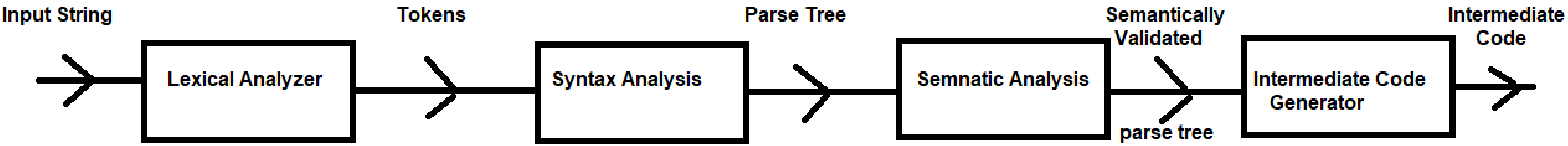
Intermediate Code

Intermediate Code



What is Intermediate Code

- Intermediate code (IC) is intermediate to source program and machine program.
- IC is generated because its not possible for compiler to generate machine code directly in one pass. If we want to do so then a full native compiler is required for each new machine.
- IC is machine independent, so can be executed on any platform.

Types of Intermediate Code Representation

- Postfix Notation
- Syntax tree & Parse Tree
- Three Address code
  - Quadruples
  - Triples
  - Indirect Triples

Types of Intermediate Code Representation

- Postfix Notation in Arithmetic Statements: Operator appears after operands. E.g.
  - a*d-(b+c)=ad*bc+-
  - a+(b*-c)=abc-*+

Types of Intermediate Code Representation

- If we want to convert postfix notation into infix notation, then

	String symbols	Stack
	a d* bc + –	
	a	a
1	d	a d
2	*	(a * d)
3	b	(a * d) b
4	c	(a * d) b c
5	+	(a * d) (b + c)
6	–	(a * d) – (b + c)
7		

57 + 2 * 3 /

Symbol	Stack	Description
5	5	
7	5 7	5 +7
+	12	
2	12 2	
*	24	12 * 2
3	24 3	
/	8	24/3

Types of Intermediate Code Representation

- Postfix Notation in Control Statements
  - Jump to label L in postfix notation is written as **L jump**
  - Jump to label L if e1 has smaller value than e2 is written as **e1 e2 L jlt**
  - Jump to label L if e has value equal to zero is written as **e L jeqz**

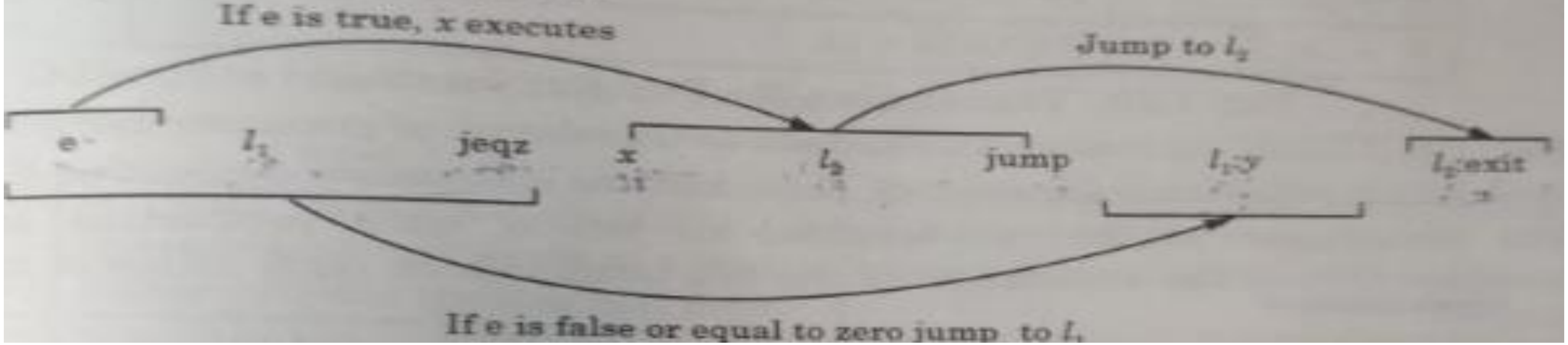
Types of Intermediate Code Representation

