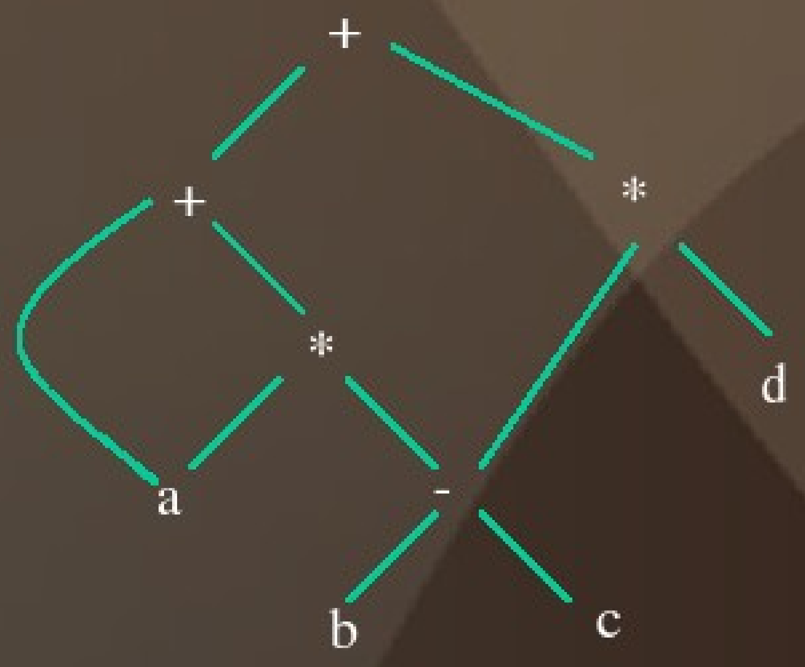
Lecture #23

Semantic Analysis Continued…

Three Address Code

- In a three address code there is at most one operator at the right side of an instruction
- Example:



t1 = b – c
t2 = a * t1
t3 = a + t2
t4 = t1 * d
t5 = t3 + t4

- *Linearised presentation of AST or DAG*
- *Explicit names given to interior nodes of the graph*

Three Address Instruction Forms

- x = y op z
- x = op y
- x = y
- goto L
- if x goto L and ifFalse x goto L
- if x relop y goto L
- Procedure calls using:
    - param x
    - call p,n
    - y = call p,n
- x = y[i] and x[i] = y
- x = &y and x = *y and *x =y

do i = i+1; while (a[i] < v);

Example

- do i = i+1; while (a[i] < v);

```
L:  t1 = i + 1
    i = t1
    t2 = i * 8
    t3 = a[t2]
    if t3 < v goto L
```

Symbolic labels

```
100:    t1 = i + 1
101:    i = t1
102:    t2 = i * 8
103:    t3 = a[t2]
104:    if t3 < v goto 100
```

Position numbers

Example

- b * minus c + b * minus c

Three address code

t1 = minus c
t2 = b * t1
t3 = minus c
t4 = b * t3
t5 = t2 + t4
a = t5

Quadruples

op	arg1	arg2	result
minus	c		t1
*	b	t1	t2
minus	c		t3
*	b	t3	t4
+	t2	t4	t5
=	t5		a

Triples

	op	arg1	arg2
0	minus	c	
1	*	b	(0)
2	minus	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)

Indirect Triples

	op
35	(0)
36	(1)
37	(2)
38	(3)
39	(4)
40	(5)

	op	arg1	arg2
0	minus	c	
1	*	b	(0)
2	minus	c	
3	*	b	(2)
4	+	(1)	(3)
5	=	a	(4)

