→ basic type (int, float for this grammar)

→ composite variable

{T → B { t = B.type ; w = B.width ;} C

↳ temporary variables

↳ dummy non-terminal

} { T → B X C

X → ε { t = B.type , w = B.width }

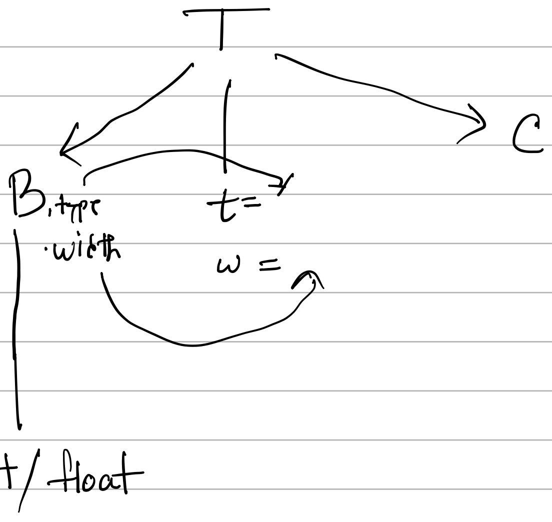
T → BC

B → int

B → float

C → ε

C → [num] C_1



* annotated parse tree
* Dependency graph.

32 bit machine take 4 bytes to store int

8 " " " float

C → ε হয় তাহলে C এর type হবে basic type (B)

কারণ, এখানে t , w এর value store করি যখন C → ε হয়

মানে C এর expansion শেষ হয়।

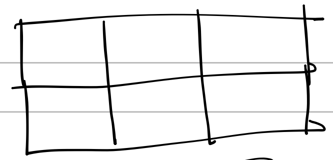
array (num.value , C_1.type)

↓ index/ element count

↳ gets from C in body

* int [2][3];

↳ 2 rows, 3 columns →



6 elements

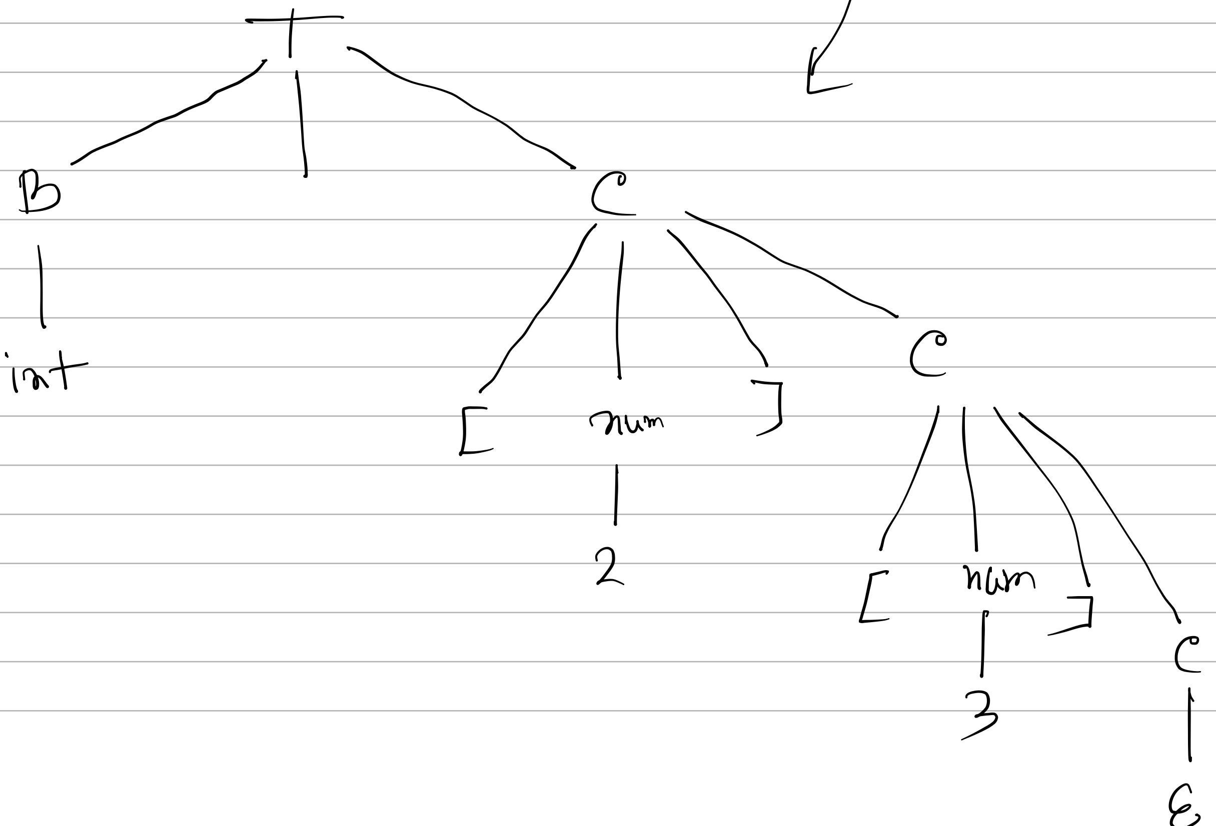
∴ $6 \times 4 = 24$ bytes needed.

