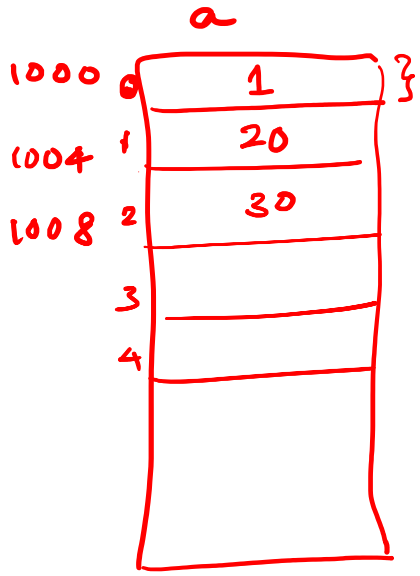
SDT for Arrays

Addressing array elements

- Array can be easily accessed if the elements are consecutive locations
- width = w, ith element = base + (i –low) * w –
- Base – relative address of A[low]
- i *w + (base – low * w)



$i * 4$

8 bytes

$i * w$

base

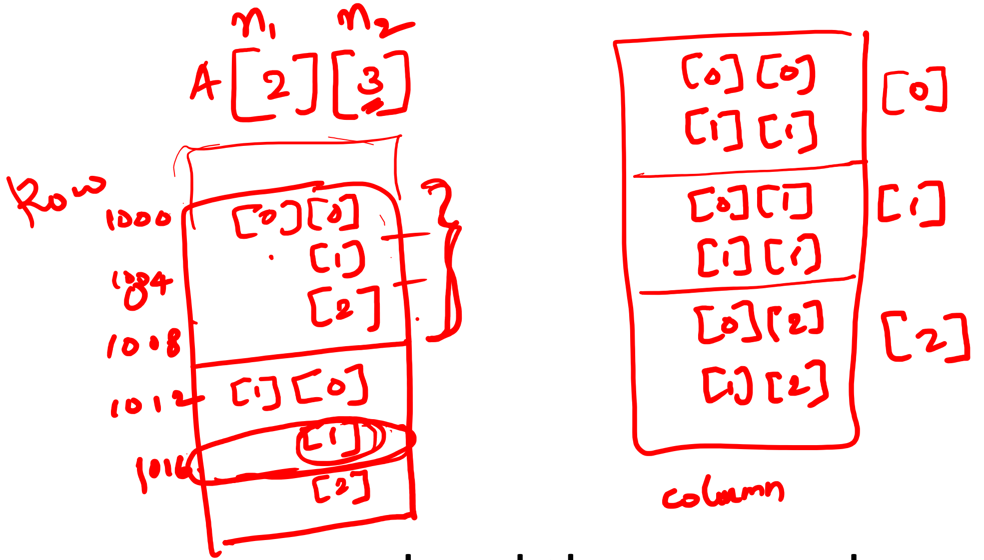
a[2]

1000 + 2 * 4 = 1008

Addressing of array elements

- i * w – evaluated during compile time
- c = base – low * w is evaluated during run time
- A[i] – address = i *w +c

Arrays



- Row major
  - Accessed row by row
  - For 2 d arrays – for every row, all the columns are accessed and then second row is accessed
- Column major
  - Accessed column by column

$$A[1][1] \quad (i = 1,\ j = 1)$$

$$1000 + [(1 \times 3) + 1]\, 4 = 1016$$

$$base + [(i \times n_2) + j] * w$$

$$base + [(i - low) \times n_2 + j] * w$$

Addressing Array Elements: Multi-Dimensional Arrays

```
A : array [1..2,1..3] of integer;
```

$low_1 = 1$, $low_2 = 1$, $n_1 = 2$, $n_2 = 3$, $w = 4$

$base_{\mathtt{A}}$
`A[1,1]`
`A[1,2]`
`A[1,3]`
`A[2,1]`
`A[2,2]`
`A[2,3]`

Row-major

$base_{\mathtt{A}}$
`A[1,1]`
`A[2,1]`
`A[1,2]`
`A[2,2]`
`A[1,3]`
`A[2,3]`

Column-major

2d Array

- Row major
  - base + ((i1-low1) * n2) + i2 – low2) * w
  - ((i1*n2) + i2) * w +