- Postfix Notation in Control Statements
  - E.g. if e then x else y

If - else statement can be written as

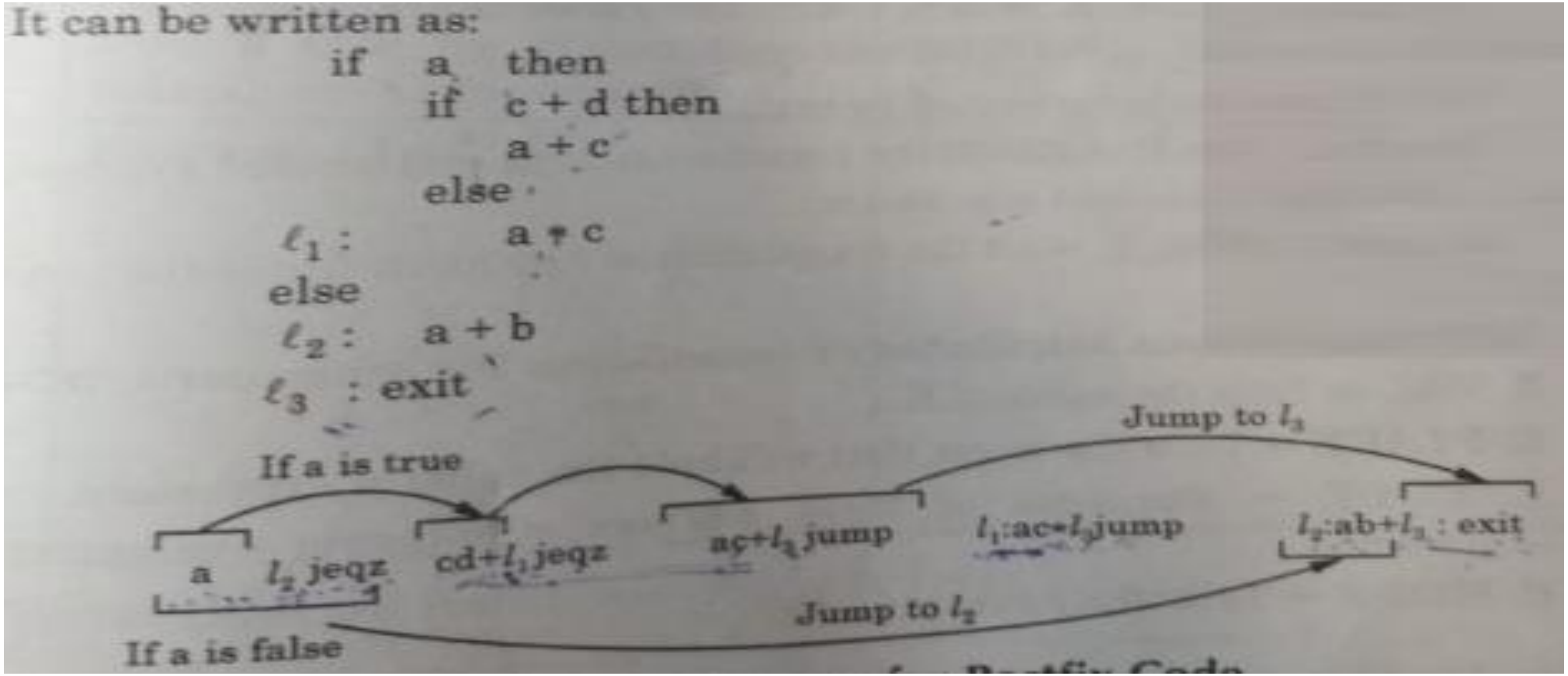
```
If   e   then
         x
else
ℓ1 :   y
ℓ2 :   exit.
```