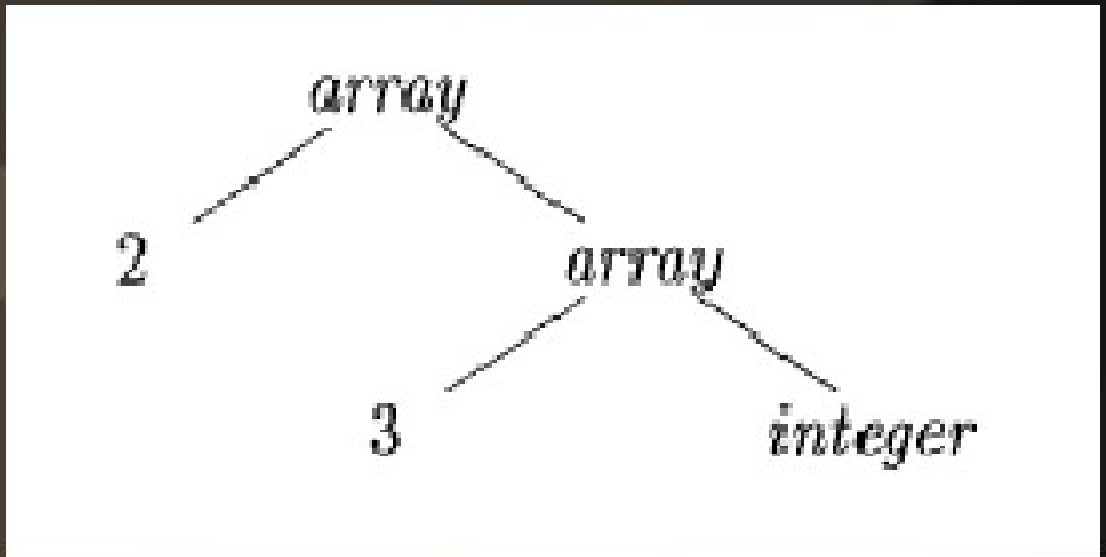
Type Equivalence

- They are the same basic type.
- They are formed by applying the same constructor to structurally equivalent types.
- One is a type name that denotes the other.

Type Expressions

Example: int[2][3]

array(2,array(3,integer))



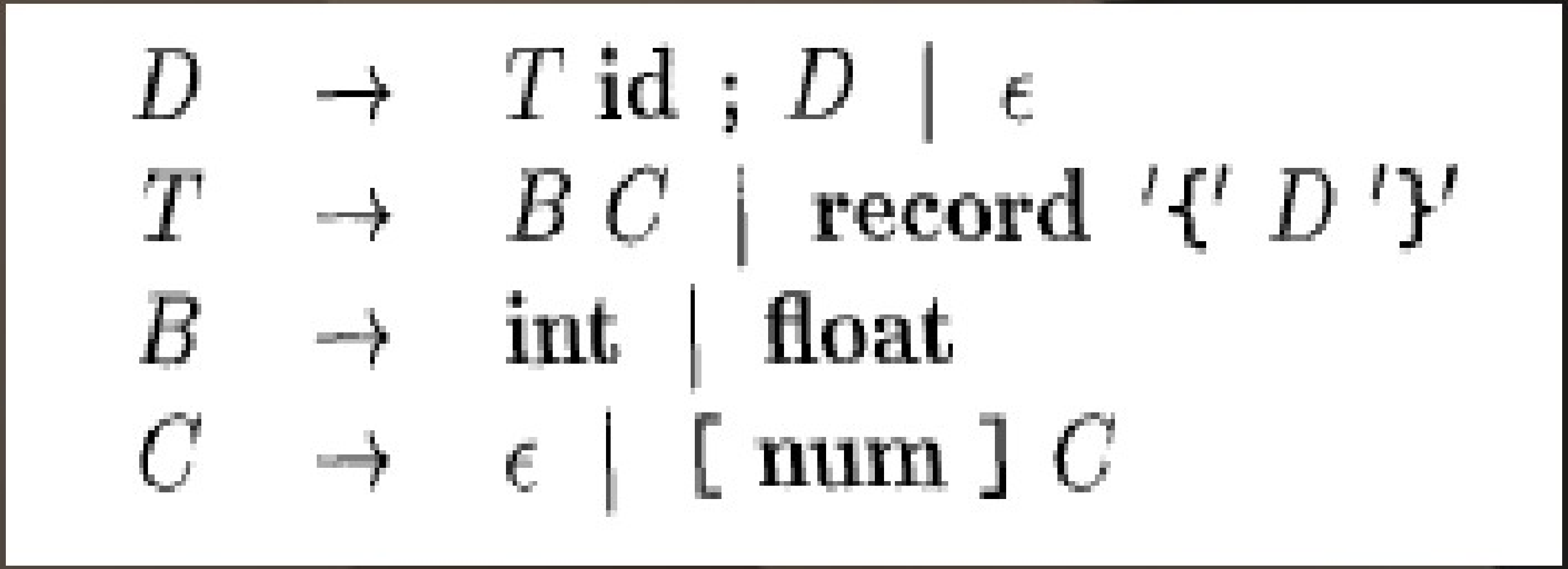
- Type of a language construct is denoted by a type expression
- It is either a basic type or it is formed by applying operators called *type constructor* to other type expressions
- A type constructor applied to a type expression is a type expression
- A basic type is type expression
  - *type error* : error during type checking
  - *void* : no type value

