Complier Lab(CSE4112)

Assignment#3: Generating Three-address Codes

Assignment details

This is the first group assignment. In this assignment, you are to generate three address instructions for code written in SubC. Following is the grammar for SubC. This grammar is slightly modified version of the grammar specified in assignment#2. Specifically, we have added if-then-else and while statements. These statements have the same semantic meaning as in C language. List of possible three address instructions and description of a syntax directed translator for generating three address codes for an expression grammar can be found in the resources tab in piazza (Lecture Note/Intermediate Code Generation). Also see section 6.2 and 6.4 of the textbook (Dragon book).

SubC grammar

Stmt_list → Stmt | Stmt_list Stmt
Stmt → Variable **assignop** Expression **;**
| if **(** Expression **)** **{** Stmt_list **}** else **{** Stmt_list **}**
| while **(** Expression **)** **{** Stmt_list **}**
Variable → **id**
Expression → Simple_expression | Simple_expression **relop** Simple_expression
Simple_expression → Term | Simple_expression **addop** Term
Term → Factor | Term **mulop** Factor
Factor → **id** | **num** | **(** Expression **)** | **!** Factor

Tokens/Terminal symbols of this grammar are in BOLD font.

Implementation

You are to write (F)LEX and YACC/BISON specification files (e.g., subc.l and subc.y). You may use the LEX and YACC specification files you created for assignment #2. Here are some implementation issues and ideas (1) Maintain a symbol table for each id. Initially the symbol table should contain the keywords. This is useful because whenever you detect an id you can then check the symbol table to see whether the lexeme matched is a keyword or not, (2) Use the union data structure available in YACC to specify different types of attributes for the variables in the grammar, (3) Use quaduples/quads to store the three address codes.

Sample input#1

```
b = b + 1;
if(b > a){b = 2 + c;} else{b = d - 2;}
a = c + b * d;
```

Sample output#1

```
t1 = 1
t2 = b + t1
if t2 > a goto label1
t3 = 2
t4 = t3 + c
goto label2
label1:
t5 = 2
t6 = d - t5
label2:
t7 = b * d
t8 = c + t7
a = t8
```

Sample input#2

```
while ( a+3 ) { a = a - 1; }
```

Sample output#2

```
label1: t1 = 3
t2 = a + 3
if t2 = 0 goto label2
t3 = 1
t4 = a - t3
a = t4
goto label1
label2 :
```