The postfix Notation will be



Types of Intermediate Code Representation

- Postfix Notation in Control Statements
  - E.g. if a then if c+d then a+c else a*c else a+b


It can be written as:

```
          if   a   then
               if  c + d then
                   a + c
               else
  l1 :             a * c
  else
  l2 :         a + b
  l3  : exit
```

Types of Intermediate Code Representation

- Postfix translation for the grammar

Production	Semantic Action
$E \rightarrow E^{(1)} + E^{(2)}$	$E.CODE = E^{(1)}.CODE \;\|\|\; E^{(2)}.CODE \;\|\|\; '+'$
$E \rightarrow E^{(1)} * E^{(2)}$	$E.CODE = E^{(1)}.CODE \;\|\|\; E^{(2)}.CODE \;\|\|\; '*'$
$E \rightarrow (E^{(1)})$	$E.CODE = E^{(1)}.CODE$
$E \rightarrow id$	$E.CODE = id$

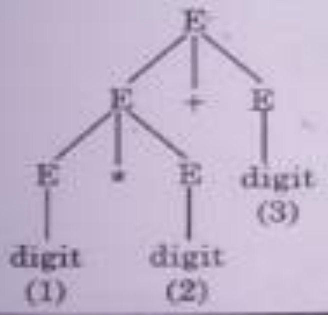
- E.Code represents 3-address statements evaluating the expression
- E.val represents the value of E
- E.place represents the name that will hold the value of the expression
- || represents the concatenation symbol

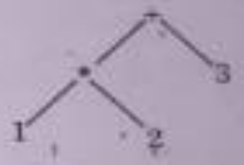
Types of Intermediate Code Representation

- Parse tree and Syntax tree

6.1 Difference b/w Parse Tree & Syntax Tree

	Parse Tree	Syntax Tree
1	It can contain operators & operands at any node of tree *i.e.* either interior node or leaf node.	It contains operands at leaf node & operators as interior nodes of Tree.
2	It contains duplicate or redundant information.	It does not contain any redundant information.
3	Parse Tree can be changed to Syntax tree by elimination of redundancy *i.e.* by compaction	Syntax tree cannot be changed to Parse Tree.
4	**Example :** 1 * 2 + 3	**Example :** 1 * 2 + 3





Types of Intermediate Code Representation

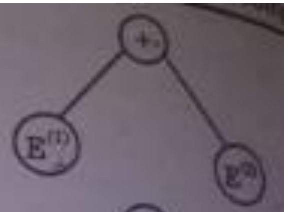
- Syntax directed translation of Parse tree and Syntax tree

Production	Semantic Action
$E \rightarrow E^{(1)} + E^{(2)}$	$\{E.VAL = Node\ (+, E^{(1)}.VAL, E^{(2)}.VAL)\}$
$E \rightarrow E^{(1)} * E^{(2)}$	$\{E.VAL = Node\ (*, E^{(1)}.VAL, E^{(2)}.VAL)\}$
$E \rightarrow (E^{(1)})$	$\{E.VAL = E^{(1)}.VAL\}$
$E \rightarrow -E^{(1)}$	$\{E.VAL = UNARY\ (-, E^{(1)}.VAL\}$
$E \rightarrow id$	$\{E.VAL = Leaf\ (id)\}$

Node $(+, E^{(1)}, VAL, E^{(2)}. VAL)$ will create a node labeled +.

$E^{(1)}. VAL$ & $E^{(2)}. VAL$ are Left & Right children of this node.

