Date _____

Expt. No. _____ Page No. _____

Syntax Directed Translation (SDT)

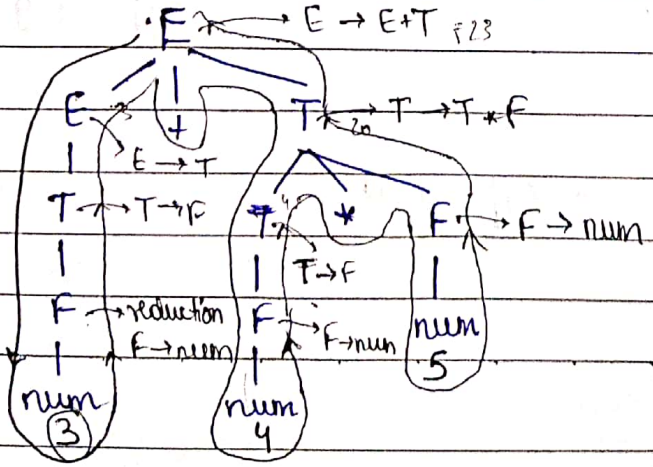
(works in Semantic analysis)

SDT = Grammar + Semantic Rule

- Syntax Directed Definition (How to give rule of syntax)
  - Synthesized Attribute
    - ↓
    - left side non-terminal's value is calculated by it's children
    - $A \rightarrow BCD$
  - Inherited Attribute
    - ↓
    - $C_i = A_i$ } Value of children is calculated via parent
    - $C_i = B_i$, $C_i = D_i$ } Value of children is calculated via siblings
- Syntax Directed Translation Scheme (order of execution)

eg:

SDT → Grammar	Semantic Rule
$E \rightarrow E^{(1)} + T$ (same E just to distinguish b/w left & right side E)	{ E.value = $E^{(1)}$.value + T.value }
$E \rightarrow T$	{ E.value = T.value }
$T \rightarrow T^{(1)} * F$	{ T.value = $T^{(1)}$ value * F.value }
$T \rightarrow F$	{ T.value = F.value }
$f \rightarrow num$	{ F.value = num.lexvalue }



eg → 3 + 5 * 4 = 23 (input)

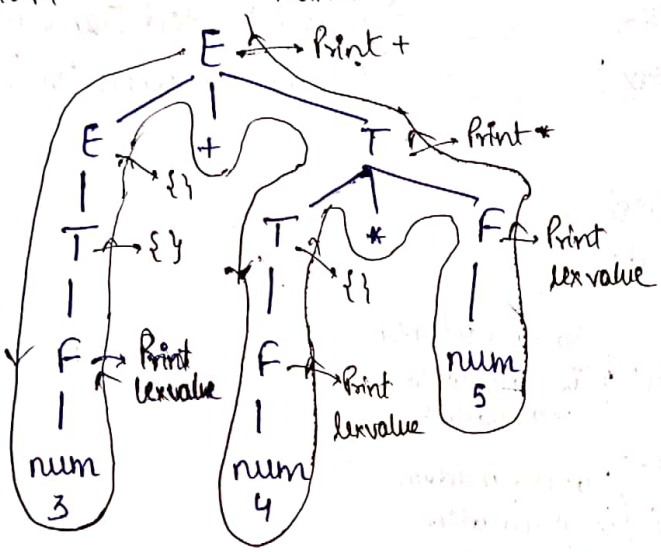
Teacher's Signature _____

Design a SDT that converts infix to postfix.

eg→

Production	Action
$E \rightarrow E+T$	{ Print '+' }
$E \rightarrow T$	{ }
$T \rightarrow T*F$	{ Print '*' }
$T \rightarrow F$	{ }
$F \rightarrow num$	{ Print (lexvalue) }

eg→ Infix = 3+4*5
Postfix = 3 4 5 * +

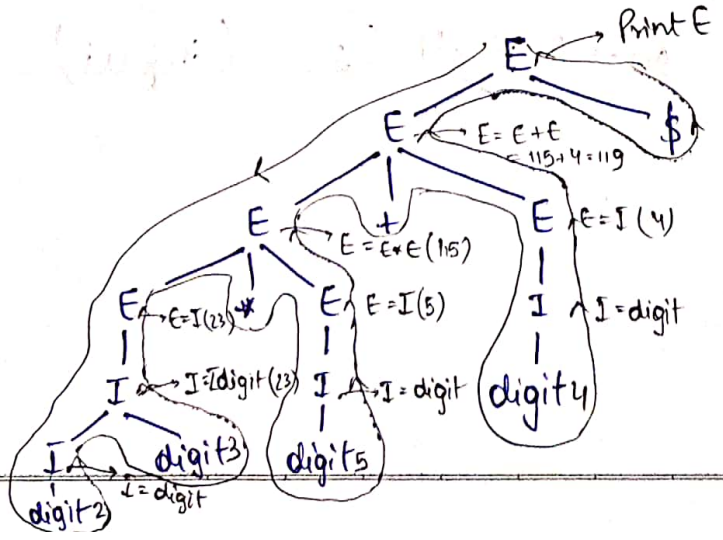


Output: 3 4 5 * +

Que- Construct SDT for Desk Calculator → normal Arithmetic calc.