This is not S attributed since (শুধু synthesize attribute নিয়ে আলোচনা আছে)

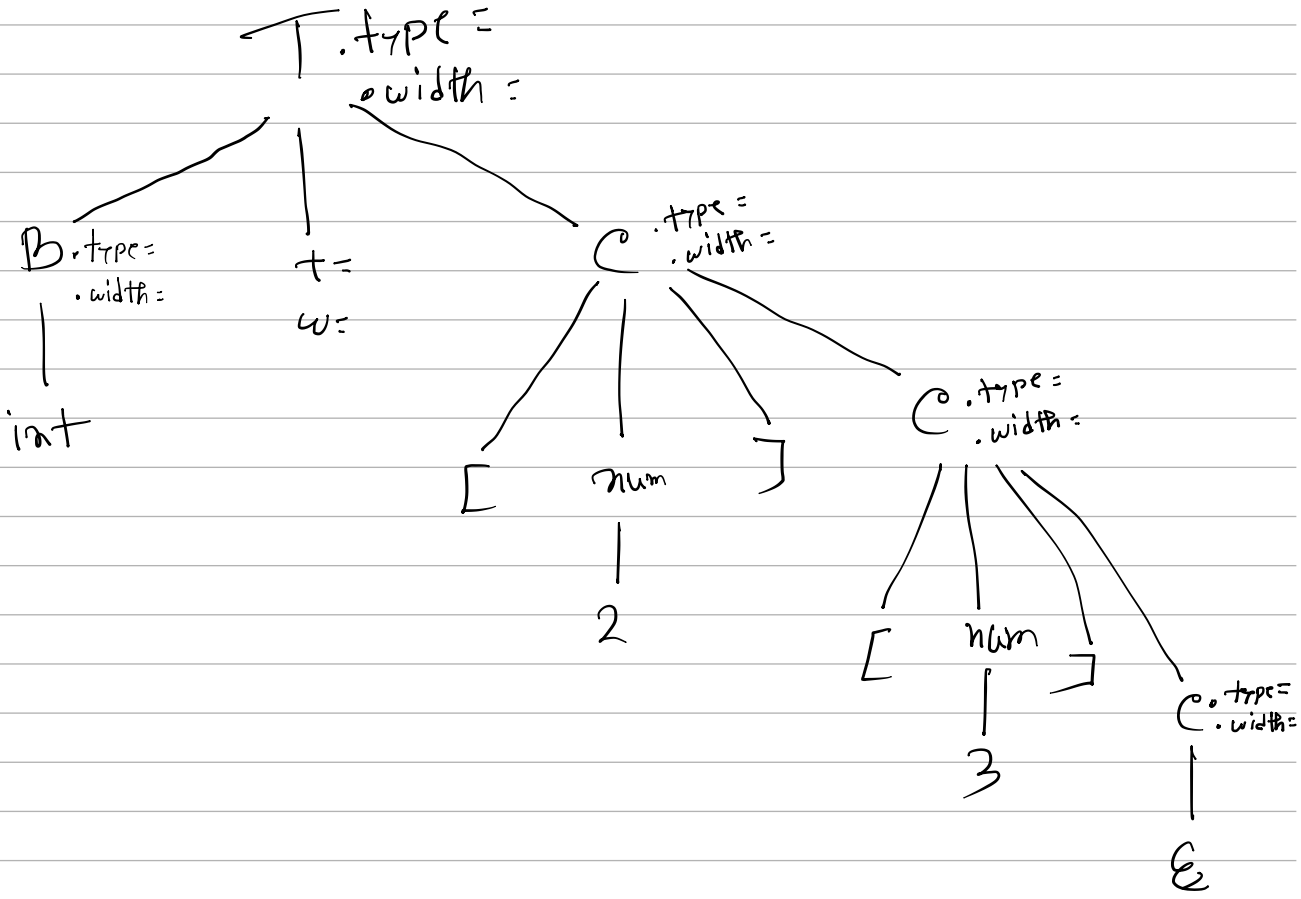
" " " L " " (L to right এখন sibling এর থেকে নেয়া না.)