Similarly **Node** $(*, E^{(1)}. VAL, E^{(2)}. VAL)$ will make the syntax as :

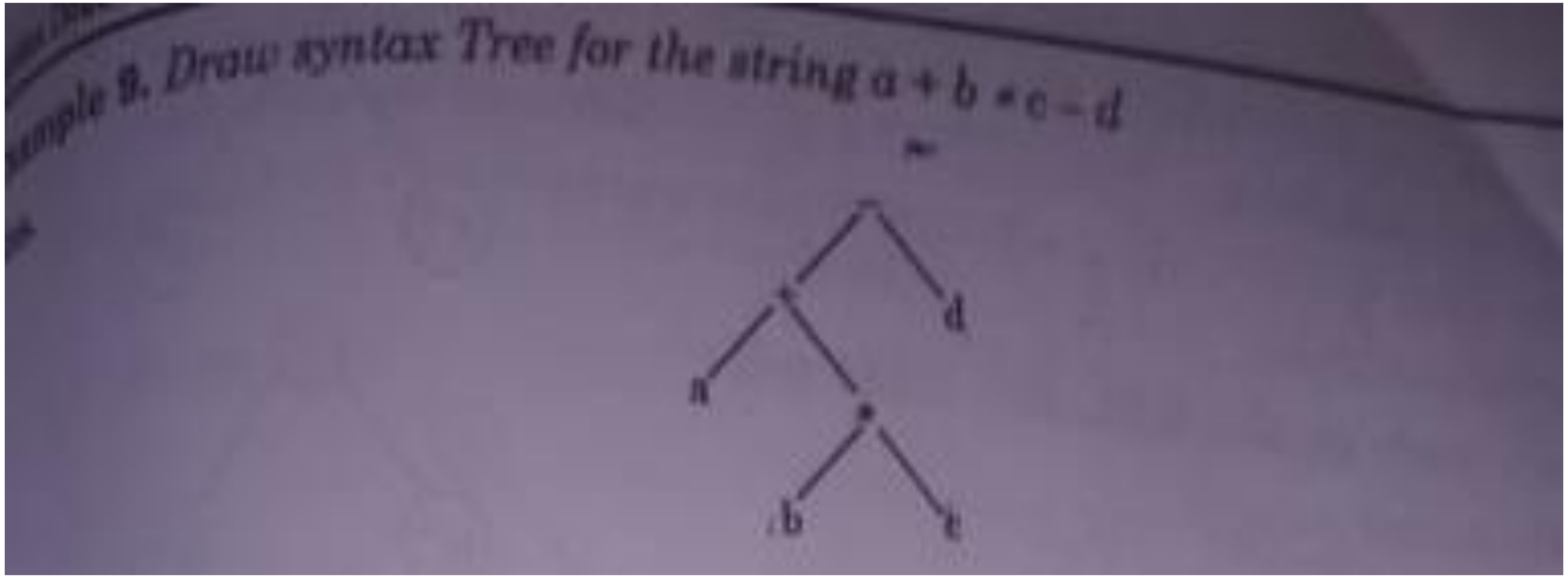


Function **UNARY** $(-, E^{(1)}. VAL)$ will make a node–(unary minus) & $E^{(1)}. VAL$ will be only child of it.

Function **LEAF (id)** will create a Leaf node with label id.

