CS5363 Programming Languages and Compilers Project Report – Spring 2016

Project: Compiler for TL Language

Instructor: Xiaoyin Wang

Student Name: Manju Priya Hari Krishnan/dat986

1. Purpose

The project is the design of a compiler for TL language as specified in the TL 15.0 specification. The project is developed as part of the Programming Languages and Compilers Course during Spring 2016.

2. Architecture

2.1 Scanner

The Scanner takes TL program as the input, splits the program into valid tokens, and prints if any lexical errors exists. Output of this phase is the .tok file which consists of the tokens.

2.2Parser

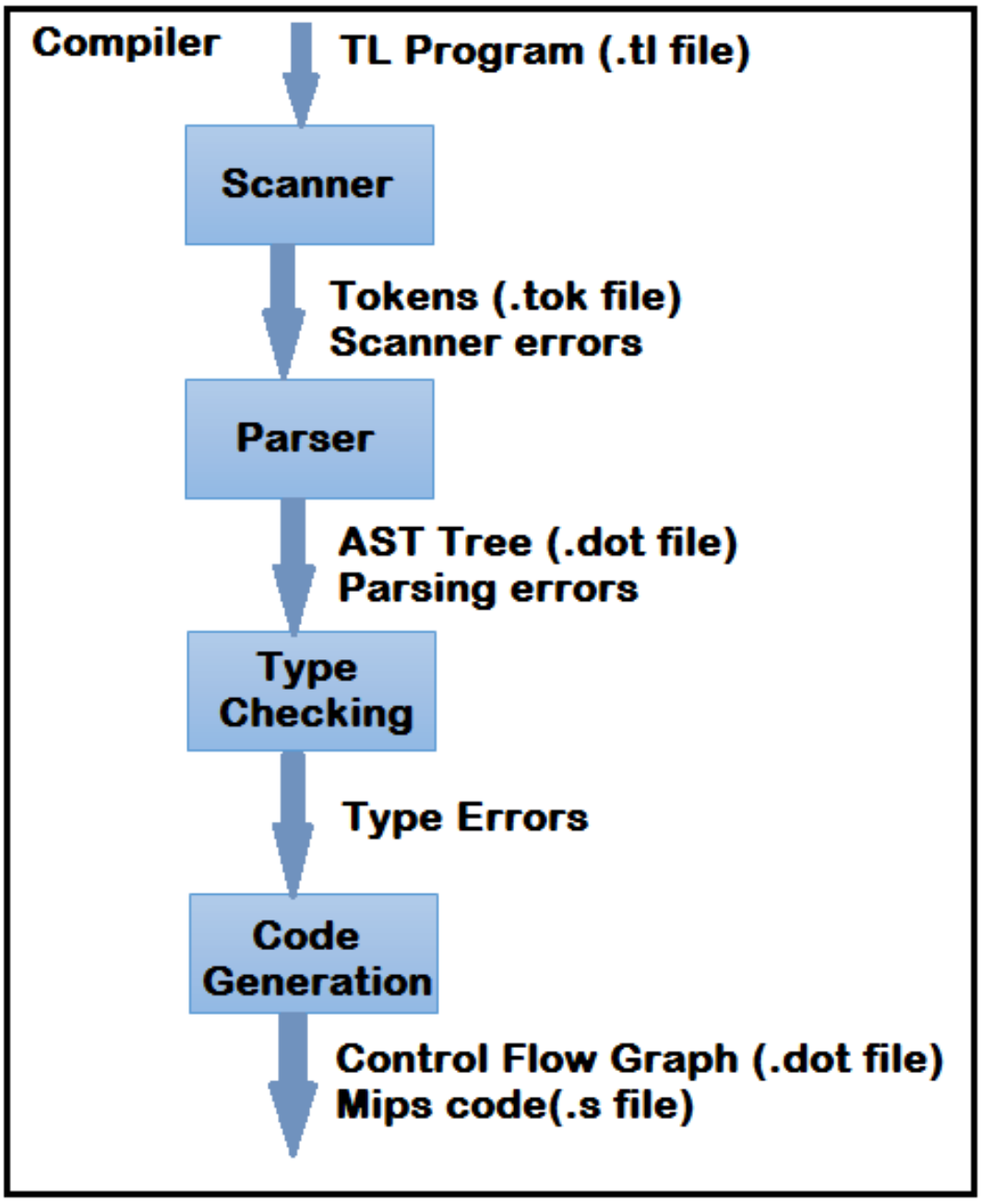
Parser takes the tokens which was generated by the scanner as the input and outputs an AST tree and prints if any parsing error occurs. The parsing is checked based on the BNF grammar specified in TL 15.0 specification. Output of this phase is .dot file which consists of the program to generate the AST.

