USAMA SARWAR
FA17-BCS-090

Question # 01

(b) Type Checking $E_1 \rightarrow E_2 == E_3$

If (E_1.type == E_3.type) and (E.type == int/bool)
then E_1.type= boolean
else error

(a) Calculate S5

If & has higher priority as compared to it, then it should be below in grammer
i.e. our grammer should be:

$E \rightarrow T \# F$
$E \rightarrow T$
$T \rightarrow E \& T$
$T \rightarrow F$
$F \rightarrow id$

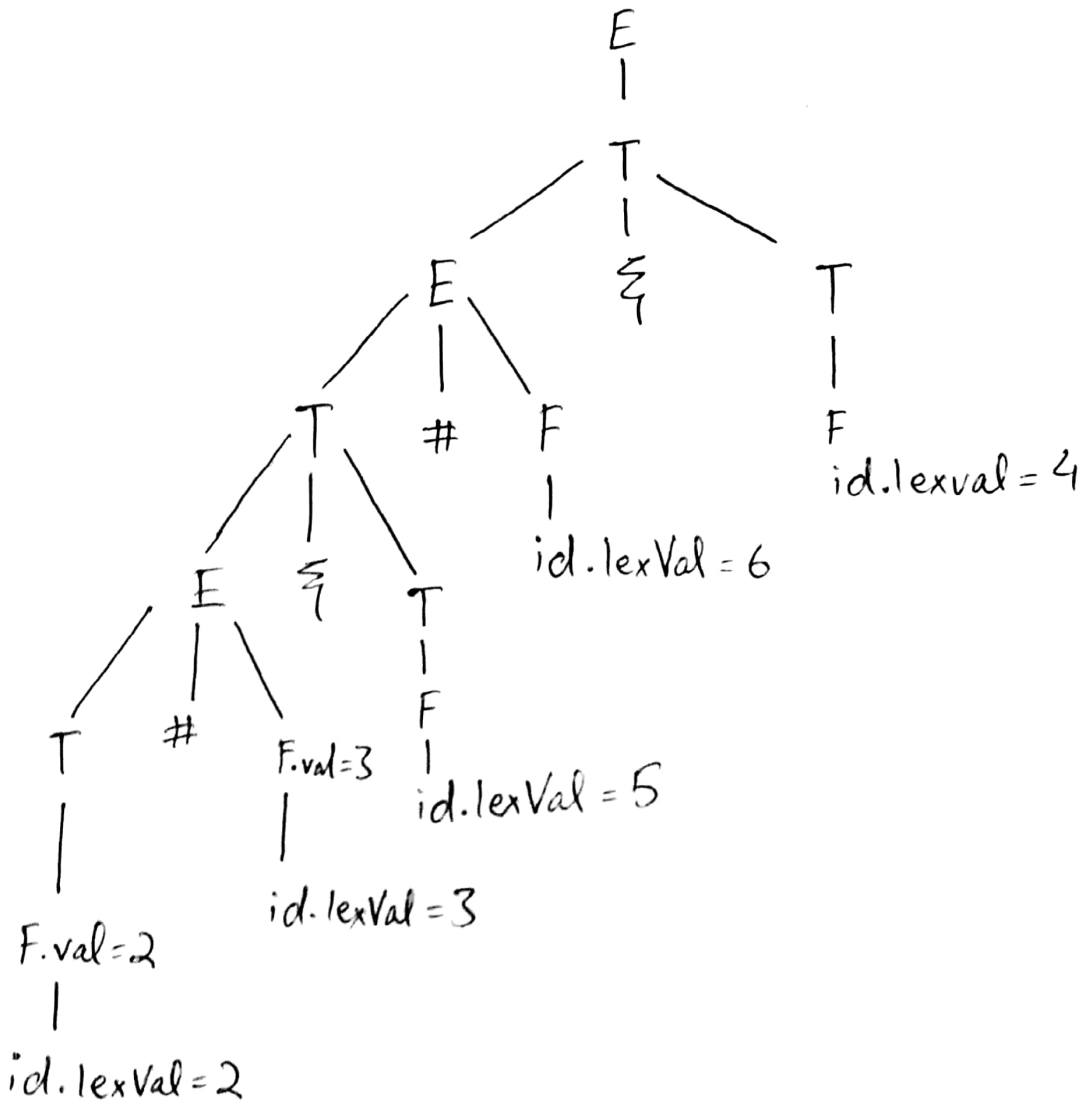
$SS \rightarrow 2 \# 3 \& S \# 6 \& 4$
$E \rightarrow T \# F$
$E \rightarrow T$
$T \rightarrow E \& T$
$T \rightarrow F$
$F \rightarrow id$

