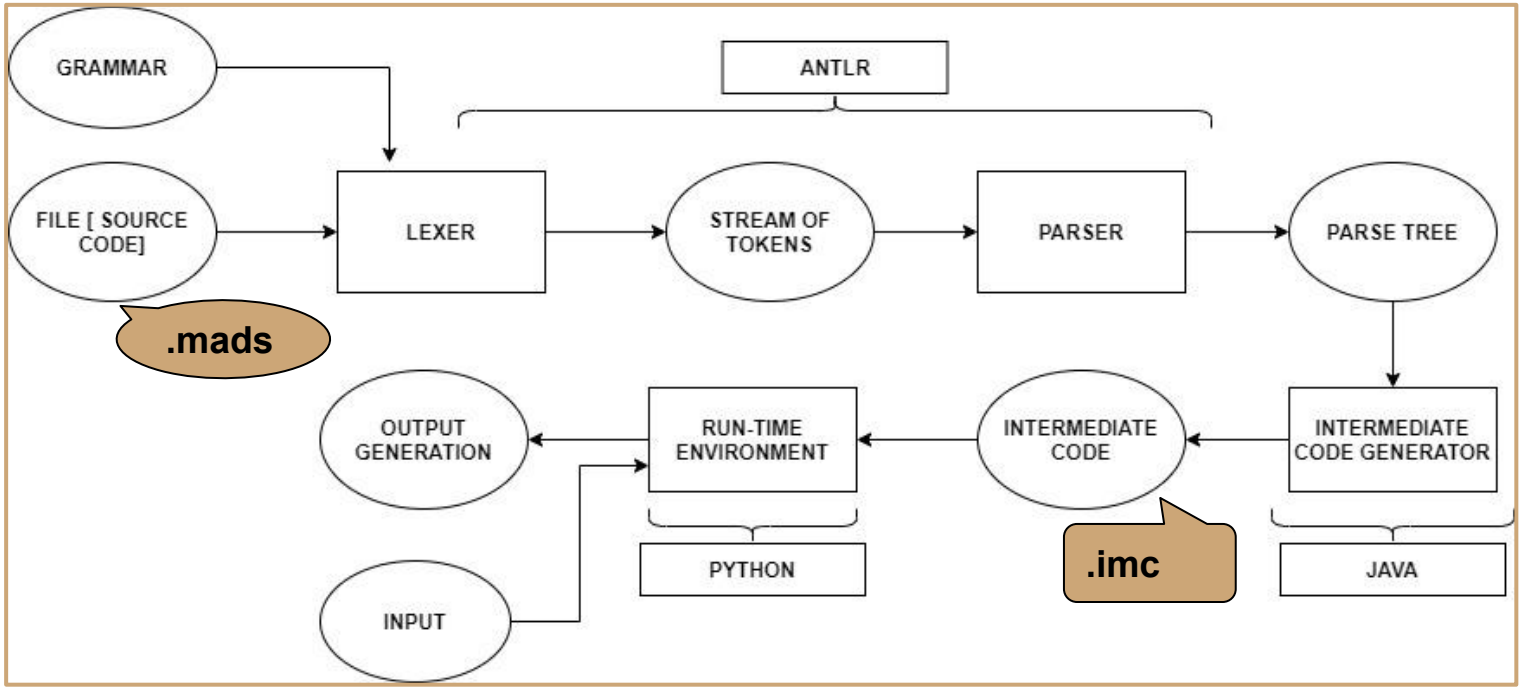
.g4 file for ANTLR input

```
expr : expr Plus expr_term #addExpression
| expr Minus expr_term #subExpression
| expr_term #termExpression
;

expr_term : expr_term Star expr_fact #mulExpression
| expr_term Div expr_fact #divExpression
| expr_term Mod expr_fact #modExpression
| expr_fact #factExpression
;

expr_fact : LeftParen expr RightParen #bracketExpression
| varName = Identifier #identifierExpression
| num = DigitSequence #numExpression
| floatNum = FractionalSequence #floatExpression
;
```

