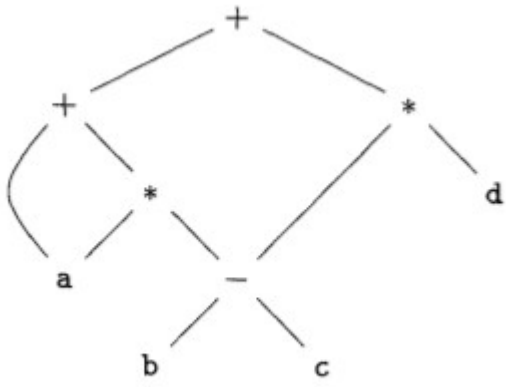
Three Address Code Generation

How to store?
How to represent?

Directed Acyclic Graphs for Expressions (DAG) :

Like the syntax tree for an expression, a DAG has leaves corresponding to operands and interior codes corresponding to operators. The difference is that a node N in a DAG has more than one parent if N represents a common subexpression; in a syntax tree, the tree for the common subexpression would be replicated as many times as the subexpression appears in the original expression.

Dag for the expression a + a * (b – c) + (b - c) * d



Representations of 3 address code

Three representations are called "quadruples," triples," and "indirect triples."

1. Quadrapules

A quadruple has four fields, which we call op, arg,, arg2,
and result. The op field contains an internal code for the operator. For instance, the three-address instruction x = y + x is represented by placing + in op, y in op1, z in op2 and x in result

```
t1 = minus c
t2 = b * t1
t3 = minus c
t4 = b * t3
t5 = t2 + t4
 a = t5
```

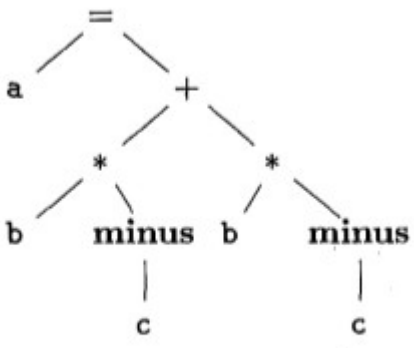
(a) Three-address code

	op	arg_1	arg_2	*result*
0	minus	c		t_1
1	*	b	t_1	t_2
2	minus	c		t_3
3	*	b	t_3	t_4
4	+	t_2	t_4	t_5
5	=	t_5		a
		...		

(b) Quadruples

2. Triples

A triple has only three fields, which we call op, op1, and op2.



(a) Syntax tree

	op	arg_1	arg_2
0	minus	c	
1	*	b	(0)
2	minus	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)
	...		

(b) Triples

3. Indirect Triples

Indirect triples consist of a listing of pointers to triples, rather than a listing of triples themselve

Common Three-Address Instruction Forms

1. Assignment

x = y op z

where op is a binary arithmetic or logical operation, and x, y, and z are addresses.

x = op y

where op is a unary operation.

2. Copy

x = y

where x is assigned the value of y.

3. Unconditional jump

goto L

where the three-address instruction with labelL is the next to be executed.

5. Conditional jumps

if x goto L

These instructions execute the instruction with label L next if x is true

if x reop y goto L

which apply a relational operator to x and y

6. Procedure calls

call p
return y

7. Indexed copy instructions

x = y[i]
x[i] = y

8. Address and pointer assignments

x=&y
x = *y
*x = y