Assignment Statements

$S \rightarrow id := E$ { p = lookup(id.name);
if p ≠ nil then
emit (p ':=' E.place)
else error }

$E \rightarrow E_1 + E_2$ { E.place := newtemp;
emit (E.place ':=' E_1.place + E_2.place) }

$E \rightarrow E_1 * E_2$ { E.place := newtemp;
emit (E.place ':=' E_1.place * E_2.place) }

$E \rightarrow - E_1$ { E.place := newtemp;
emit (E.place ':=' 'uminus' E_1.place) }

$E \rightarrow (E_1)$ { E.place := E_1.place }

E → id { p := lookup(id.name);
if (p ≠ nil) then
E.place := p
else error }

x := a * b + c * d - e * f

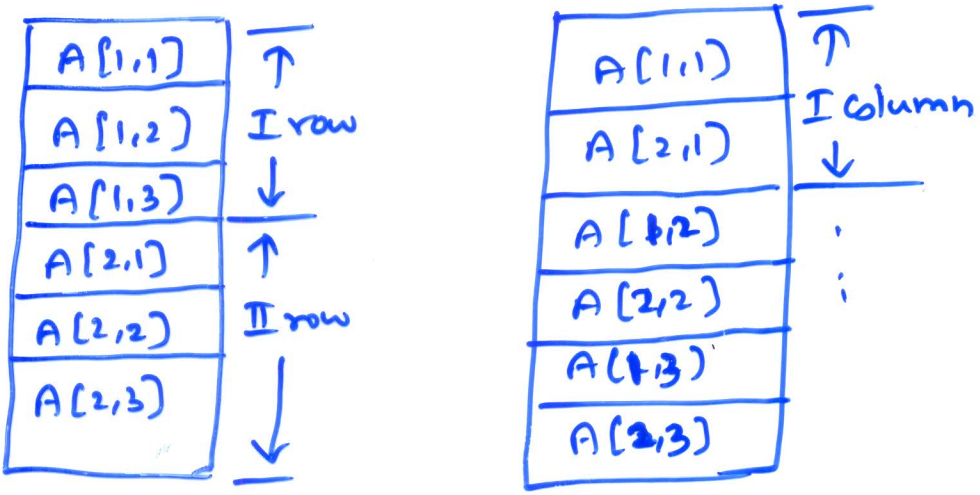
Statement	Value of c
	0
$0 := a * b	1
$1 := c * d	2
$0 := $0 + $1	1
$1 := e * f	2
$0 := $0 - $1	1
x := $0	0

Addressing Array elements

base + (i - low) xw

ixw + (base - low xw)

Row-major $A_{2\times3}$



2-D array stored in row-major form:

$$base + ((i_1 - low_1) \times n_2 + i_2 - low_2) \times w$$

$$((i_1 \times n_2) + i_2) \times w + (base - ((low_1 \times n_2) + low_2) \times w)$$

Relative addrss of $A[i_1, i_2, \cdots, i_k]$

$$((\cdots((i_1 n_2 + i_2) n_3 + i_3) \cdots) n_k + i_k) \times w +$$
$$base - ((\cdots((low_1 n_2 + low_2) n_3 + low_3) \cdots) n_k + low_k) \times w$$

$$\begin{cases} e_1 = i_1 \\ e_m = e_{m-1} \times n_m + i_m \end{cases}$$

l-value 'L' → L.place
→ L.offset

$S \rightarrow L := E$

$E \rightarrow E + E$

$E \rightarrow (E)$

$E \rightarrow L$

$L \rightarrow Elist\ ]$

$L \rightarrow id$

$E \rightarrow Elist, E$

$Elist \rightarrow id[E$

(1) $S \rightarrow L := E$ {if L.offset = null then
emit (L.place ':=' E.place);
else
emit(L.place '[' L.offset ']' ':=' E.place) }

(2) $E \rightarrow E_1 + E_2$ { E.place := newtemp;
emit(E.place ':=' E_1.place '+' E_2.place) }

(3) $E \rightarrow (E_1)$ {E.place := E_1.place}

(4) $E \rightarrow L$ {if L.offset = null then
E.place := L.place
else
E.place := newtemp;
emit(E.place ':=' L.place '[' L.offset ']')
end }

(5) $L \rightarrow Elist\ ]$ { L.place := newtemp;
L.offset := newtemp;
emit(L.place ':=' c(Elist.array));
emit(L.offset ':=' Elist.place '*'
width(Elist.array))}

(6) $L \rightarrow id$ {L.place := id.place
L.offset := null}

(7) Elist → Elist, E { t := newtemp;
m := Elist, .ndim + 1;
emit(t ':=' Elist, .place '*'
limit(Elist, .array, m));
emit(t ':=' t '+' E.place);
Elist.array := Elist, .array;
Elist.place := t;
Elist.ndim := m; }