Calculating SS

E.val → T.val # F.val
E.val → T.val
T.val → E.val & T.val
T.val → F.val
F.val → id.LexValue

Question # 01

(a - continue)



id	#	id	&	id	#	id	&	id
2		3		5		6		4

Question # 01

(c) Intermediate Code

The representation that represents the source code in efficient way, called as intermediate code.

Types of intermediate code

1. Polish Notations
   - → Infix to Prefix
   - → Infix to Postfix
   - → Prefix to Postfix

   Like these
2. Three address code
   - → Quardruples
   - → Tripples
   - → Indirect triples

ABSTRACT Syntax Tree

It represents upcomming code in efficient ways by using rotations.

Symentic Analysis → Intermediate Code → Code optimization
Code Generation ←

Intermediate Code Id_1 Id_2 Id_3

temp1 = m to real val(10) a = b + c * 10 (in efficient way)
temp 2 = Id_3 x temp 1
temp 3 = Id2 + temp2
Id_1 = temp3

Question # 01

(d) So L attributed is better than S attribute. As it involves both Synthesized and inherited attribute.

→ In S attribute in grammer semantic is only placed at right position.

→ In L attribute we placed semantic rule at any where at start at end and at middle.

(e) The code become inefficient due to two factors

1- Programmer
2- Compiler

(f) $S_2 \rightarrow 11001.1001$

$1 \times 2 + 1$

$3 \times 2 + 0$

$6 \times 2 + 0$

$12 \times 2 + 1$

25

$1001 = 9$

So, $9/2^4 = 0.5625$

So, the answer is

$= 25.5625$

$N \rightarrow L_1 . L_2$	$\{N.dval = L_1.dval + L_2.dval / 2^{\wedge} L_2.count;\}$
$L_1 \rightarrow L_1 B$	$\{L.c = L_1.c + B.c;\ L.dval = L_2.dval + B.dval;\}$
$L \rightarrow B$	$\{L.c = B.c;\ L.dval = B.dval\}$
$B \rightarrow 0$	$\{B.c = 1;\ B.dval = 0\}$
$B \rightarrow 1$	$\{B.c = 1;\ B.dval = 1\}$

