Assignment 5
Syntax Directed Translation

Three address code:

In three address code, each statement generally contains three addresses,two for the operands and one for result.

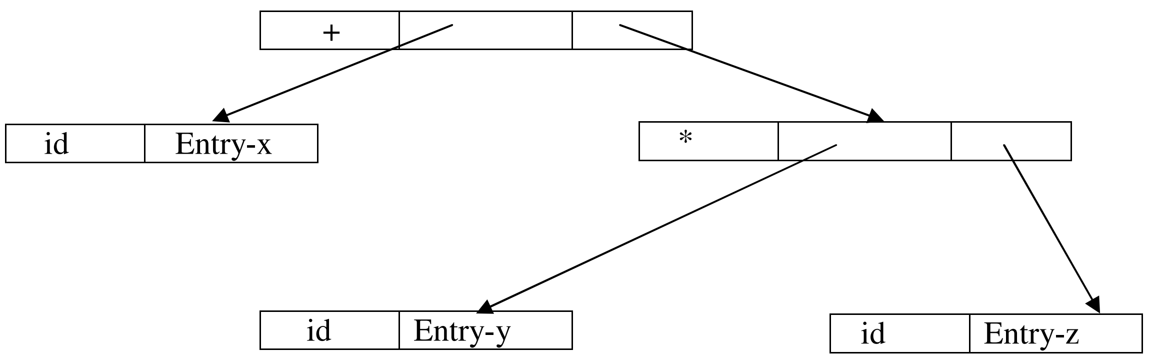
The three address code is a sequence of statements of the general from:

X:= y op z

Where x,y and z are names (variables) ,constants or temporary variables generated by a compiler.

Consider a source language expression like : x+y*z

Syntax tree:



Three address code:

t1:= y*z

t2:= x+t1

Here t1 and t2 are temporary names which are generated by the compiler.
We can say that three address code is linearized representation of a syntax tree.

The advantages of using temporary names for intermediate results is that the three address code can be easily rearranged.
Operators Examples

Arithmatic +,-,*,/, %
Logical AND &&,
Logical OR ||
Logical Negation ~
Bitwise AND &
Bitwise OR |
Shift Left <<
Shift Right >>
Relational <,>,<=,>=,==,!=
Assignment =
Conditional ?:

Types of Three Address statements

Three address statements are akin to assembly code statements can have symbolic labels and there are statements for flow of control. A symbolic label represents the index of the three address statement in the array holding intermediate code. Actual indices can be substituted for the labels either by a separate pass or by using back patching.

1: Assignment statement of the form x:=y op z

where

op is a binary arithmetic or logical operation

2: Assignment statement of the form x:=op y

where

op is an unary operation

Unary operation include unary minus, logical neagation, shift operators and conversion operators

For eg:

Convert a fix point no. (int) to a floating pt. no.(float)-typecasting

3: Copy statement of the form x:=y

where

value of y is assigned to x

4: The unconditional jump goto L

The three address statement with label L is a next to be executed.

5: The conditional jump such as :

If x relop y goto L

This instruction applies a relational operator to x,y and execute the statement with label L next ,if the relation is true.

If not , the next statement is a normal flow sequence is executed

6: a:=b[i] and a[i]:=b, indexed assignment statement r used to handle arrays

7:Address and pointer assignment statements r of the form

x:=&y   x:=*y   *z:=y

Implementation of Three address statements

A three address statements is an abstract form of intermediate code.

In a compiler, these statements can be implemented as records with fields for the operator and operands. Three such representations are qudrapules,triples, and indirect triples

Quadruple

A quadruple is a record structure having four fields as follows

operator	operand1	operand2	result

where operand1 and operand2 are two operands for the operator and the result field contains the result of the operation on operand1 and operand2.

eg: a:=b+c will be represented as : a = + b c a

The fields operand1,operand2 and result are pointers to symbol table entries, therefore temporary names must be entered into the symbol table.