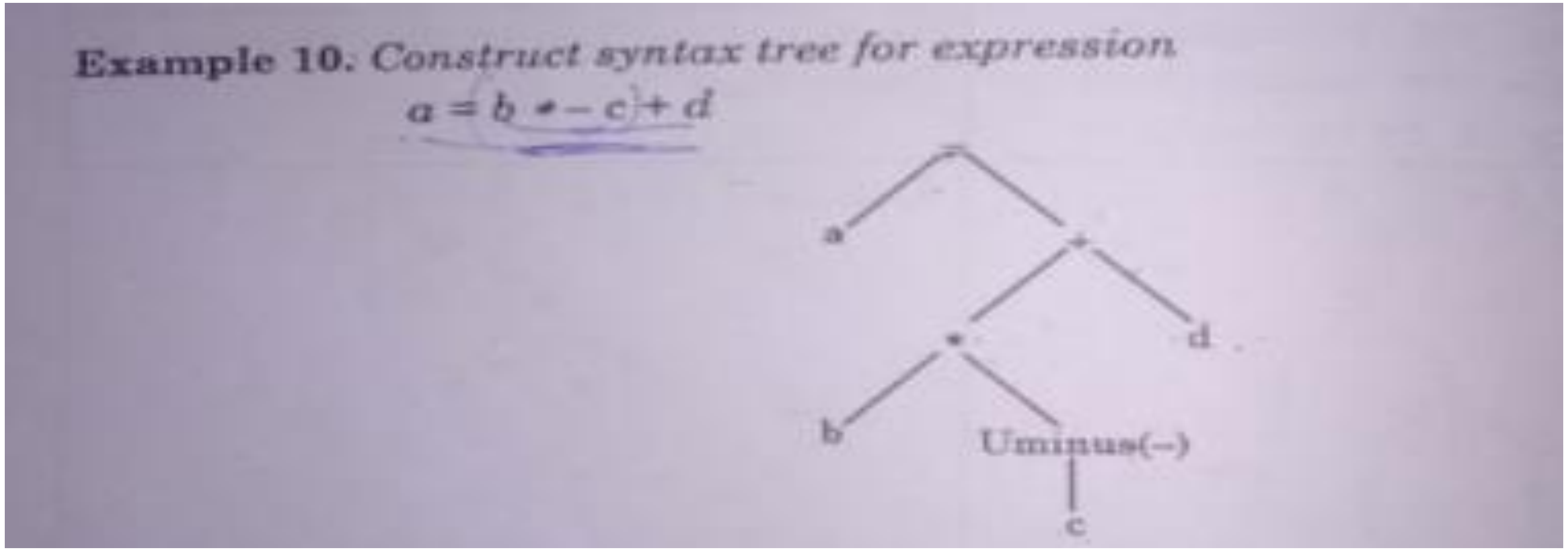
Types of Intermediate Code Representation