Question # 01

(G) $S \rightarrow f = ex$

$ex \rightarrow ex\ addop\ term \mid term$

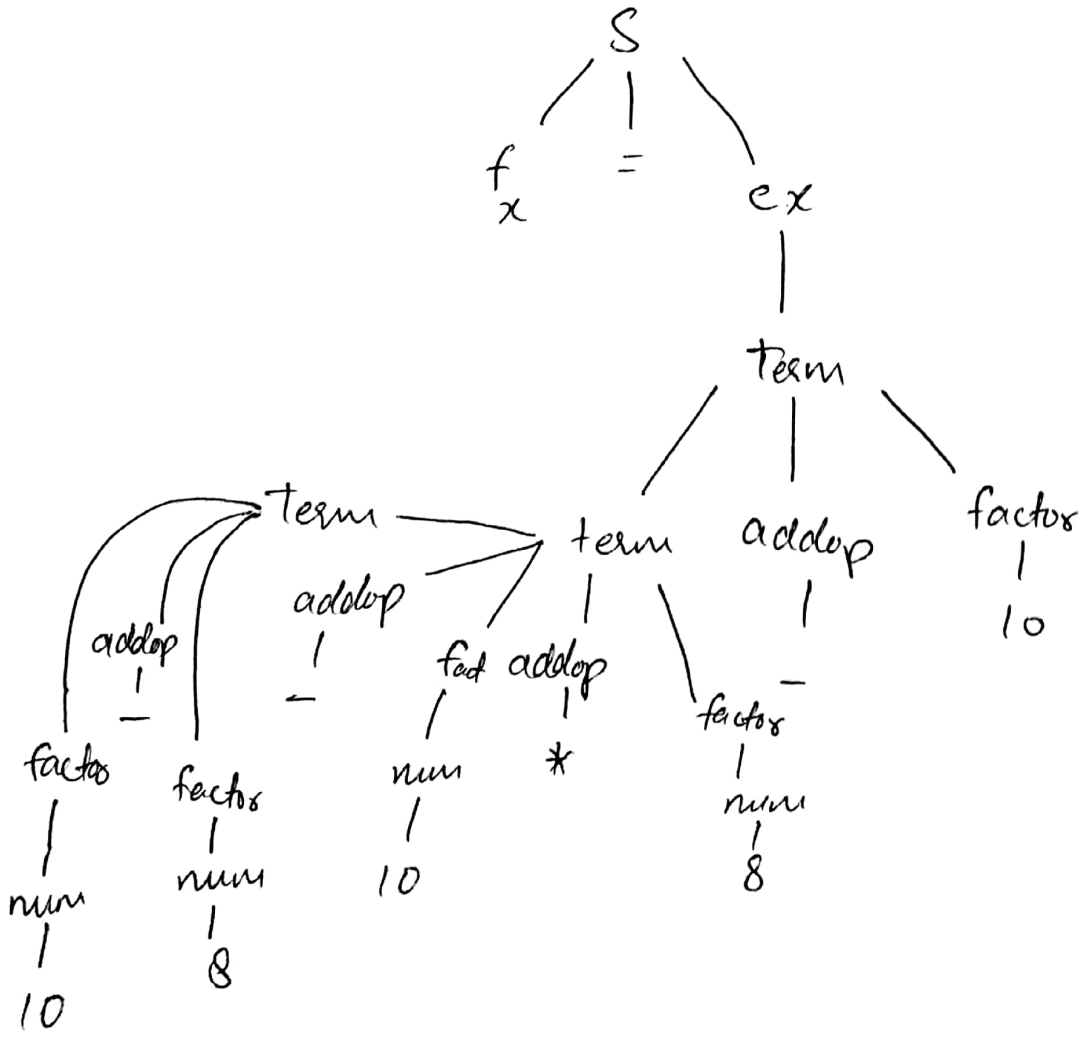
$term \rightarrow term\ addop\ factor \mid factor$

