Syntax Tree to Three Address Code Generation

SDD

$S \rightarrow id = E ;$ $S.code = E.code \,||\, gen(top.get(id.lexeme) '=' E.addr)$

$E \rightarrow E_1 + E_2 ;$ $E.addr = new\ Temp()$

$E.code = E_1.code \,||\, E_2.code \,||\, gen(E.addr '=' E_1.addr '+' E_2.addr)$

$\rightarrow - E_1 ;$ $E.addr = new\ Temp()$

$E.code = E_1.code \,||\, gen(E.addr '=' 'minus' E_1.addr)$

$\rightarrow (E_1) ;$ $E.addr = E_1.addr$

$E.code = E_1.code$

$\rightarrow id ;$ $E.addr = top.get(id.lexeme)$

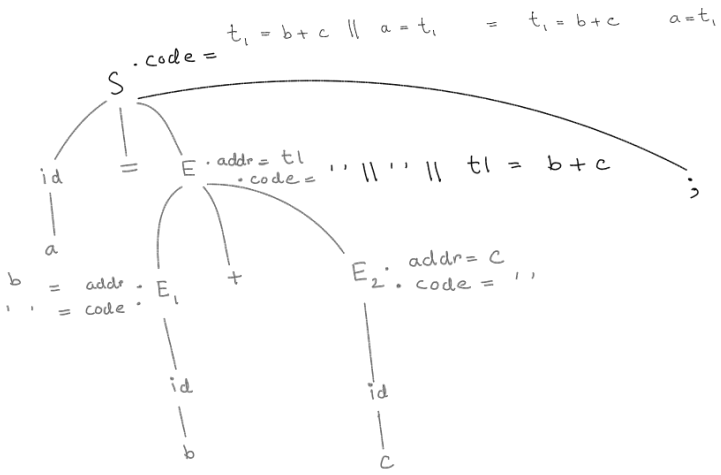
$E.code = ''$

input string: a = b + c ;

t1 = b + c

a = t1

$\Rightarrow$



Relative Address Calculation for 1D array

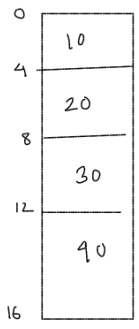
arr

10	20	30	40
0	1	2	3

- Datatype = integer.
- array base address is 0.
- array for each integer 4 byte of memory will be allocated.
- ২ ধরনের way তে memory তে array রাখতে পারি.
  - row major
  - column major

  } 1 dimensional এর জন্য ২ টা same.

row major representation:



0 1 2 3 → 10 | 4 5 6 7 → 20 | 8 9 10 11 → 30 | 12 13 14 15 → 40

relative address for 1 dimensional array = base address + (index * byte_size)

∴ arr [3] address = 0 + (3 * 4) = 12.

Relative Address Calculation for Multidimensional Array :

- 5 * 5 array.

arr →

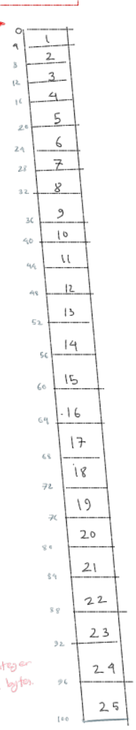
	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

The datatype can be mixed matched like both integer & float

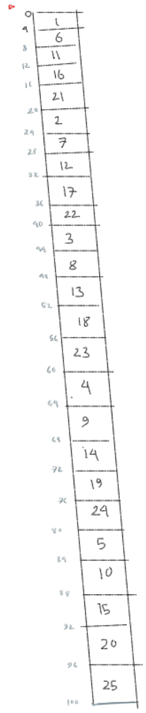
Most of the time we will calc the memory address wrt to row major.

arr [1] [3] = 9 (32)

row major (row এর value আগে fillup)



Column major → (column value আগে fillup)



per row 5 elements

each element is integer which takes 4 bytes

প্রত্যেকটি row এর size = (5 * 4) byte

= 20 byte.

∴ each row size = 20 byte.

(20 * 1) + (3 * 4) = 32.

↳ (row_size * row index) + (Column index * byte_size)

base address থেকে 32 distance এ যেতে লাগবে

relative memory address for 2D array = base_address + (row_index * row_size) + (Column_index * byte_size)

SDD

S → id = E ; { gen (top.get (id.lexeme) '=' E.addr) ; }

| L = E ; { gen (L.addr.base '[' L.addr ']' '=' E.addr) ; }

E → $E_1 + E_2$ { E.addr = new Temp ( ) ;

gen (E.addr '=' E_1.addr '+' E_2.addr) ; }

| id { E.addr = top.get (id.lexeme) ; }

| L { E.addr = new Temp ( ) ;

gen (E.addr '=' L.array.base '[' L.addr ']') ; }

L → id [E] { L.array = top.get (id.lexeme);

L.type = L.array.type.elem ;

L.addr = new Temp ( ) ;

gen (L.addr '=' E.addr '*' L.type.width) ; }

| L_1 [E] { L.array = L_1.array ;

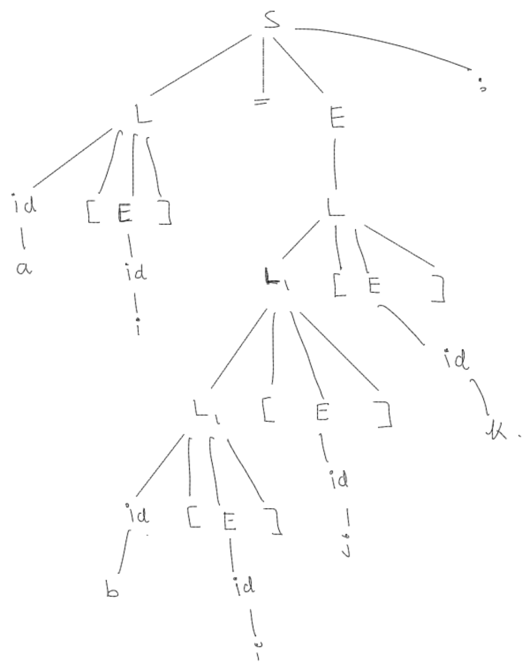
L.type = L_1.type.elem ;

t = new Temp ( ) ;

L.addr = new Temp ( ) ;

gen (t '=' E.addr '*' L.type.width) ; }

gen (L.addr '=' L_1.addr '+' t) ; }



input string : a [i] = b [i] [j] [k]