$S \rightarrow E\$$	{ Print E.value }
$E \rightarrow E^1+E^2$	{ E.value = E^1.value + E^2.value }
$E \rightarrow E^1*E^2$	{ E.value = E^1.value * E^2.value }
$E \rightarrow I$	{ E.value = I.value }
$I \rightarrow I\,digit$	{ I.value = 10 * I'.value + digit } → for 2 digit no. evaluation
$I \rightarrow digit$	{ I.value = digit.lexvalue }

eg → I/P = 23 * 5 + 4
O/P = 119



→ O/P = 119

Synthesized Attribute: eg → A → BC
{A.value = B.value, C.value} → Value of parent is calculated using all its children

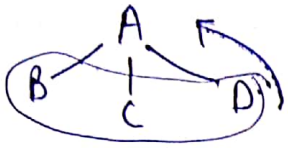
Inherited Attribute: eg → A → BCD
{B.value = A.value}
{C.value = D.value}
{E.value = B.value}
→ Value of children is calculated using value of parent or using value of siblings

Types of SDT

S-attributed SDT	L-attributed SDT
1) Uses only synthesized attributes eg → A → BC A.value = B.value, C.value	1) Uses both synthesized & inherited but each inherited is restricted to inherit from parent or left sibling eg → A → BCD → A.value = B.value, C.value, D.val B.value = A.value ✓ C.value = B.value ✓ C.value = D.value ✗
2) Semantic action placed at the right end of production eg: E → E+T {E.value = E.value + T.value}	2) Semantic action placed anywhere in the line of production. eg → A → BCD {-} A → {-} BCD A → B {-} CD

Teacher's Signature

3) Attributes are calculated in a Bottom up Manner.



3) Evaluated by Depth first and left to right.

Intermediate Code Generation → converts grammar to a 3 address code

3 types:

1) Postfix Notation
2) Syntax Tree
3) Three Address Code < Quadruple, Triple.

1) Postfix Notation

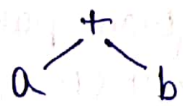
Infix	Prefix	Postfix
a+b	+ab	ab+
(a+b)*c		ab+c*

why? Postfix →
- better implementation
- implemented through stack
- no parenthesis
- no associativity & operator precedence of the operators

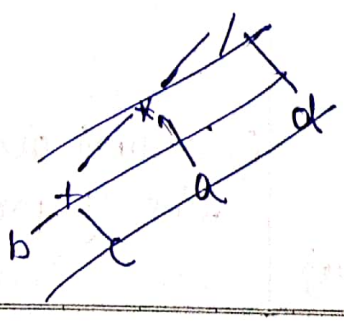
② Syntax Tree

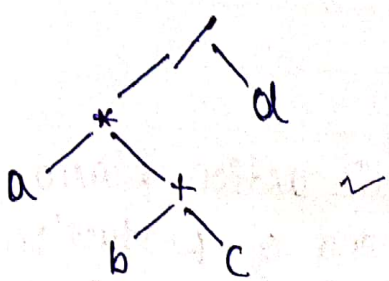
leaf node → operand
intermediate nodes → operator

eg → a+b



eg → a*(b+c)/d





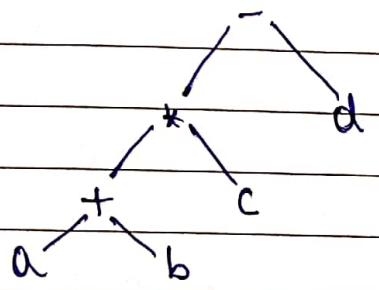
Date _____

Expt. No. _____ Page No. _____

From the syntax tree, we can calculate:
infix : left, root, right
prefix : root, left, right
postfix : left, right, root

Check infix : a*b+c/d
Prefix : /*a+bcd
Postfix : abc+*d/

eg → ((a+b)*c)-d



Prefix : -*+abcd
Postfix : ab+c*d-

(3) ~~Intermediate Co~~ Three Address Code

The o/p of the semantic analysis is converted to a three address code

→ It converts the input to a fixed particular format

x := y op z

Variable or temp variable — operator

↓

atmost 3 variables
so 3 memory addresses

Teacher's Signature _____

eg → Input: $a = x + y * z$ → Convert to 3 address code

$t_1 = y * z$ (3)
$t_2 = x + t_1$ (3)
$a = t_2$ (2)

Types of Three Address statement:

1) Logical & Binary operator → $x = y$ op z
2) Unary operator → $x =$ op y
3) Copy statement → $x = y$
4) Procedure call → param y
   param x
5) Index Assignment → $x = y[i]$
   $x[i] = y$
6) Conditional jump → if x relates y goto serial
7) Address & pointer → $x = \&y$
   $x = *y$

Two types of implementation

1) Quadruple

↳ uses four fields → arg_1, arg_2, op, result

eg → $a = b * -c + b * -c$

(0) $t_1 = -c$
(1) $t_2 = b * t_1$
(2) $t_3 = -c$
(3) $t_4 = b * t_3$
(4) $t_5 = t_2 + t_4$
(5) $a = t_5$