2.3Type Checking

This phase checks if there is any type errors in the programs. If any type error in the program will be represented in the AST tree with a red color.

2.4Code generation

This phase takes the AST tree as the input and generates the MIPS code for it. This phase outputs two files. First one is the CFG which explains with the instructions to be executed and their flow of execution. The Second one is the .s file which contains the MIPS program.

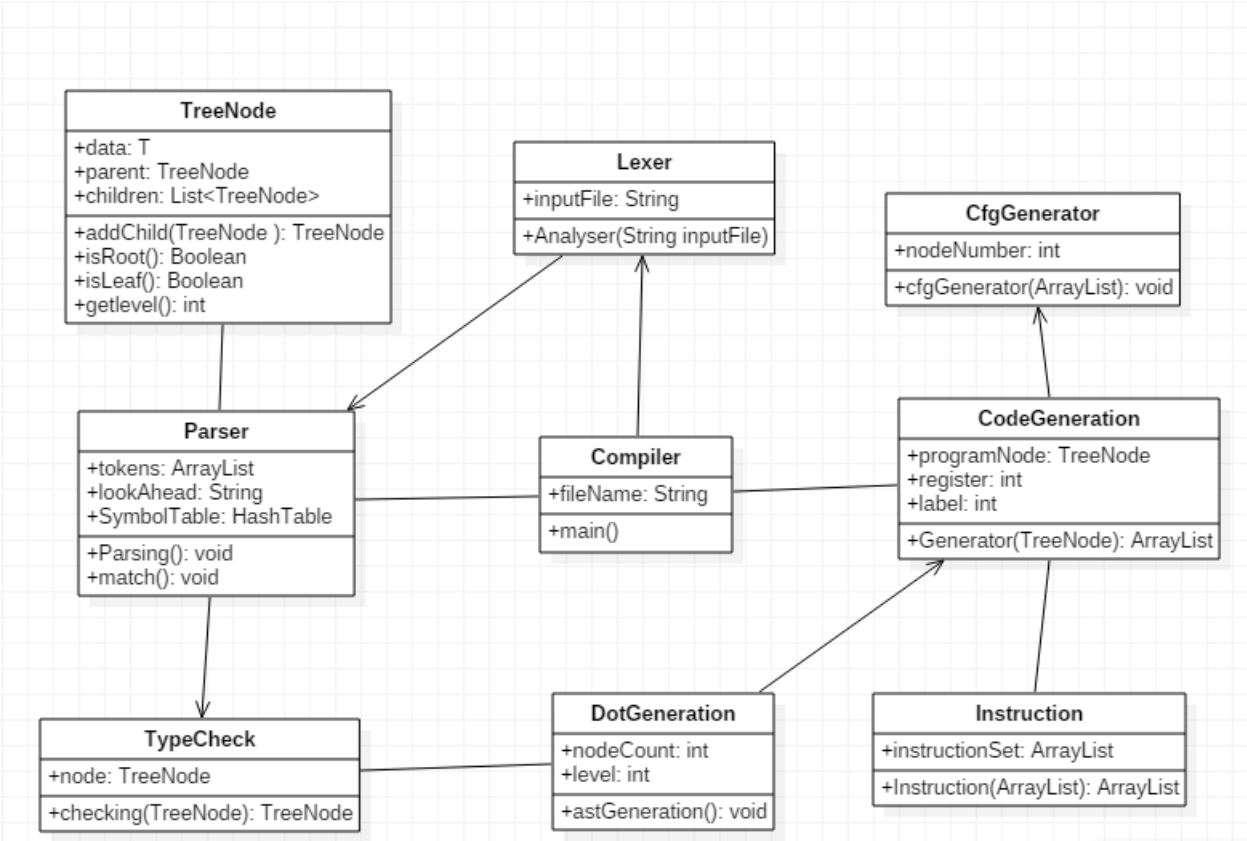


3. Technologies used:

The overall system is coded in JAVA Programing Language using Eclipse IDE. The other tools like Graphviz and QT simulator where used to check the AST, CFG and execution of the MIPS instructions.

4. Class Diagram:

The overall class diagram of the whole project is shown below:



4.1 Compiler

The Compiler is main class which contains the main method. This method takes the file name as the input and calls the classes which are the phases of compilers.

4.2 Lexer

Lexer class is responsible for the Scanner phase of the compiler. This takes the file name as the input from the main class, and generates the .tok file which contains the various tokens in the program. It also outputs the lexical errors if any exists.