* int [2][3]

step 1: draw the tree

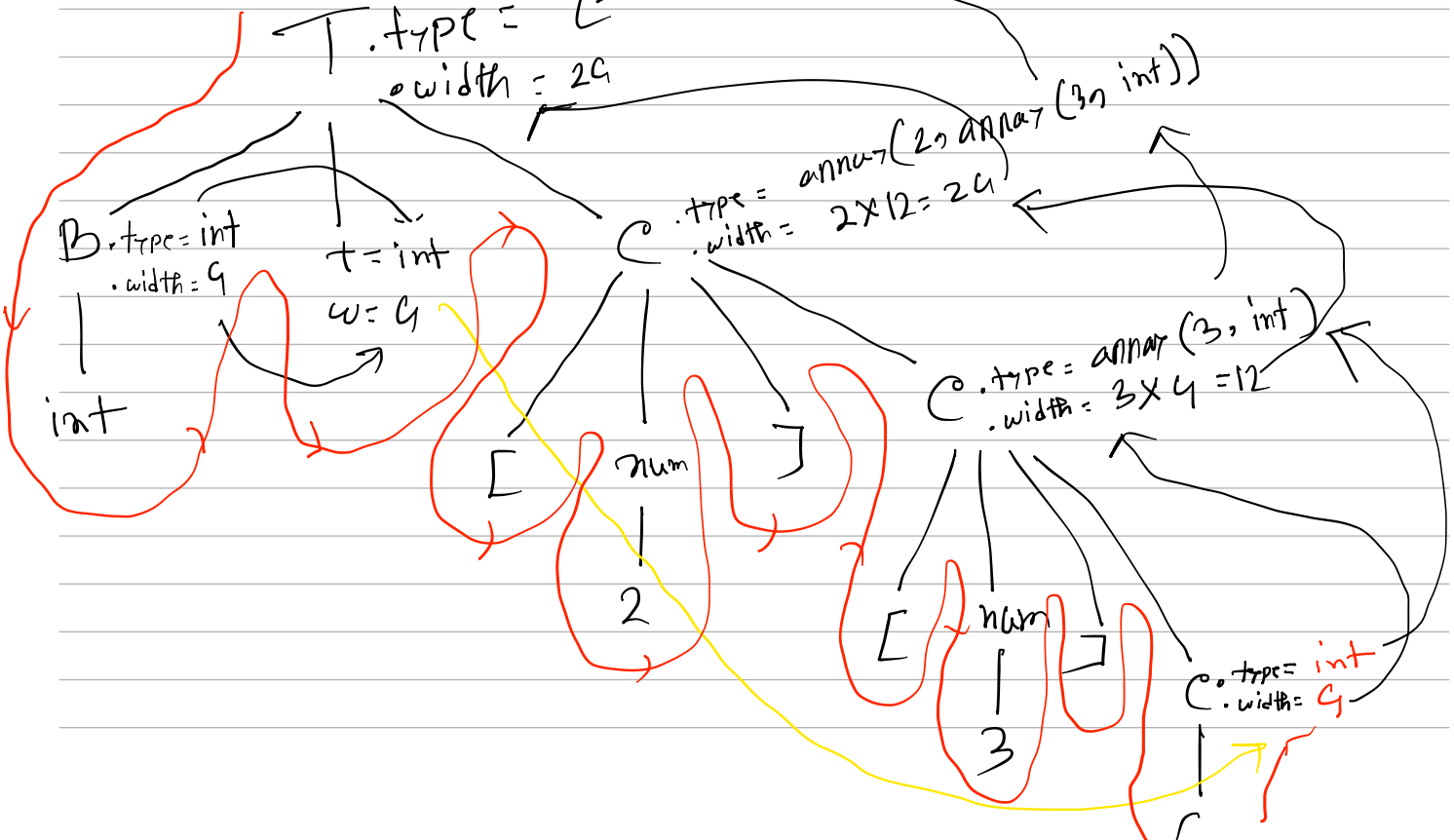


Step 2: Attribute writing



Step 3: DFS

C.type = array(num.value, C1.type)



dependency graph এরকম arrow দিতে হবে যেন

arrow edge গুলো যাতে intersect করতে পারবে না,

তবে edge (.....) line এ দিতে হবে।

★ T → record '{' D '}'

record is a structure which can store declaration of multiple variables.