$addop \rightarrow + \mid - \mid * \mid /$

$factor \rightarrow x \mid num$

String S = x=10-8-10*8-10

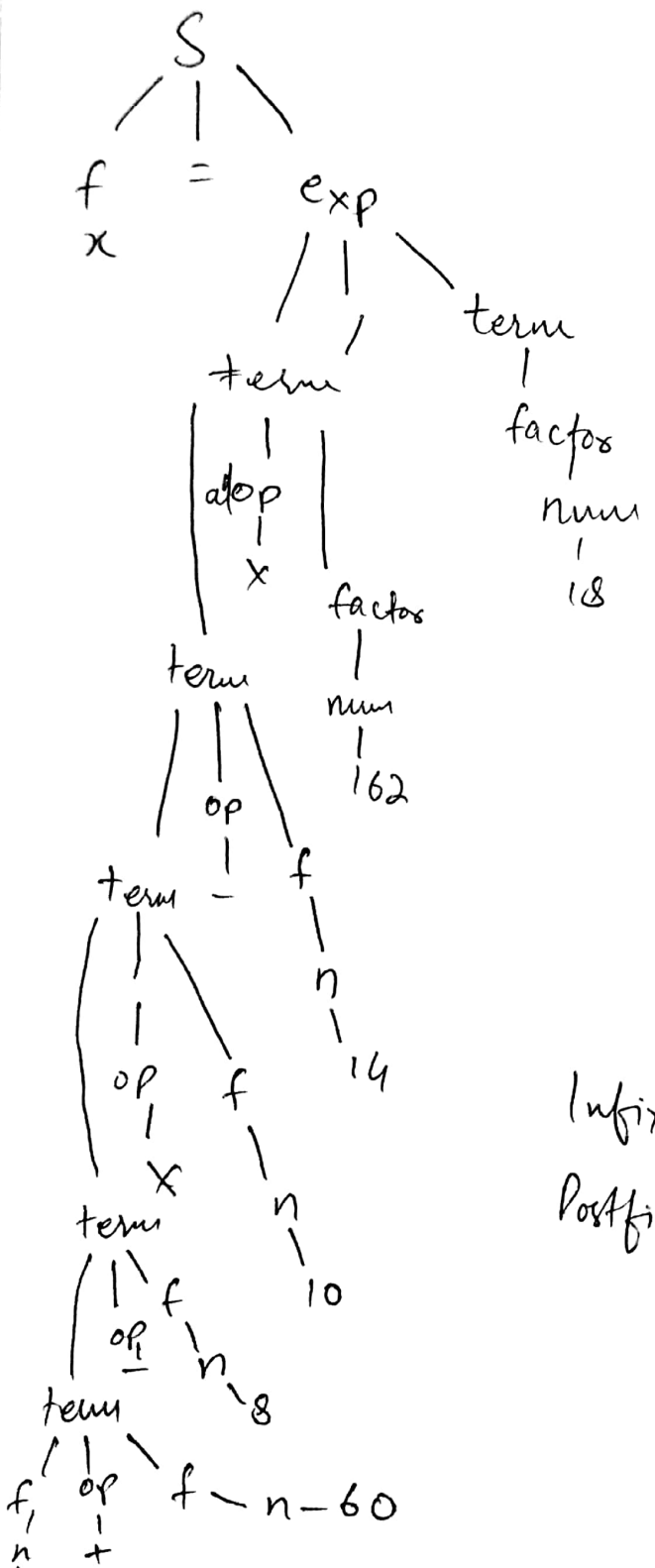
Annotated Parse Tree



Question # 01

(h) Postfix

x = 40 * 60 + 8 10 K - 84 / 162 K 18 / +



f ⇒ factor
n ⇒ num

Infix = 40 + 60 - 8 * 10 + 84 / 14 *

Postfix = 40 60 + 8 10 * - 84 14 / 162 * 18 / +

Question #01

(i)

CFG:

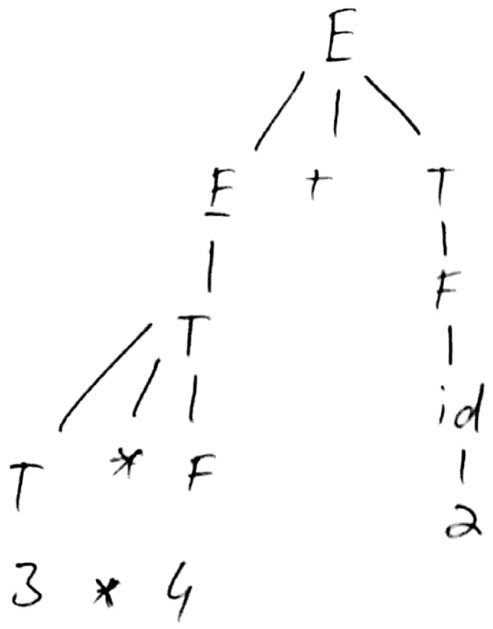
$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow id$$