- Parse tree and Syntax tree



Types of Intermediate Code Representation

- Parse tree and Syntax tree

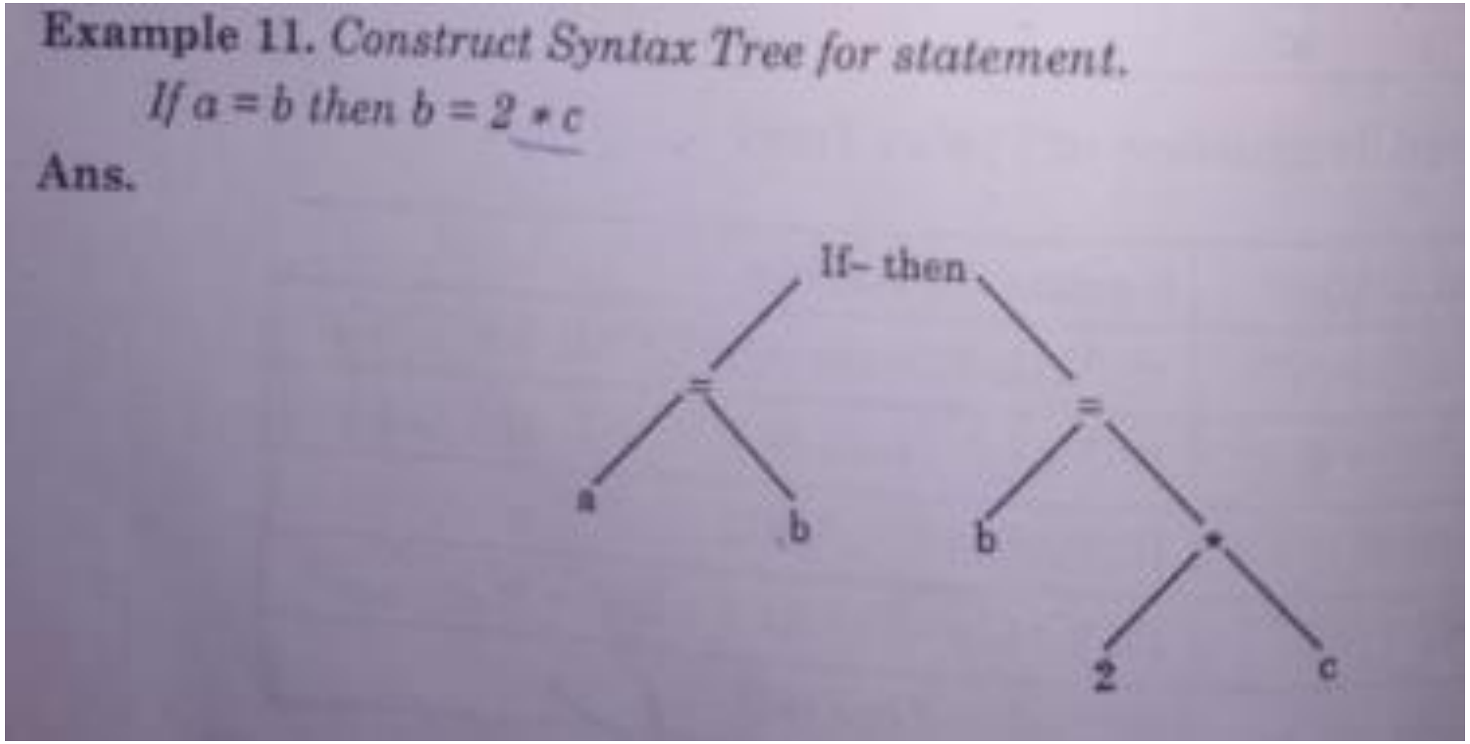


Example 10. *Construct syntax tree for expression*

$a = b * - c + d$

Types of Intermediate Code Representation

- Parse tree and Syntax tree

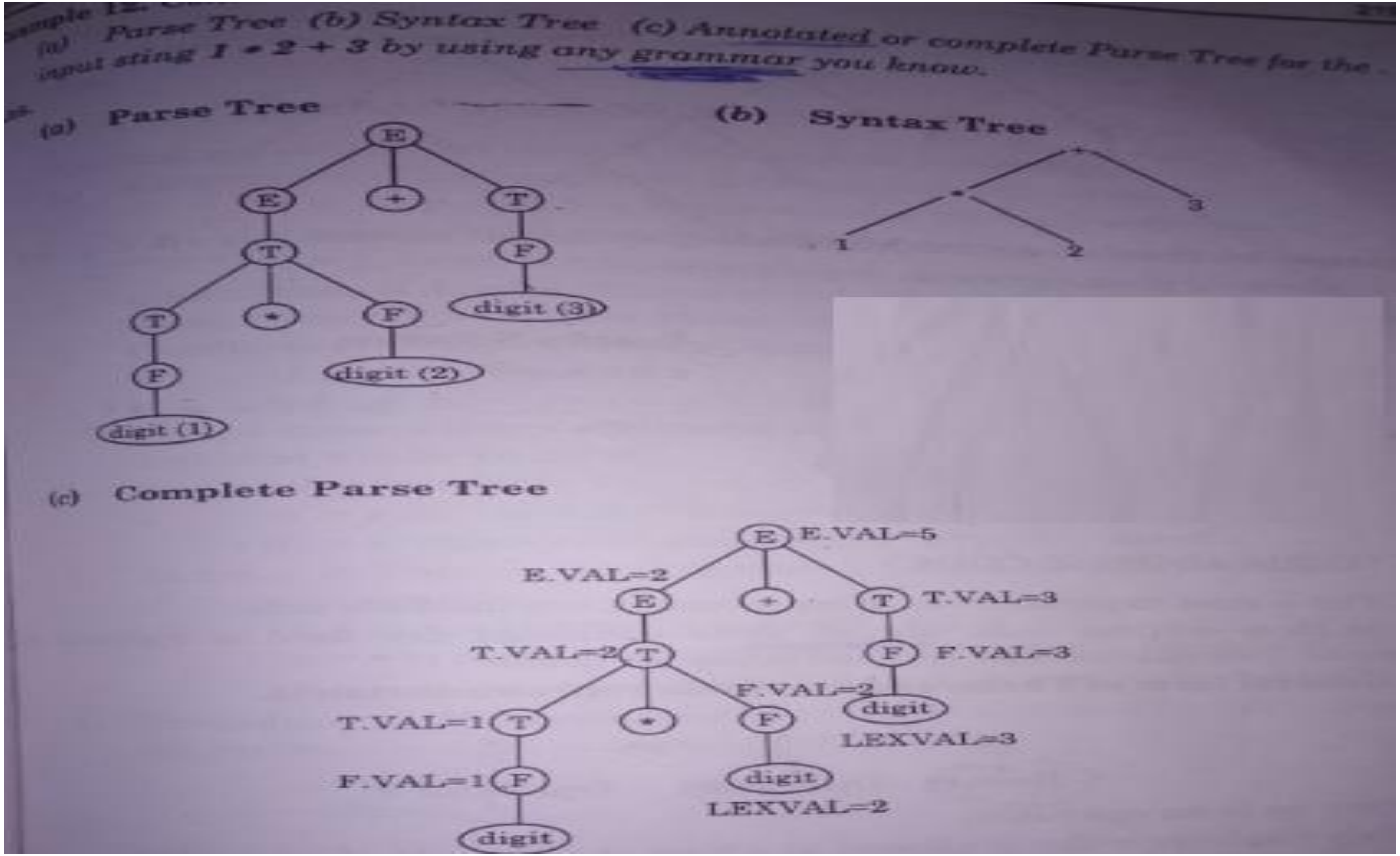


Example 11. *Construct Syntax Tree for statement.*

If $a = b$ *then* $b = 2 * c$

Ans.