    (base – ((low1 * n2) + low2) * w)

base

3D

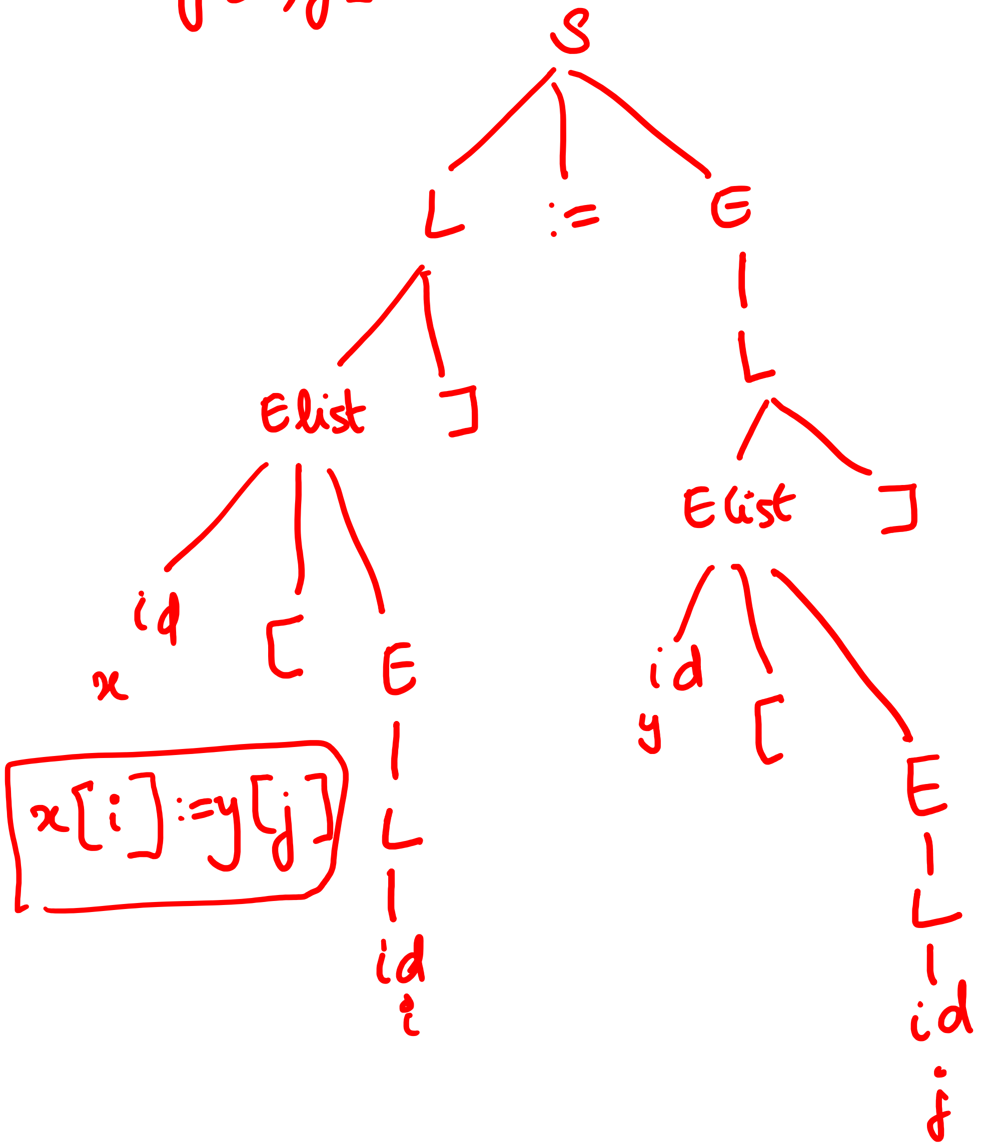
$(((i1 * n_3) + i2) n_2 + i_3) * w$

Grammar for Array

$x := y[i]$ $\quad$ $x[i] := y$ $\quad$ $\boxed{x[i] := y[j]}$

$x := y[i, j]$

- S → L : = E
- E → E + E | (E)
- E → L
- L → Elist]
- L → id
- Elist → Elist, E
- Elist → id [E



Array

- Array variable L has two attributes
  - place – ptr to symbol table
  - offset – to move through the array index
- A n-dimension array can be generalized using the recursive expression
  - $e_m = e_{m-1} + i_m$ recursive
  - $e_1 = i_1$

Functions

$A[2][3]$

$j = 2$

- Elist.ndim – records the number of dimensions in the Elist
- limit(array, j) – returns nj number of elements in the jth dimension of the array
- Elist.place – temporarily hold a value from index expression

Addressing Array Elements: Multi-Dimensional Arrays

```
A : array [1..2,1..3] of integer;
```
(Row-major)

```
… := A[i,j]
```

$= base_{\mathtt{A}} + ((i_1 - low_1) * n_2 + i_2 - low_2) * w$

$= ((i_1 * n_2) + i_2) * w + c$

where $c = base_{\mathtt{A}} - ((low_1 * n_2) + low_2) * w$

with $low_1 = 1$; $low_2 = 1$; $n_2 = 3$; $w = 4$

```
t1 := i * 3
t1 := t1 + j
t2 := c   // c = baseA - (1 * 3 + 1) * 4
t3 := t1 * 4
t4 := t2[t3]
…  := t4
```

Semantic rules

$t1 := t2$

$t2[t3] := t_1$

Production	Semantic rule	Inference
S → L : = E	{ if L.offset = null, then emit (L.place ':=' E.place) else emit (L.place'['L.offset']' '=' E.place)}	Checks if the lefthand side variable is an array or not using the offset information and then considers it as an array or a simple variable.
E → L	{if L.offset = null emit(E.place '=' L.place) else E.place = newtemp() t2 emit (E.place ':=' L.place'['L.offset']'}	LHS non-terminal is for array. Hence similar to the previous one

$t2 := t1[t3]$

Semantic Rules for arrays

Production	Semantic rule	Inference
L → Elist]	{L.place = newtemp; L.offset = newtemp; emit(L.place ':=' c(Elist.array) emit (L.offset ':=' Elist.place * width(Elist.array)}	c(Elist.array) returns the base address of the array and Elist.place returns the dimension of the array
L → id	{L.place := id.place; L.offset := null}	Final termination of L with id, where the address of id is used as the place of L and hence offset is set null
E → E1+E2	Same as earlier	
E → (E1)	Same as earlier	

A[i, j] k

$t := i * n_2$

$\boxed{t := t + j}$

Semantic rule

Production	Semantic rule	Inference
Elist → Elist1, E	{t := newtemp; m:= Elist1.ndim+1 2 emit (t ‘:=‘ Elist1.place ‘*’ (limit(Elist1.array, m)); emit (t ‘:=‘ t ‘+’ E.place); Elist.array := Elist1.array; Elist.place := t Elist.ndim : = m	The recursive expression is evaluated here. To start with number of dimensions of Elist1 is taken. This production implies a multi dimension array. The first one computes the offset of the first dimension, the last three lines carry forward this to incorporate multiple dimensions

array [i,j]