Here we have

$S_4 \rightarrow 2+3\times4$

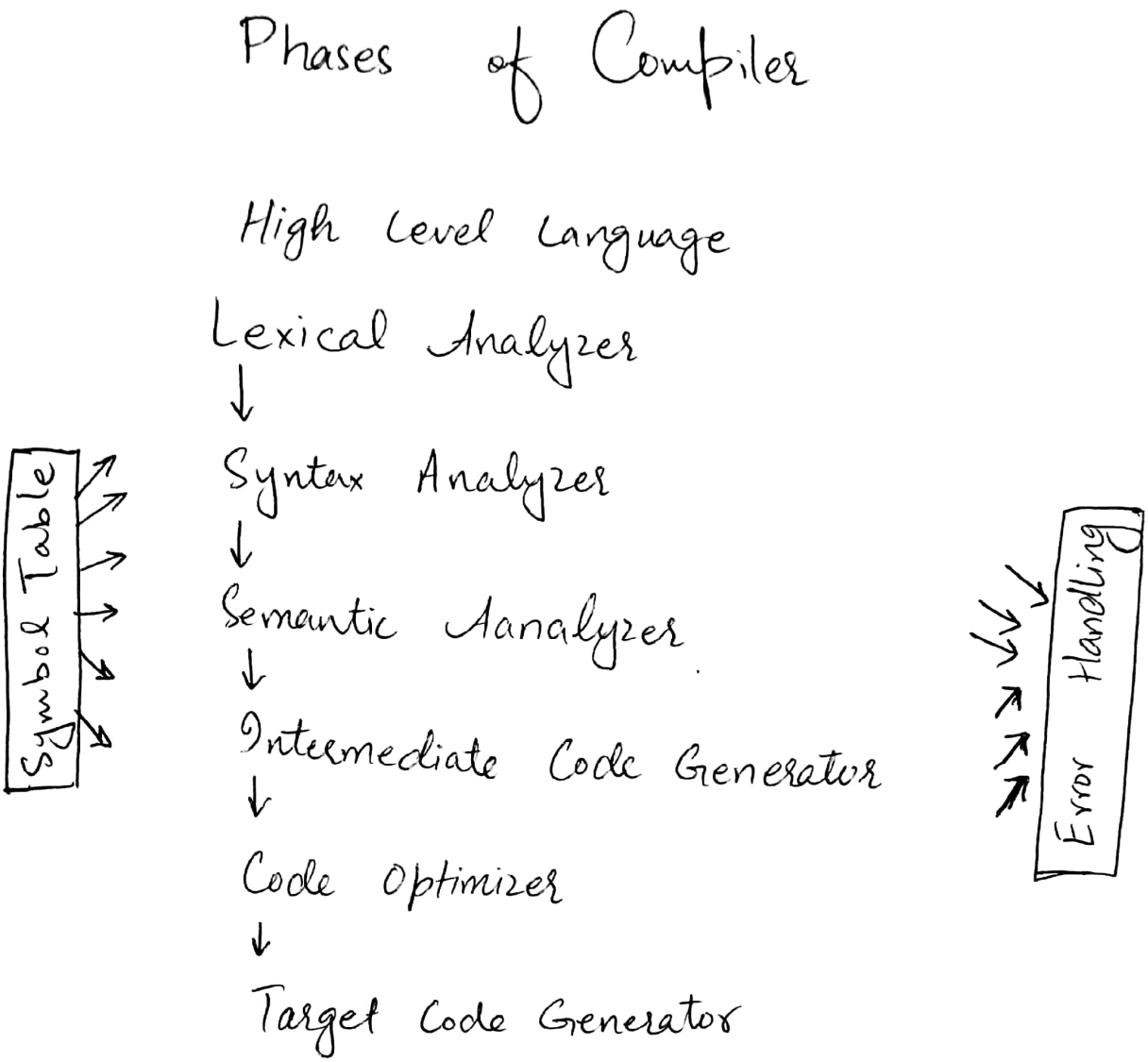


$$(3*4)+2=14$$

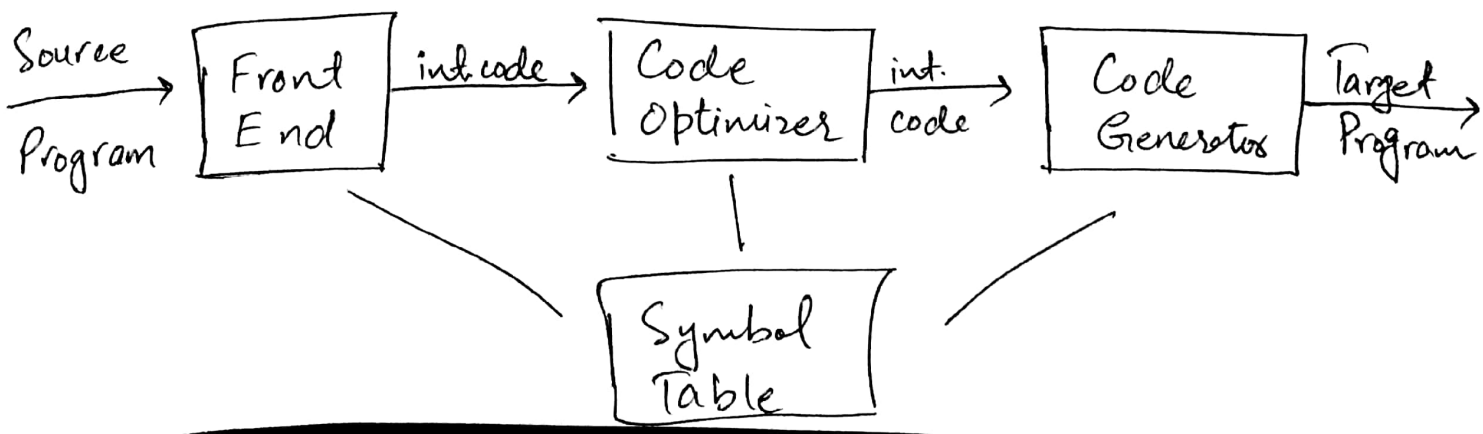
Question # 1

(I)

Phases of Compiler



3-Main Phases



Question # 01

(K) There are two notations for attaching semantic rules:

1. Syntax Directed Definitions

   High-level specification hiding many implementation details (also called Attribute Grammars).

2. Translation Schemes

   More implementation oriented: Indicate the order in which semantic rule are to be evaluated.

Question # 01

(L) Code Optimization

Code optimization is a technique required to produce efficient code and it makes program to consume less memory and delivers high speed. This optimization technique will be applied whenever it is needed.
It also reduces the time complexity of program.

✱ Issues:

There are two issues:

→ Meaning of the source code should be changed
→ The efficiency of the source code must be gained without changing the algorithm.

Techniques:

→ Dead code elimination
→ Common sub expression elimination
→ Strength Reduction
→ Code Movement

Question # 02.

Convert SG to 3AC

SG → a+b*c /e^f+-b+a

Solution.

$t_1 = -b$

$t_2 = e \wedge f$

$t_3 = b * c$

$t_4 = t_3 / t_2$

$t_5 = a + t_4$