Type Expressions

A basic type is a type expression

- A type name is a type expression
- A type expression can be formed by applying the array type constructor to a number and a type expression.
- A record is a data structure with named field
- A type expression can be formed by using the type constructor → for function types
- If s and t are type expressions, then their Cartesian product s*t is a type expression
- Type expressions may contain variables whose values are type expressions

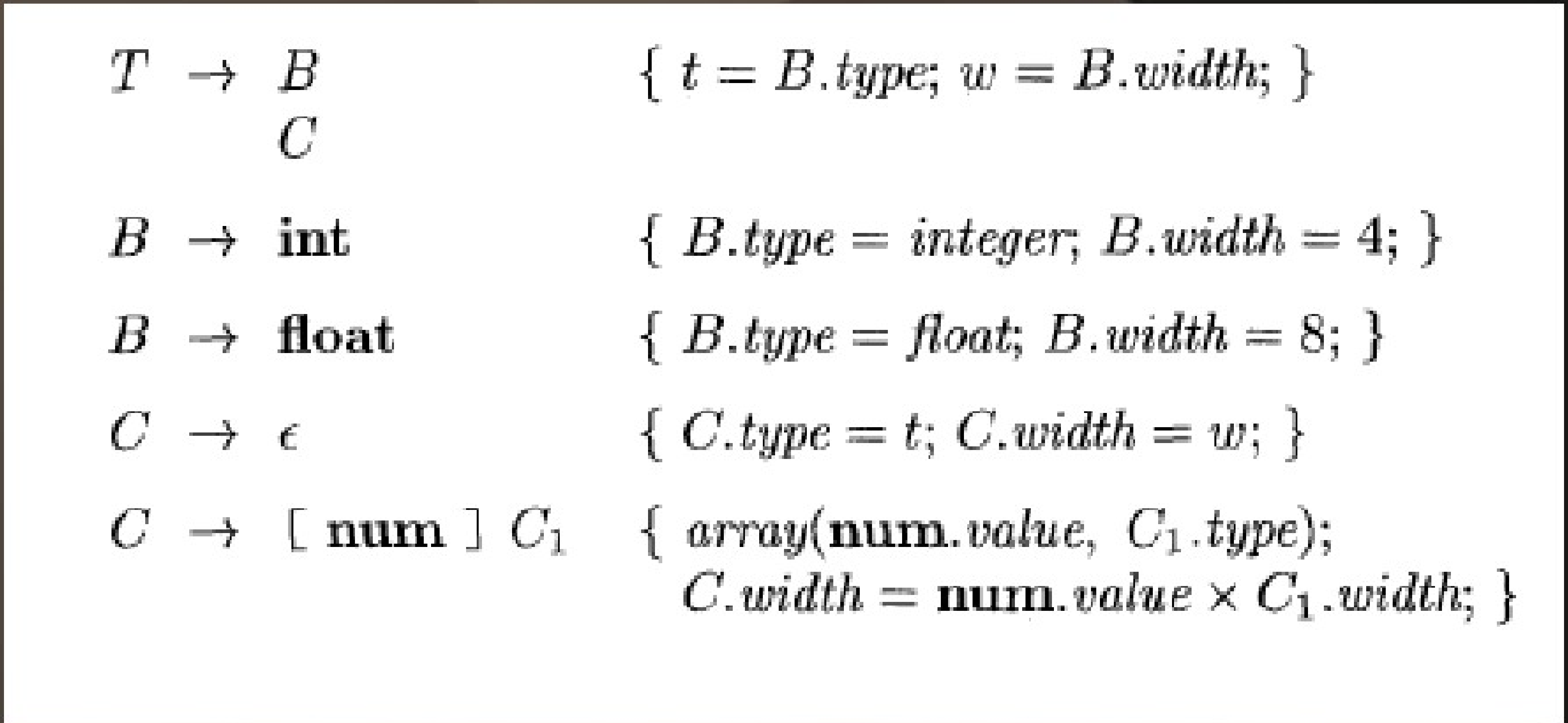
Declarations



$$\begin{array}{lcl} D & \rightarrow & T\ \textbf{id}\ ;\ D \ \mid\ \epsilon \\ T & \rightarrow & B\ C\ \mid\ \textbf{record}\ '\{'\ D\ '\}' \\ B & \rightarrow & \textbf{int}\ \mid\ \textbf{float} \\ C & \rightarrow & \epsilon\ \mid\ [\ \textbf{num}\ ]\ C \end{array}$$