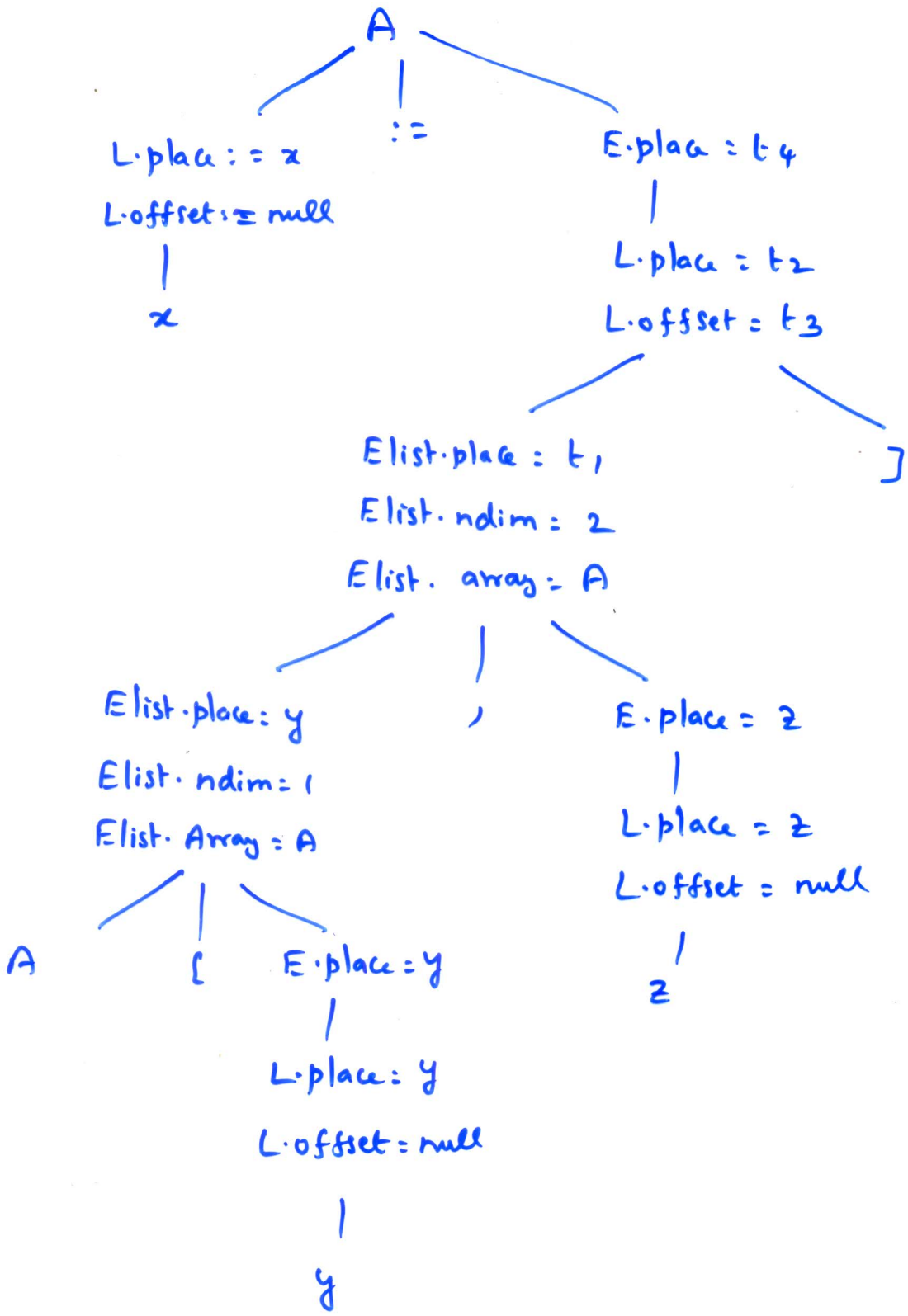
(8) Elist → id [E { Elist.array := id.place;
Elist.place := E.place
Elist.ndim := 1 }

Example: A → 10 × 20 array $low_1 = low_2 = 1$

$n_1 = 10$ $n_2 = 20$ $w = 4$

x := A[y, z]

$t_1 := y * 20$
$t_1 := t_1 + 2$
$t_2 := c$
$t_3 := 4 * t_1$
$t_4 := t_2[t_3]$
$x := t_4$



Type Conversions within assignments

$E \rightarrow E + E$ { E.type := if E_1.type = integer and E_2.type = integer then integer else real }

```
E.place := newtemp;
if E1.type = integer and E2.type = integer then
        emit( E.place ':=' E1.place 'int+' E2.place)
        E.type := integer
end
else if E1.type = real and E2.type = real then
    emit( E.place ':=' E1.place 'real+' E2.place)
    E.type := real
end
else if E1.type = integer and E2.type = real then
        u := newtemp;
        emit( u ':=' 'inttoreal' E1.place);
        emit( E.place ':=' u 'real+' E2.place);
        E.type := real
end
else if E1.type = real and E2.type = integer then
        u := newtemp;
        emit( u ':=' 'inttoreal' E2.place);
        emit(E.place ':=' E1.place 'real+' u);
        E.type := real;
end
else
        E.type := type_error;
```

```
x := y + i * j          x, y → real
                        i, j → int

t1 := i int* j
t3 := inttoreal t1
t2 := y real+ t3
x := t2
```