$t_6 = t_5 + t_1$

~~$t_7 = t_6 + a$~~

$t_7 = t_1 + a$

Question # 03.

Convert Infix into Prefix

S1 => x = 40+60-8*10+84/14*162/18

Taking inverse

18/162 x 14/84+10x8-60+40

Input	Stack	Prefix
18	-	18
/	- /	18
162	- /	18162
x	- / x	18162
14	- / x	18162 14
/	- / x /	18162 14
84	- / x /	18162 14 84 / x /
+	- +	18162 14 84 / x / 10
10	- +	18162 14 84 / x / 10
x	- + x	
8	- + x	18162 14 84 / x / 10 8
-	- + -	18162 14 84 / x / 10 8 x
60	- + -	18162 14 84 / x / 10 8 x 60
+	- + - +	
40	- + - +	18162 14 84 / x / 10 8 x 60 40

Poping the Stack

18 162 14 84 / x / 10 8 x 60 40 + - +

Reversing: + - + 40 60 x 8 10 / x / 84 14 162 18

Question # 04.

Assuming

M = Muqsit
B = Bahadur
A = Ali
Y = Yasir
Aw = Awais
D = DAUD
Z = Zahid
Aq = Aqib
w = Waqas

Removing Indirect Recursion.

D → D Z | B Aq Aq | A Aq | Y w

D → D Z | D Z Aq Aq | w Aq Aq | A Aq | Y w

D → D Z | D Z Aq Aq | w Aq Aq | A Aq | D Z w | w w

Removing left Recursion

D → w Aq Aq D' | A Aq D' | w w D'

D' → Z D' | Z Aq Aq D' | Z w D' | ε

Reformatting

M → B Aq | A

B → Y Aw

Y → D Z | w

D → w Aq Aq D' | A Aq Aq D' | w w D'

D' → Z D' | Z Aq Aq D' | Z w D' | ε

First	Follow
{Ali, Waqas}	{$}
{Waqas, Ali}	{Aqib}
{Waqas, Ali}	{Awais}
{Waqas, Ali}	{Zahid}
{Zahid, ε}	{Zahid}