compilen

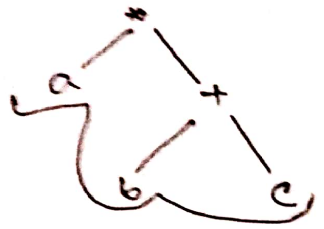
- AST, DAG, Three addresn Code (Quadtriples, Triples)

AST (Abstract syntax tree)

AST are more compact than a parse tree and can be easily used by compiler.

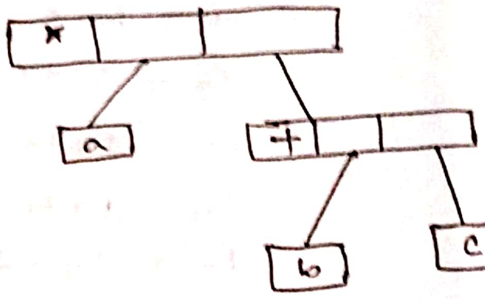
Example: a*b+c

Parse tree



AST

AST represented as:



Three address code

1. It is an intermediate code. It is used by optimizing complier.

2. In three address code, the given expression is broken down into several separate instructions. These instructions can easily translate into assembly language.

3. Each three address code instruction has at most three operands. It is a combination of assignment and a binary operator.

In TAC, there is at most one operator on the right side of an instruction.

Example; $x + y * 2$

$t_1 = y * 2$

$t_2 = x + t_1$

Example: $a + a * (b - c) + d * (b - c)$

$t_1 = b - c$

$t_2 = a * t_1$

$t_3 = a + t_2$

$t_4 = d * t_1$

$\boxed{t_5 = t_3 + t_4}$

There are two data structure for TAC.

1. Quadtriples 2. Triples

$\downarrow$

4 field
- operator
- source 1
- source 2
- destination

Example; Quadtriples

TAC: $a := - b * c + d$

$t_1 = - b$

$t_2 = c + d$

$t_3 = t_1 * t_2$

$a := t_3$

Quad triples

	operator	source1	source2	result
(0)	uniminus	b	-	t_1
(1)	+	c	d	t_2
(2)	*	t_1	t_2	t_3
(3)	:=	t_3	-	a

② Triples
⇓
3 fields → Operator, Source 1, Source 2

Note
* Need to take help of Quadtriples

Example: $a = -b * c + d$

TAC:

$t_1 = -b$
$t_2 = c + d$
$t_3 = t_1 * t_2$
$a := t_3$

Triples

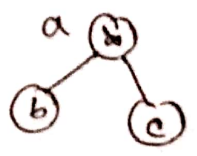
	Operator	Source 1	Source 2
(0)	−	b	−
(1)	+	c	d
(2)	*	(0)	(1)
(3)	:=	(2)	−

DAG (Directed Acyclic Graph)

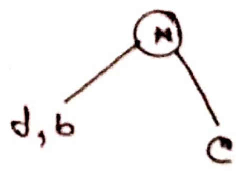
Example:

$a = b * c$
$d = b$
$e = d * c$
$b = e$
$F = b + c$
$g = d + f$

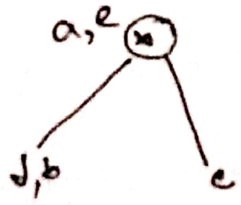
Step 1:



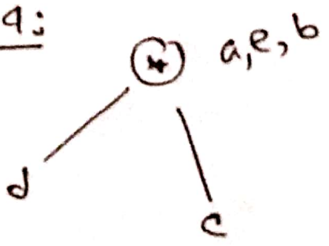
Step 2:



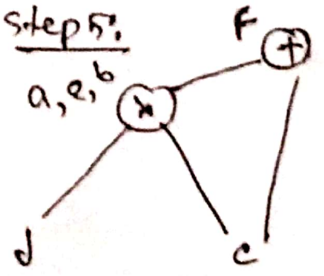
Step 3:



Step 4:



Step 5:



Step 6: