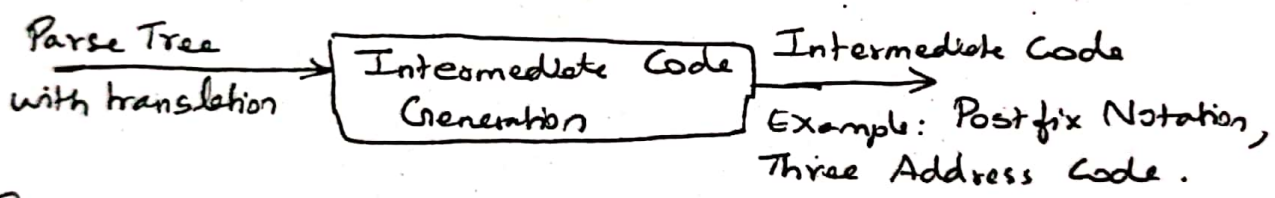
Intermediate Code Generation

* Intermediate code can translate the source program into the machine program.

* Intermediate code is generated because the compiler can't generate machine code directly in one pass.

* First it converts the source program into intermediate code, which performs efficient generation of machine code further.

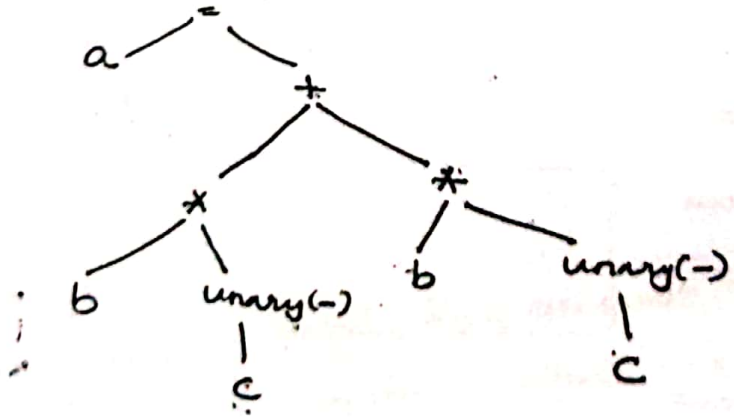
* The intermediate code can be represented in the form of postfix notation, syntax tree, directed acyclic graph, three address codes → Quadruples and Triples



Representation of Intermediate Code Generation

1) Syntax Tree:

$a = b * -c + b * -c$



Scanned with CamScanner

2. Three address Code :

* These are statements of form $c := a \ op \ b$, i.e. in which there will be at most three addresses i.e. two for operands & one for Result. Each instruction has at most one operator on the right hand side.

* Other form is $x := op \ y$

Example :

$a = b * -c + b * -c$ ⟹ Three address Code

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

Types of Three Address Code :

① Assignment Statement (Binary)

$x := y \ op \ z$

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

Intermediate Code :

```
t1 := c * d
t2 := b + t1
```

② Assignment Statement (Unary)

$x := op \ y$

Example : $a = b * -c$

Intermediate Code :

```
t1 = -c
t2 = b * t1
a = t2
```

③ Copy Statement

$x := y$

Scanned with CamScanner

④. Unconditional jump.

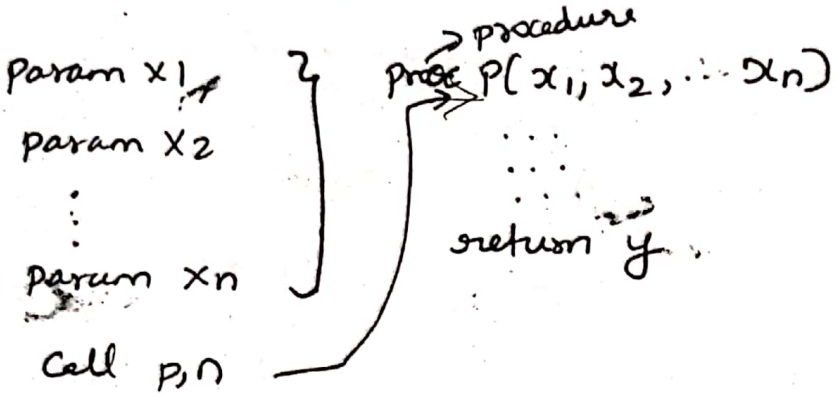
goto L

⑤. Conditional jump

if x relop y goto L

Example: if a < b goto L

⑥. Param x and Call P, n



⑦. Indexed

x := y[i]

x[i] = y

Example:

a[i] = b[j]

```
t := b[j]
a[i] := t
```

⑧. Pointers

x := &y

x := *y

*x = y

3. Postfix Notation:

* In postfix notation, the operator comes after an operand, i.e the operator follows an operand.

* Example:

(a+b)*(c+d) ⇒ Postfix Notation: ab+cd+*

(c*d)-(a*b) ⇒ Post fix Notation: cd*ab*-

Scanned with CamScanner

Implementation of 3 address Statement.

① Quadruples

② Triples

③ Indirect Triples

① Quadruples :

* Record Structures with 4 fields which we call operator, argument 1, argument 2 and Result.

$a = b * -c + b * -c$ ⇒ Intermediate Code :

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

S.No.	op	Arg1	Arg2	Result
(0)	Unary-	c	—	t1
(1)	*	b	t1	t2
(2)	Unary-	c		t3
(3)	*	b	t3	t4
(4)	+	t2	t4	t5
(5)	=	t5		a

② Triples

Each instruction in triples presentation has 3 fields. op, arg1 and arg2.

Location	op	arg1	arg2
(1)	Unary-	c	
(2)	*	b	(1)
(3)	Unary-	c	
(4)	*	b	(3)
(5)	+	(2)	(4)
(6)	=	(5) a	

Scanned with CamScanner

Indirect Triples.

* This representation is an enhancement over triples representation. It uses pointers instead of position to store results.

* This enables the optimizers to freely reposition the sub expression to produce an optimized code.

ptr	Statement
35	(0)
36	(1)
37	(2)
38	(3)
39	(4)
40	(5)

Location	op	arg1	arg 2
(0)	unary-	c	
(1)	*	b	(0)
(2)	unary-	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	=	a	(4)

Practise problem:

Translate the following expression to quadruple, triple and Indirect triple.

$$a+b \times c / e \uparrow f + b \times c$$

Scanned with CamScanner