Amirkabir University of Technology
(Tehran Polytechnic)

Department of Computer Engineering and Information Technology

Course Project

Principles of Compiler Design
Phase one
Intermediate Code Generation

By
Nima Davari

Professor
Dr. MohammadReza Razzazi

Fall 2019

Overall description

Using the parse tree constructed in the second phase, an intermediate code representing the input data is to be generated and compiled as C code. You are allowed to use the following instructions in your intermediate code.

Instruction Type	Syntax	Notes
Assignment	$T_1 = T_2;$	-
	$T_1 = \text{op } T_2;$	$\text{op} \in \{! \sim -\}$
	$T_1 = T_2 \text{ op } T_3;$	$\text{op}_{\text{Arithmetic}} \in \{+ \ - * / \% \ \& \ | \ \ll \gg\}$ $\text{op}_{\text{Boolean}} \in \{\&\& \ ||\}$
	The variables at the right hand side may be replaced by raw constants. (i.e. 24 or true)	
Goto	goto L;	-
Label	L:	-
If-Goto	$\text{if}(T_1 \text{ op } T_2)$ goto L;	$\text{op} \in \left\{ \begin{matrix} > \\ < \\ >= \\ <= \\ == \\ != \end{matrix} \right\}$ Numbers may replace the variables.
	if(T) goto L;	Numbers or Boolean constants may replace the variable.
Pointer-Based Operations	$T_1 = {}^*T_2;$	-
	${}^*T_1 = T_2;$	The right hand side variable may be replaced by a constant.

Arithmetic Expression Evaluation

Each arithmetic expression needs to be evaluated and its final value should be stored in the stack for later use.

Code must be generated for each grammar rule specified below.

exp → INTEGER

exp → REAL

exp → lvalue

unary_operation → − exp

unary_operation → ~ exp

bitwise_operation → exp & exp

bitwise_operation → exp | exp

binary_operation → exp + exp

binary_operation → exp − exp

binary_operation → exp ∗ exp

binary_operation → exp / exp

binary_operation → exp % exp

binary_operation → exp << exp

binary_operation → exp >> exp

Boolean Expression Evaluation

Each Boolean expression needs to be evaluated and should consider leading the program to the true/false sections of the corresponding statement.

Code must be generated for each grammar rule specified below.

exp → TRUE

exp → FALSE

exp → lvalue

unary_operation → ! exp

logical_operation → exp && exp

logical_operation → exp || exp

comparison_operation → exp > exp

comparison_operation → exp < exp

comparison_operation → exp >= exp

comparison_operation → exp <= exp

comparison_operation → exp == exp

comparison_operation → exp != exp

Assignment

Arithmetic and Boolean assignments need to be handled, using the following rule.

assignment → lvalue = exp ;

Control Flow Statement

The code for the following rules must be generated.

statement → if

statement → for

statement → while

Function Call

Function calls are handled through the following rules.

func_body → ID (formal_arguments) block

statement → return

function_call → ID function_call_body