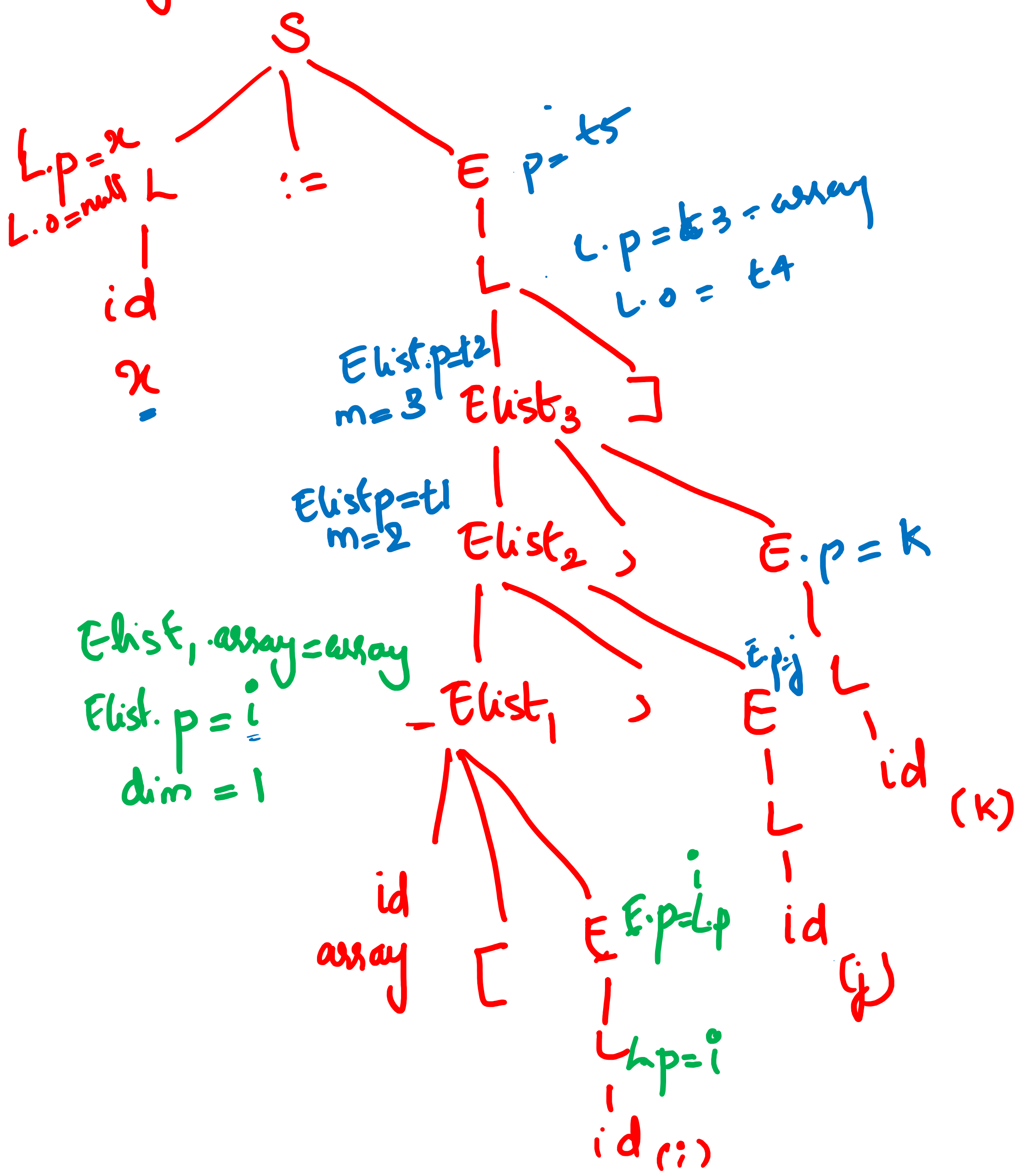
A[i,j]

Production	Semantic rule	Inference
Elist → id [E	{Elist.array := id.place; A Elist.place := E.place; i Elist.ndim: = 1;}	c(Elist.array) returns the base address of the array and Elist.place returns the dimension of the array

array [3,2,4]

$x := array[i,j,k]$

integer



$t_1 := i * 2$

$t_1 := t_1 + j$

$t_2 := t_1 * 4$

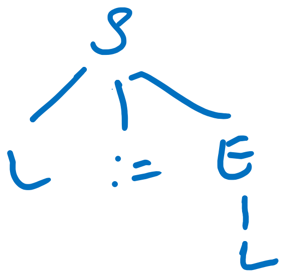
$t_2 := t_2 + k$

$t_3 := array$

$t_4 := t_2 * 4$ ← w

$t_5 := t_3[t_4]$

$x := t_5$



Example

- A – 10 x 20 array
- n1 = 10, n2 = 20
- W = 4
- x : = A[y,z]

Example

- First production S → L: = E
- L → id
- E → L
- L → Elist]
- Elist → Elist1, E
- Elist1 → id [E

Rules application

- t1 : = y * 20
- t1 : = t1 + z
- t2 := c
- t3 := 4 * t1
- t4 := t2 [t3]
- x := t4

Type conversions

- E → E1 + E2

{E.type = if E1.type = int &

E2.type = int then int; else real}

E.place = newtemp ()

If E1.type = int and E2.type = int

emit (E.place ‘:=‘ E1.place ‘int +’

E2.place)