4.3 Parser

Parser class takes the responsibility of Parser phase of the compiler. This takes the .tl files which is returned by the Lexer class as the input and generates the AST tree, which is given by the .dot file. It also outputs in the console if any parsing error occurs.

4.4 TreeNode

TreeNode class maintains the tree data structure which is used to represent the AST tree. It also contains methods for various operations to the tree like adding a child, traversing the tree and getting the level of a node.

4.5 TypeCheck

TypeCheck class is responsible for checking the type errors in the program. This class takes the AST tree generated by the Parser class and outputs the tree with different color node if any error exists.

4.6 DotGeneration

DotGeneration class generates the .dot file taking the AST tree as the input. Here the whole tree is traversed using the method in the TreeNode class to generate the .dot file.

4.7 CodeGeneration

CodeGeneration class takes the AST tree generated by the Parser class as the input. It traverses the tree and generates a 3-address code which is stored in the ArrayList. ArrayList is used as a data structure to store the set of instructions.

4.8 Instruction

Instruction class takes the values for each instruction and generates the instruction and adds all the instruction to a ArrayList. This data structure will represent a 2-dimensional array which is generated using ArrayList of ArrayList in the program.

4.9 CfgGeneration

CfgGeneration class takes the ArrayList of ArrayList as the input and generates a .dot file to represent the Control Flow Graph of the progam. This Control Flow Graph shows the instructions to be executed and the sequence in which they will be executed.

5. Input and Output Samples

5.1 Scanner Phase:

Input: (TL program)

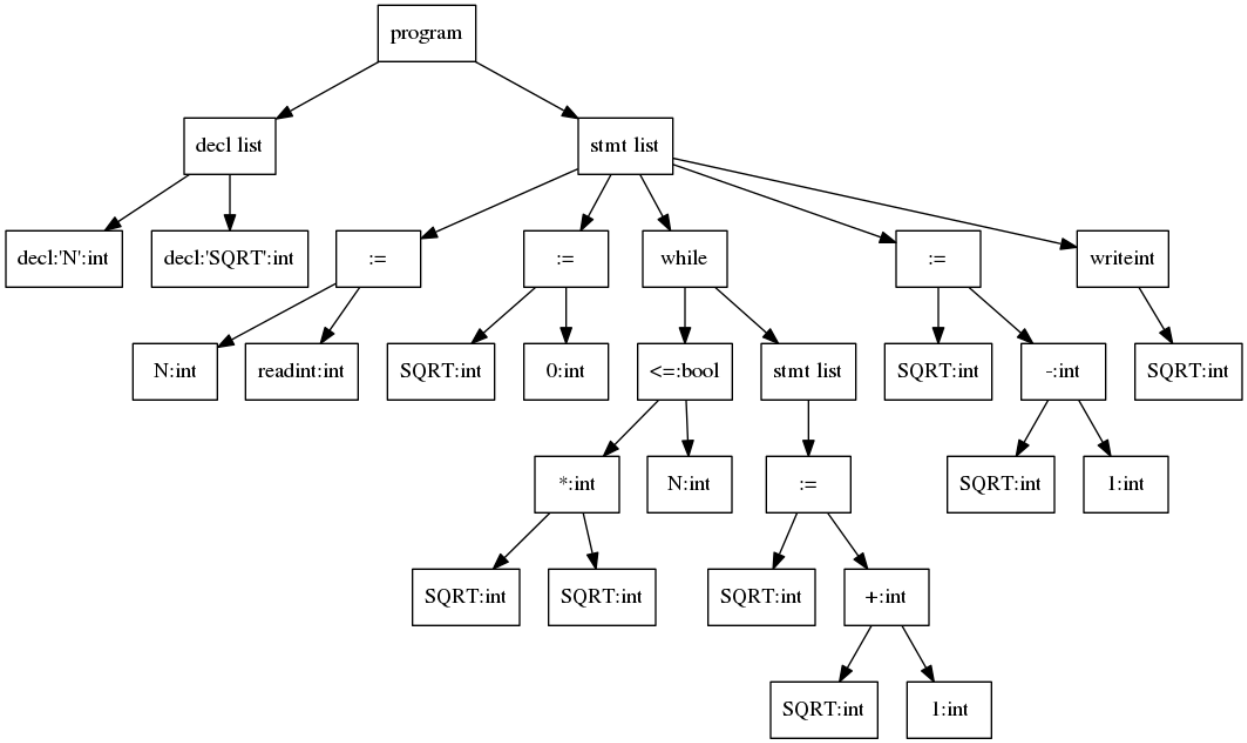
```
program
  var N as int ;
  var SQRT as int ;
begin
  N := readint ;
  SQRT := 0 ;

  while SQRT * SQRT <= N do
    SQRT := SQRT + 1 ;
  end ;

  SQRT := SQRT - 1 ;

  writeint SQRT ;

end
```

