Project 3
Implementation of a Code Generator
Due Wednesday, June 23, 2021

1. Problem:

In this assignment you are requested to use ANTLR to implement a code generator for MIPS processors using the techniques introduced for intermediate code generation. You can use the SPIM simulator, which simulates a MIPS processor, to verify the correctness of your code generator. The SPIM simulator and related information can be obtained from http://www.cs.wisc.edu/~larus/spim.html.

To simplify register allocation, you may maintain an integer counter of 10 denoting general registers $t0~$t9, and implement a function getReg() to request a register and a function putReg() to return a register. You may assume that 10 registers are enough to evaluate every expression in the program. You should return all the registers you use back after evaluating an expression. You may also maintain an integer counter for labels and implement a function newLabel() to request a new label. Since there is no limitation on the number of labels, you do not need to return a label.

The Read, Write, and Exit statements will be implemented by the system calls read_int, print_int, and exit of the SPIM simulator. The code generator reads input from stdin, writes output to stdout, and writes errors to stderr.

You can follow the following steps:

1. Edit the grammar Cactus.g to add actions to parser rules.

```
// The attribute grammar for Cactus language
grammar Cactus;

// Parser rules
program : PROGRAM ID BEGIN
            { System.out.println("\t" + ".data");}
            declarations
            {
                  System.out.println("\t" + ".text");
                  System.out.println("main:");
            }
            statements END
;
...
// lexer rules
ELSE : 'else'
 ...
COMMENT : ...
```

2. Use the ANTLR tool to generate the scanner, parser, and semantic analyzer java code.

```
$antlr4 Cactus.g4
```

3. Compile the generated java code.

```
$javac Cactus*.java
```

4. Use the ANTLR tool to execute the code generator.

```
$grun Cactus program < input_file >! output_file
```

A sample **Cactus** program is given as follows:

```
// A program to sum 1 to n
Program sum Begin
    Var  n
    Var  s

    Read n
    If n < 0 Then
        Write -1
        Exit
    Else
        Set s = 0
    EndIf
```

```
        While n > 0 Do
            Set s = s + n
            Set n = n – 1
        EndWhile
        Write s
    End
```

The stdout output could be as follows:

```
    .data
n:                  # Var n
    .word 0
s:                  # Var s
    .word 0
    .text
main:
    li $v0, 5       # Read n
    syscall
    la $t0, n
    sw $v0, 0($t0)
    la $t0, n       # If n < 0
    lw $t0, 0($t0)
    li $t1, 0
    blt $t0, $t1, L1
    b L2
L1:                 # Then
    li $t0, 1       # Write -1
    neg $t0, $t0
    move $a0, $t0
    li $v0, 1
    syscall
    li $v0, 10      # Exit
    syscall
    b L3
L2:                 # Else
    li $t0, 0       # Set s = 0
    la $t1, s
    sw $t0, 0($t1)
```

```
L3:                     # EndIf
L4:                     # While n > 0
    la $t0, n
    lw $t0, 0($t0)
    li $t1, 0
    bgt $t0, $t1, L5
    b L6
L5:                     # Do
    la $t0, s           # Set s = s + n
    lw $t0, 0($t0)
    la $t1, n
    lw $t1, 0($t1)
    add $t0, $t0, $t1
    la $t1, s
    sw $t0, 0($t1)
    la $t0, n           # Set n = n - 1
    lw $t0, 0($t0)
    li $t1, 1
    sub $t0, $t0, $t1
    la $t1, n
    sw $t0, 0($t1)
    b L4
L6:                     # EndWhile
    la $t0, s           # Write s
    lw $t0, 0($t0)
    move $a0, $t0
    li $v0, 1
    syscall
```

2. Handing in your program

To turn in the assignment, upload a compressed file proj3 containing Cactus.g4, Cactus.tokens, Cactus*.java, and Cactus*.class to eCourse2 site.

3. Grading

The grading is based on the correctness of your program. The correctness will be tested by a number of test cases designed by the instructor and teaching assistants.

It is best to incrementally generate your program so that you always have a partially-correct working program.