TOOLS USED: ANTLR v 4.8

- *“ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files.”*
- Grammar is written as .g4 file for ANTLR input.
- *ANTLR is used for token generation.*
- *ANTLR builds lexer and parser by translating the grammar.*
- *ANTLR helps to generate the parse tree. From a grammar, ANTLR generates a parser that can build and walk parse trees.*

SYSTEM DESIGN - Workflow of MADS



LEXICAL ANALYSER and PARSER

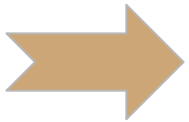
- **Lexer**
  - ANTLR reads the input file
  - Divides the input given into tokens based on the defined grammar.
- **Parser**
  - The parser takes a stream of tokens from the Lexer as the input and generates the parse tree.
  - ANTLR generates a parser that can build and walk parse trees. The generated parse tree is provided as input to intermediate code generation.

PARSER

Consider an example:

Code:

```
main() {
    int a = 5 ;
    int b = 2 ;
    int c ;
    c = a + b ;
}
```

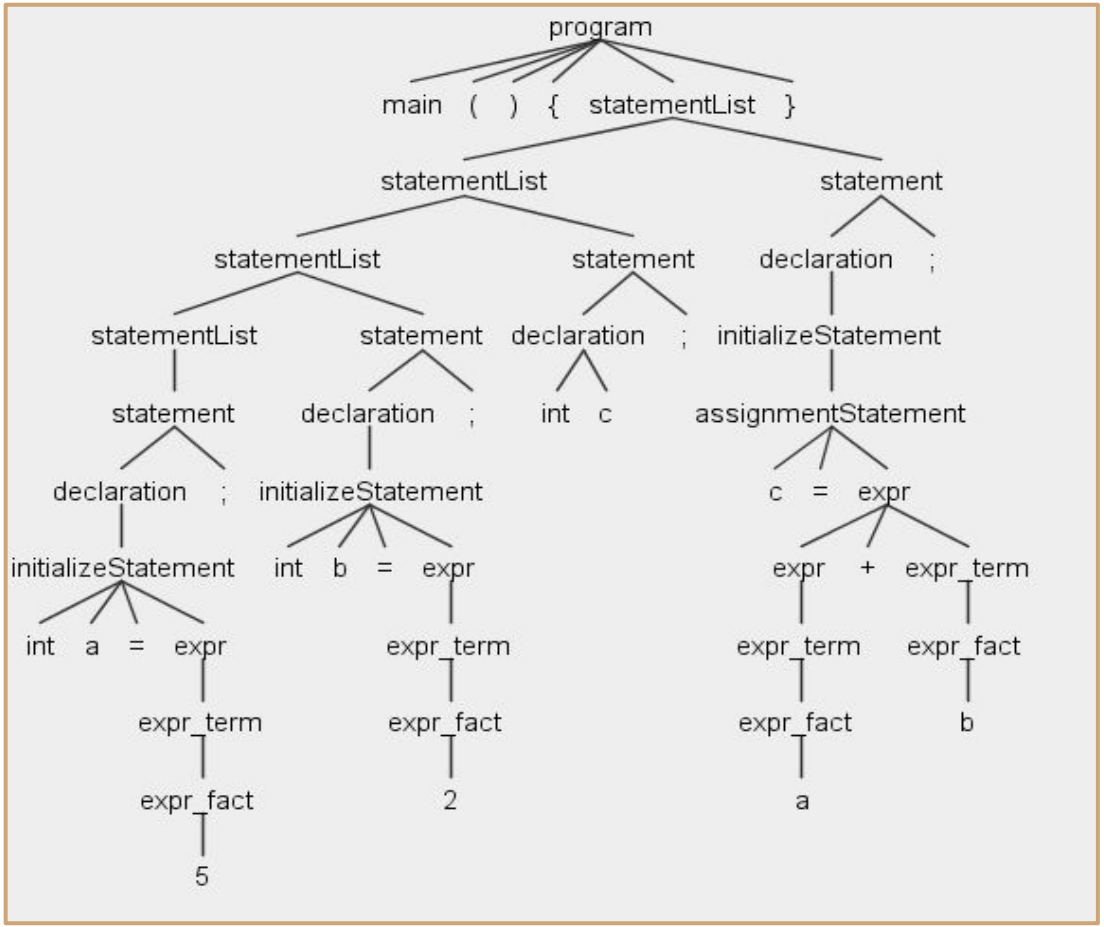


Parse Tree for the given program:

(program main ( ) { (statementList (statementList (statementList (statementList (statement (declaration (initializeStatement int a = (expr (expr_term (expr_fact 5))))) ;)) (statement (declaration (initializeStatement int b = (expr (expr_term (expr_fact 2))))) ;)) (statement (declaration int c) ;)) (statement (declaration (initializeStatement (assignmentStatement c = (expr (expr (expr_term (expr_fact a))) + (expr_term (expr_fact b)))))) ;)) })

PARSE TREE

- Syntactic structure of the program
- Each node in the parse tree is either a Non Terminal or Terminal.
- Parse tree is provided as input to intermediate code generation.



INTERMEDIATE CODE GENERATION

- Intermediate code generation written in Java.
- Intermediate code is generated by a custom listener class written by us in java which uses listener class generated by ANTLR.
- The custom listener class is implemented in IntermediateCodeGenerator.java
- Data structure used is HashMap which is used as a lookup table for data type and identifier pair.

INTERMEDIATE CODE FORMAT

KEYWORDS:

START/ END	→	Start and end of program
DECL	→	Declaration of variable
ASGN	→	Assign value to a variable
PULL	→	To load the variable
STORE	→	To store the variable
ADD/SUB/MUL/DIV/MOD	→	Arithmetic Operations
SML/SMLEQL/GTR/GTREQL/EQL/NOTEQL	→	Relational Operations
NOT/AND/OR	→	Logical Operations (not, and, or)
IFLOOP/ELSE/ENDIF	→	If else loop
CNDT / CNDTEND	→	Condition statement
WHILE/ENDWHILE	→	While loop
FORLOOP/ENDFOR	→	For loop
PRINT	→	Prints the value
INT/FLOAT/BOOL/STRING	→	DataType of a variable

INTERMEDIATE CODE GENERATION SAMPLE

Code to check whether check palindrome or not: → Intermediate Code generated:

```
main(){
    int n= 1001;
    int reversedN = 0;
    int remainder;
    int originalN;
    print("Enter an integer: ");
    originalN = n;

    while (n != 0) {
        remainder = n % 10;
        reversedN = reversedN * 10 + remainder;
        n = n / 10;
    }

    if (originalN == reversedN){
        print(originalN);
        print("is a palindrome.");
    }else{
        print(originalN);
        print("is not a palindrome.");
    }
}
```

```
1  START
2  DECL INT n
3  NUM 1001
4  STORE n
5  DECL INT reversedN
6  NUM 0
7  STORE reversedN
8  DECL INT remainder
9  DECL INT originalN
10 PRINT "Enter an integer: "
11 ASGN originalN n
12 WHILE
13 CNDT
14 PULL n
15 NUM 0
16 NOTEQL
17 CNDTEND
18 PULL n
19 NUM 10
20 MOD
21 STORE remainder
22 PULL reversedN
23 NUM 10
24 MUL
25 PULL remainder
26 ADD
27 STORE reversedN
28 PULL n
29 NUM 10
30 DIV
31 STORE n
32 ENDWHILE
33 IFLOOP
34 CNDT
35 PULL originalN
36 PULL reversedN
37 EQL
38 CNDTEND
39 PRINT originalN
40 PRINT "is a palindrome."
41 ELSE
42 PRINT originalN
43 PRINT "is not a palindrome."
44 ENDIF
45 END
```

RUNTIME

- Runtime is built using python 3.7
- Runtime uses data structures:
  - Stack: used list to create python stack which helps in expression evaluation.
  - Dictionaries: Two dictionaries are used. One for keeping track of data type and identifier pair; other for storing identifier and value pair.

Executing Compiler and Runtime

SAMPLE PROGRAM

Source Code (.mads)

Output

THANK YOU !!!