Output:

```
PROGRAM
VAR
ident(N)
AS
INT
SC
VAR
ident(SQRT)
AS
INT
SC
BEGIN
ident(N)
ASGN
READINT
SC
ident(SQRT)
ASGN
num(0)
SC
WHILE
ident(SQRT)
MULTIPLICATIVE(*)
ident(SQRT)
COMPARE(<=)
ident(N)
DO
ident(SQRT)
ASGN
ident(SQRT)
ADDITIVE(+)
num(1)
SC
END
SC
ident(SQRT)
ASGN
ident(SQRT)
ADDITIVE(-)
num(1)
SC
WRITEINT
ident(SQRT)
SC
END
```

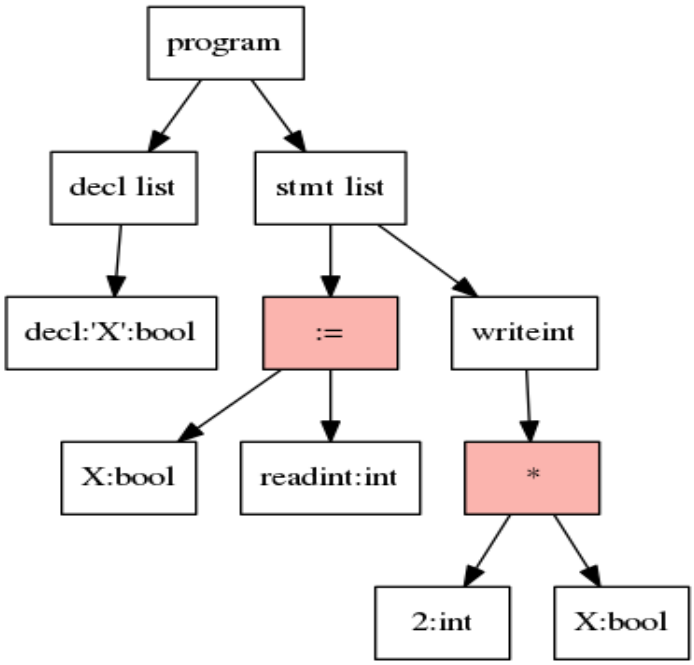
5.2 Parser:

Ouput: (AST Tree)