Date _____

Expt. No. _____ Page No. _____

Quadruple:

	op	arg_1	arg_2	result
(0)	unary-	c		t_1
(1)	*	b	t_1	t_2
(2)	unary-	c		t_3
(3)	*	b	t_3	t_4
(4)	+	t_2	t_4	t_5
(5)	assign	t_5		a

2) Triple:

	op	arg_1	arg_2
(0)	unary-	c	
(1)	*	b	(0)
(2)	unary-	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	assign	a	(4)

Ques - Write the quadruple ~~tuple~~ triple for following:

$(x+y) * (y+z) + (x+y+z)$

Three address code: (0) $t_1 = x+y$

(1) $t_2 = y+z$

(2) $t_3 = t_1 * t_2$

(3) $t_4 = t_1 + z$

(4) $t_5 = t_3 + t_4$

Teacher's Signature _____

Quadruple:

	op	arg_1	arg_2	result.
(0)	+	x	y	t_1
(1)	+	y	z	t_2
(2)	*	t_1	t_2	t_3
(3)	+	t_1	z	t_4
(4)	+	t_3	t_4	t_5

Triple:

	op	arg_1	arg_2
(0)	+	x	y
(1)	+	y	z
(2)	*	(0)	(1).
(3)	+	(0)	z
(4)	+	(2)	(3)

Que- Give the three address code for the following program.

(i)
```
int i ×
i = 1 ✓
while i<10 do ✓ (from condition)
    if x>y then
        i = x+y
    else
        i = x-y
```

1) i = 1
2) if i<10 goto (4)
3) goto (10)
4) if x>y goto (6)
5) goto (8)
6) i = x+y
7) goto (2)
8) i = x-y
9) goto (2)
10) stop

Date

Expt. No. Page No.

(ii)
```
c = 0
do
{
  if (a<b) then
     x++;
  else
     x--;
  c++;
}
while (c<5);
```

1) c = 0
2) if a<b goto (4)
3) goto (7)
4) x = x+1
5) c = c+1
6) goto (10)
7) x = x-1
8) c = c+1
9) goto (10)
10) if (c<5) goto (2)
11) goto (12)
12) stop

More about Translation

1) Array reference in Arithmetic
2) Procedure call
3) Case statement

I Array reference in Arithmetic.

i) 1-D Array

int a[5] = {15, 10, 11, 44, 34}

Base address B ↓ ; w = diff of bytes (width)

Actual address memory	1100	1104	1108	1112	1116
Element	15	10	11	44	34
Index	0	1	2	3	4

↑ lower bound LB

Address = B + w(i - LB)
A[i]

Teacher's Signature

eg→
```
int i
int a[10]
i = 1
while (i<10)
{
    a[i] = 0
    i = i+1
}
```

1) i = 1
2) if i<10 goto (4)
3) goto (8)
4) $t_1 = 4 * i$
5) $a[t_1] = 0$
6) i = i+1
7) goto (2)
8) stop

Address
$A[i] = B + w * (i - LB)$
B → system generated (need not to write)
w → 4
i → i
LB → 0
$= 4(i-0)$
$= 4 * i$

ii) 2-D Array

Lr \ Lc	0	1	2	3
0	8	6	3	2
1	4	5	9	1
2	6	3	2	4

Row index: 0, 1, 2 — Column index: 0, 1, 2, 3 — A[m][n]

Row wise:

8	6	3	2	4	5	9	1	6	3	2	4

$A[i][j] = \text{Base Address} + w * [n * (i - Lr) + (j - Lc)]$

Column major:

8	4	6	6	5	3	3	9	2	2	1	4

$A[i][j] = \text{Base Address} + w * [(i - Lr) + m * (j - Lc)]$

eg→
```
int i;
int a[10][10];
i = 0;
while (i<10)
{
    a[i][i] = 1;
    i++;
}
```

Address
$A[i] = B + 4[10(i-0) + (i-0)]$
$= B + 4(10i + i)$
$= B + 4 \times 11i$
$= 44i$

1) i = 0
2) if i<10 goto (4)
3) goto (8)
4) $t_1 = 44 * i$
5) $a[t_1] = 1$
6) i = i+1
7) goto (2)
8) stop

Date _____

Expt. No. _____ Page No. _____

2) Case statement

eg→
```
switch(ch)
{
    case1: c = a+b;
           break;
    case2: c = a-b;
           break;
}
```

1) if ch=1 goto(3)
2) if ch=2 goto(5)
3) c = a+b
4) goto(7)
5) c = a-b
6) goto(7)
7) stop

for break

3) Procedure call

P(A1, A2, A3 --- An) → some func^c given

↳ 3 address code

param A1
param A2
:
param An
call P, n

eg→
```
void main()
{
    int x, y;
    - - - -
    swap(&x, &y);
    - - - - - -
}
void swap(int *a, int *b)
{
    int i;
    i = *b;
    *b = *a;
    *a = i;
}
```

1) call main
2) param &x
3) param &y
4) call swap, 2
5) i = *b
6) *b = *a
7) *a = i
8) stop

Teacher's Signature _____