```
record {
    int a;
    float b;
}
```

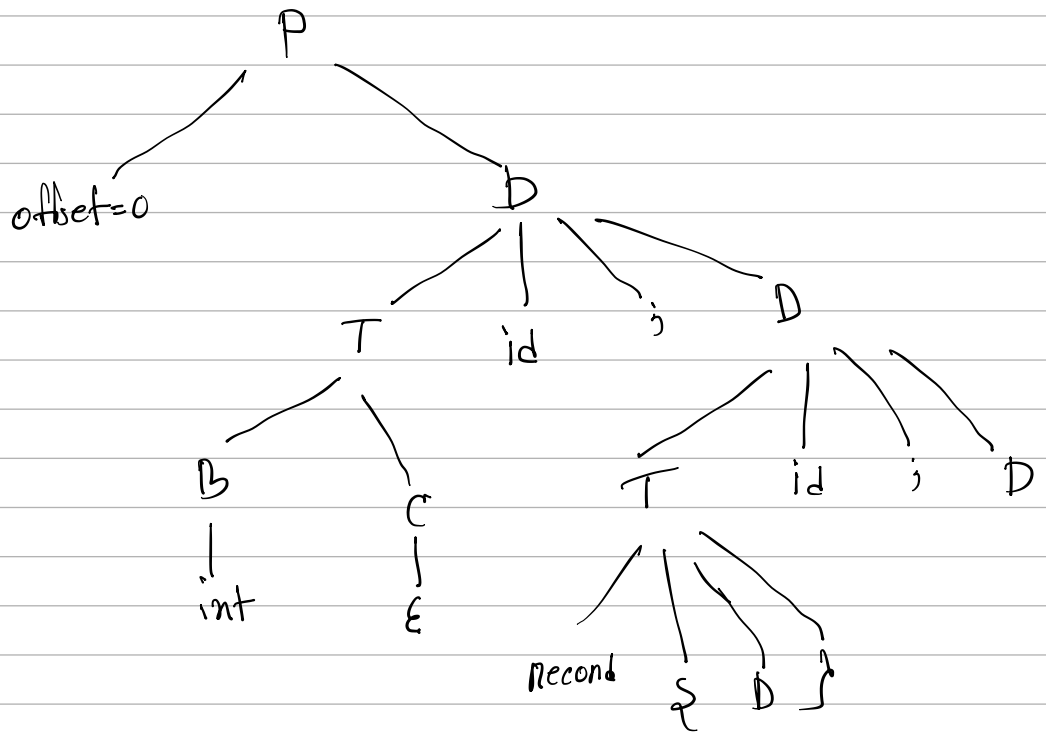
★ নতুন scope এ গেলে offset change হবে তাই আগের offset stack এ push করে,

P → {offset = 0;} D

D → T id ; {top.put(id.lexeme, T.type, offset);
offset = offset + T.width;}

D1 ← এই D এর মধ্যে আগের দিয়ে আবার করতে হবে তাই এভাবে লেখা।

$D \rightarrow \varepsilon$



int a ; float b ;