Storage Layout for Local Names

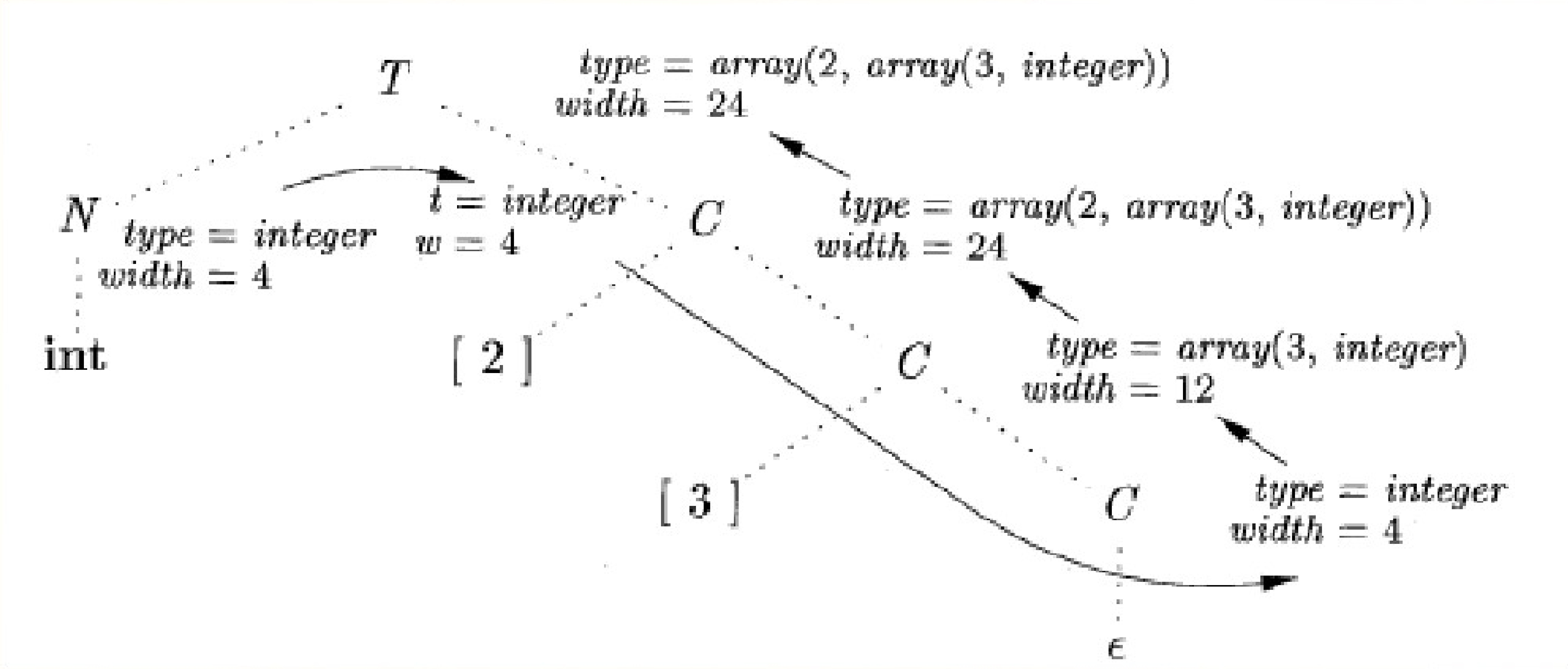
- Computing types and their widths



$T \rightarrow B$	$\{\ t = B.type;\ w = B.width;\ \}$
$\ C$	
$B \rightarrow$ **int**	$\{\ B.type = integer;\ B.width = 4;\ \}$
$B \rightarrow$ **float**	$\{\ B.type = float;\ B.width = 8;\ \}$
$C \rightarrow \epsilon$	$\{\ C.type = t;\ C.width = w;\ \}$
$C \rightarrow [$ **num** $]\ C_1$	$\{\ array(\mathbf{num}.value,\ C_1.type);$ $C.width = \mathbf{num}.value \times C_1.width;\ \}$

Storage Layout for Local Names

- Syntax-directed translation of array types


T
type = array(2, array(3, integer))
width = 24
N
type = integer
width = 4
int
t = integer
w = 4
C
type = array(2, array(3, integer))
width = 24
[2]
C
type = array(3, integer)
width = 12
[3]
C
type = integer
width = 4
ϵ