Types of Intermediate Code Representation



(a) Parse Tree (b) Syntax Tree (c) Annotated or complete Parse Tree for the input sting 1 * 2 + 3 by using any grammar you know.

(a) Parse Tree

E
E + T
T F
T * F digit (3)
F digit (2)
digit (1)

(b) Syntax Tree

+
* 3
1 2

(c) Complete Parse Tree

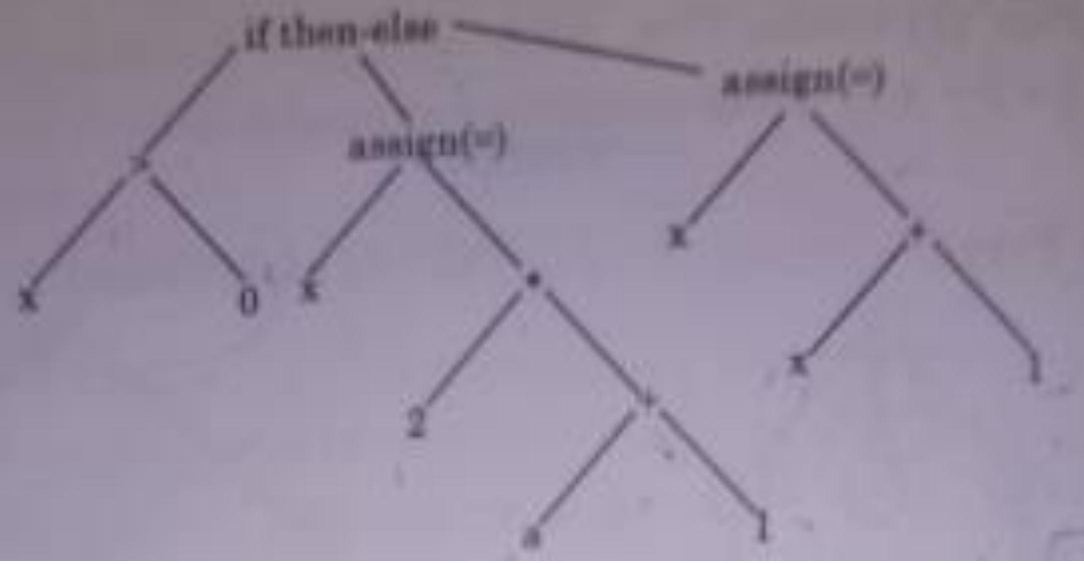
E E.VAL=5
E.VAL=2 E + T T.VAL=3
T.VAL=2 T F F.VAL=3
F.VAL=2 digit
T.VAL=1 T * F
LEXVAL=3
F.VAL=1 F digit
LEXVAL=2
digit