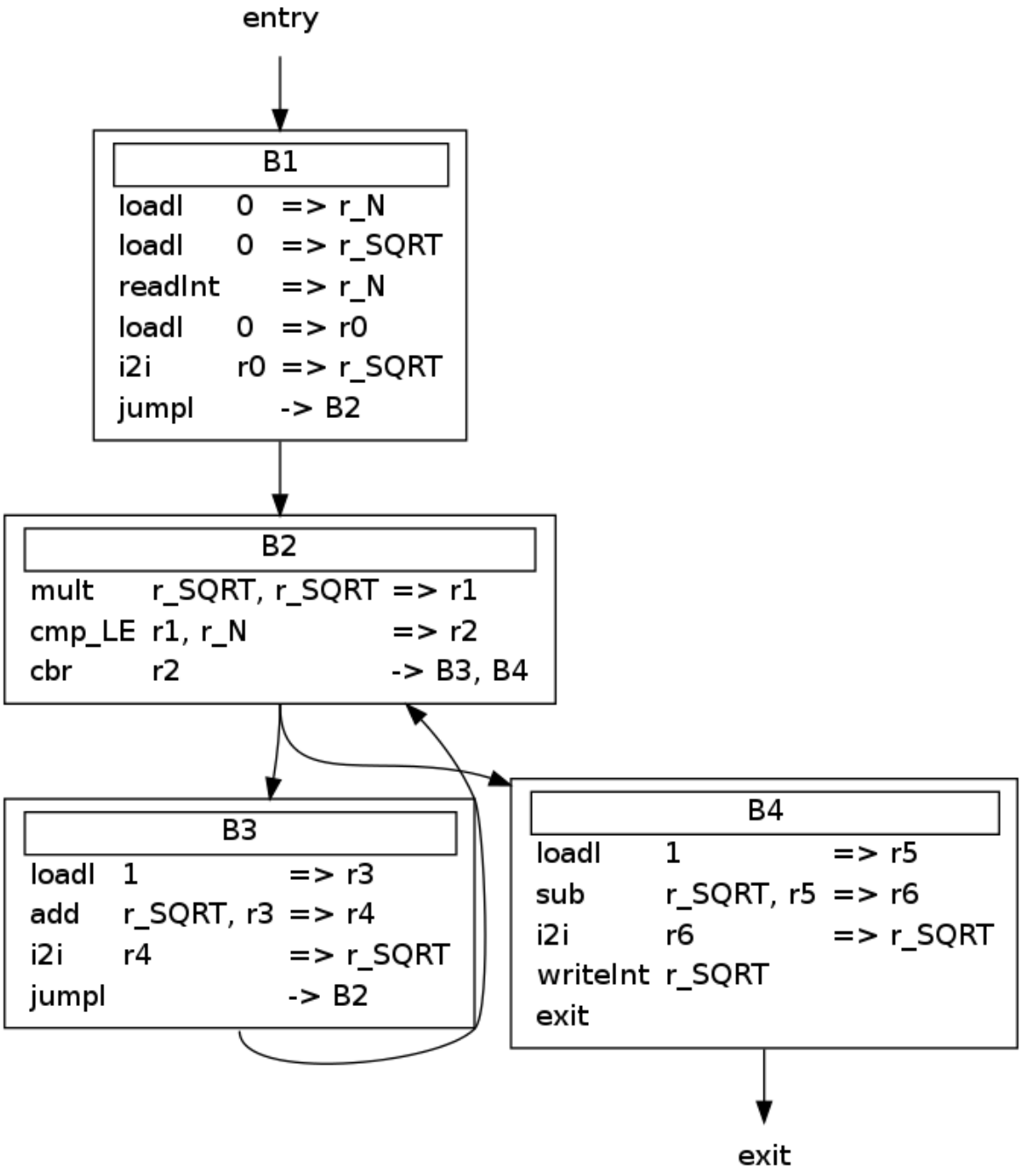
5.3 Type Checking:

Output: (AST with errors)



5.4 Code Generation

Output: (Control Flow Graph)



Output: (MIPS Code)

```
    .data
newline:    .asciiz "\n"
    .text
    .globl main
main:
    li $fp, 0x7ffffffc

B1:

    # loadI 0 => r_N
    li $t0, 0
    sw $t0, 0($fp)

    # loadI 0 => r_SQRT
    li $t0, 0
    sw $t0, -4($fp)

    # readInt  => r_N
    li $v0, 5
    syscall
    add $t0, $v0, $zero
    sw $t0, 0($fp)

    # loadI 0 => r0
    li $t0, 0
    sw $t0, -8($fp)

    # i2i r0 => r_SQRT
    lw $t1, -8($fp)
    add $t0, $t1, $zero
    sw $t0, -4($fp)

    # jumpI  -> B2
    j B2

B2:

    # mult r_SQRT, r_SQRT => r1
    lw $t1, -4($fp)
    lw $t2, -4($fp)
    mul $t0, $t1, $t2
    sw $t0, -12($fp)

    # cmp_LE r1, r_N => r2
    lw $t1, -12($fp)
    lw $t2, 0($fp)
    sle $t0, $t1, $t2
    sw $t0, -16($fp)

    # cbr r2 -> B3, B4
    lw $t0, -16($fp)
    bne $t0, $zero, B3

L11:
    j B4

B3:

    # loadI 1 => r3
    li $t0, 1
    sw $t0, -20($fp)

    # add r_SQRT, r3 => r4
    lw $t1, -4($fp)
    lw $t2, -20($fp)
    addu $t0, $t1, $t2
    sw $t0, -24($fp)

    # add r_SQRT, r3 => r4
    lw $t1, -4($fp)
    lw $t2, -20($fp)
    addu $t0, $t1, $t2
    sw $t0, -24($fp)

    # i2i r4 => r_SQRT
    lw $t1, -24($fp)
    add $t0, $t1, $zero
    sw $t0, -4($fp)

    # jumpI  -> B2
    j B2

B4:

    # loadI 1 => r5
    li $t0, 1
    sw $t0, -28($fp)

    # sub r_SQRT, r5 => r6
    lw $t1, -4($fp)
    lw $t2, -28($fp)
    subu $t0, $t1, $t2
    sw $t0, -32($fp)

    # i2i r6 => r_SQRT
    lw $t1, -32($fp)
    add $t0, $t1, $zero
    sw $t0, -4($fp)
```