Three Address Code Generation

What is a Three Address code?

Three address code is a sequence of statements of the general form **x = y op z** ,where x , y , z are names, constants and op stands for any operator such as fixed or floating point arithmetic operator or a logical operator on boolean valued data. Here three address refers to three addresses ie addresses of x, y and z.

Say for example we have a statement like a = b + c * d then we can make a three address code for it as follows:
t1 = c * d;
a = b+ t1;

Types of Three address statements:

There are different types of three address statements. Some of them are as follows :-

• Assignment statements. They are of the form **x := y op z** where op is a binary arithmetic   or logical operation

• Assignment Instructions. They are of the form **x := op y** where op is an unary operation like unary plus, unary minus shift etc....

• Copy statements. They are of the form **x := y** where the value of y is assigned to x

• Unconditional Jump **goto L**. The three address statement with label L is the next to be executed.

• Conditional Jumps such as **if x relop y goto L**. This instruction applies a relational operator (<,>,<=,>=) to x and y and executes the statement with label L if the conditional statement is satisfied. Else the statement following if x relop y goto L is executed

• **param x** and **call p,n** for procedure calls and return y where y representing a returned value (optional).Three Address statements for it are as follows.

```
param x1
param x2
param x3
  .
  .
param xn
call p,n
```

generated as a part of the three address code for call of the procedure p(x1,x2,x3,....xn)
where n are the number of variables being sent to the procedure

Structure for Three Address Code:

It is a quadruple of an operator, arg1 , arg2 and a result or it is a triple of an operator , arg1 and arg2. In triple form arg2 is generally a pointer to the triple structure.

Three address code generation:

Suppose we have the grammar

```
S ->id := E
E -> E+E
E -> E*E
```

E -> - E
E -> (E)
E -> id

Grammar Rule	***Action Statements***
S->id:=E	S.code :=E.code || gen(id.place":="E.place)

E->E+E	E.place=newtemp()
	E.code:=E1.code || E2.code || gen(E.place ":=" E1.place '+' E2.place)

E->E*E	E.place=newtemp()
	E.code:=E1.code || E2.code || gen(E.place ":=" E1.place '*' E2.place)

E-> -E	E.place=newtemp()
	E.code := E1.code || gen(E.place ':=" '-' E1.place)

E->(E)	E.place=E1.place
	E.code=E1.code

E->id	E.code=' '
	E.place=id.place

E1 refers to the first E that comes in the production .
E2 refers to the second E that comes in the production.

Here E.code stands for the three address code generated by E and E.place stands for the variable name

Swapandeep Singh
04CS1033