Types of Intermediate Code Representation

Example 13. *Consider the following code. Draw its syntax Tree*

*If x > 0 then x = 2 * (a + 1) else x = x + 1.*

Ans.

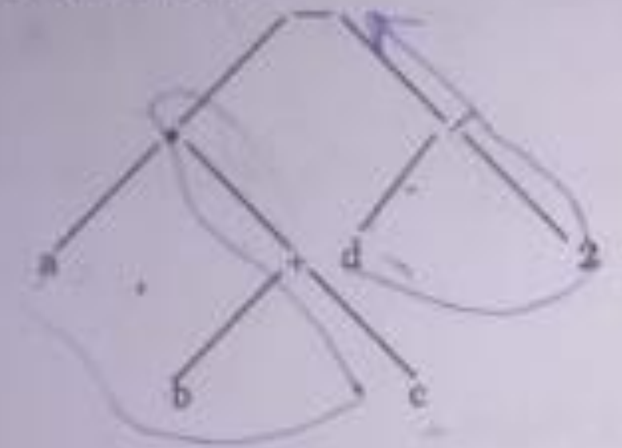


Types of Intermediate Code Representation

Example 14. *Draw syntax tree for following arithmetic expression a * (b + c) – d/*
Also write given expression in postfix form.

Ans.

Syntax Tree.



Postfix Notation

a b c + * d2/–

a bc + * d2 /

Types of Intermediate Code Representation

- Three Address Code
  - In 3 address code, at most 3 addresses are used to represent any statement.
  - Two addresses are for operand and one for result.
  - Only single operator is allowed at a time at right side of the expression.
  - E.g. a=b+c+d in 3 address code will be written as:

    t1=b+c

    t2=t1+d

    a=t2

    Where t1 and t2 are temporary variables generated by compiler.

Types of Intermediate Code Representation

- Representation of Three Address Code statements
  - Quadruples: It is a structure which contains at most four fields: Operator, Argument1, Argument2, Result. It uses temporary variables to store the values of results.
  - E.g. a=b+c*d, its equivalent 3 address code is:

    t1=c*d

    t2=b+t1

    a=t2
  - Now its quadruple representation is

	Operator	arg 1	arg 2	Result
(0)	*	c	d	t1
(1)	+	b	t1	t2
(2)	=	t2		a

Types of Intermediate Code Representation

- Representation of Three Address Code statements
  - Triples: This representation contains three fields: Operator, Argument1, Argument2. It does not use temporary variables to store results, it use number to represent pointer to that record where value of result is stored.
  - E.g. a=b+c*d, its equivalent 3 address code is:

    t1=c*d

    t2=b+t1

    a=t2
  - Now its triple representation is:

	Operator	arg 1	arg 2
(0)	*	c	d
(1)	+	b	(0)
(2)	=	a	(1)

Types of Intermediate Code Representation

- Representation of Three Address Code statements
  - Indirect Triples: Like triples, it also don't use temporary variables to store result. It uses pointers to point to record where value of result is stored. But pointer itself are indexed.
  - E.g. a=b+c*d, its equivalent 3 address code is:

    t1=c*d

    t2=b+t1

    a=t2
  - Now its indirect triple representation is:

	Statement
(0)	(11)
(1)	(12)
(2)	(13)

	Operator	arg1	arg2
(11)	*	c	d
(12)	+	b	(11)
(13)	=	a	(12)