Quadruples for an assignment statement
x := a+b*c+c*d*e

operator	Operand1	Operand2	result
*	b	c	T1
*	c	d	T2
*	T2	e	T3
+	a	T1	T4
+	T4	T3	T5
:=	T5		x

Quadruples for an assignment statement
a := b*-c + b*-c

Operator	Operand1	Operand2	Result
Uminus	c		T1
*	b	T1	T2
Uminus	c		T3
*	b	T3	T4
+	T2	T4	T5
:=	T 5		a

Quadraples properties:
1. Simple record structure for four fields
2. Easy to reorder
3. Explicit namespace management for temporaries

Triples:
To avoid entering temporary names into the symbol table, we might prefer to a temporary value by the position of the statement that computes it.
So three address statements can be represented by records with only three fields as follows:

Operator	Operand1	Operand2

The fields operand1 and operand2 are either pointers to the symbol table (for programmers defined names or constants) or pointers into the triple structure(for temporary values). Since three fields are used this intermediate code format is known as triples.
Triples for an assignment statement
X := a + b * c + c * d * e

	Operator	Operand1	Operand2
0	*	B	C
1	*	C	D
2	*	(1)	E
3	+	A	(0)
4	+	(3)	(2)
5	:=	X	(4)

Triples for an assignment statement

a := b * -c + b* -c

	Operator	Operand1	Operand2
0	uminus	c	
1	*	b	(0)
2	uminus	c	
3	*	b	(2)
4	+	(1)	(3)
5	:=	a	(4)

Triples properties:

1) It uses table index as implicit name for temporaries.
2) Requires only three fields in record
3) Harder to reorder

Triples for assignment statement

x[i] := y

	Operator	Operand1	Operand2
0	[]=	X	I
1	:=	(0)	Y

Triples for assignment statement

x := y[i]

	Operator	Operand1	Operand2
0	=[]	Y	I
1	:=	X	(0)

Indirect triples:

Indirect triples for assignment statement

X := a + b *c + c * d * e

	Statement
0	(14)
1	(15)
2	(16)
3	(17)
4	(18)
5	(19)

	Operator	Operand1	Operand2
14	*	b	c
15	*	c	d
16	*	(15)	e
17	+	a	(14)
18	+	(17)	(16)
19	:=	x	(18)

Indirect triples for assignment statement
A:=b*-c+b*-c

	Statement
0	(14)
1	(15)
2	(17)
3	(18)
4	(19)

	Operator	operand	Operand2
14	uminus	C	
15	*	B	(14)
16	Uminus	C	
17	*	B	(16)
18	+	(15)	(17)
19	:=	a	(18)

Indirect Triple properties
1. Uses array called statement to list pointers to triples.
2. Simplifies moving statements
3. More space than triples, same as quadruples for single statement
4. Implicit name space management for temperatures.

Quadruples are more useful in code optimization where statements are often moved around. if we move a statement computing x,the statements using x requires no change.

However,in the Triples notation,moving a statement that defines a temporary value requires us to change all references to that statement.This problem makes triples difficult to use in an optimizing compiler.

Indirect triples present no such problem. Reordering the statement list.sice pointers to temporary can move a statement list. Since pointers to tempory values refer to the operator,opreand1 and operand2 which are not changed. None of those pointers need to be changed. Indiect triples look very much like quadruples as far as their utility is concerned. However indirect triplescan save some place compared with qudrapules if the same temporary value is used more than once.

Syntax Directed Translation:

Syntax Directed Translation scheme to produce three address code for assignments

```
S→id:E                    { p:=lookup(id.entry)
                               If(p!= nil)then
                               Emit(p':=' E.place)
                            Else
                                Error
```

```
                                    }


E→E1+E2
                                     { E.place:=newtemp
                                        emit(E.place':=' E1.place+E2.place)
                                     }


E→E1*E2
                                     { E.place:=newtemp
                                        emit(E.place':=' E1.place*E2.place)
                                     }


E→-E
                                     { E.place:=newtemp
                                        emit(E.place':=' Uminus E1.place)
                                     }


E→(E1)                                        { E.place:=E1.place}



S→id                               { p:=lookup(id.entry)
                                           If(p!= nil)then
                                           E.place:=p)
                                       Else
                                            Error
                                    }
```

IMPLEMENTATION OF SYMBOL TABLE

The symbol table has been implemented using a structure. The format is as follows:

```
Struct symtable
{
        char sym[10];
        char type[10];
}symtab[100];
```

Here the symbol is stored in sym and its data type in type. The symbol table is used to store the details of the variables. Whenever a variable is found it is matched with the entries of the symbol table. If the variable in the declaration statement then it is checked whether it is already there in the symbol table , if yes then it is a redeclaration error else insert it in the symbol table. The symbol table is also used to detect type mismatch.