Compiler Design (CS335), Spring 2024
Indian Institute of Technology Kanpur
Homework Assignment Number 3

Student Name: Siddhant Jakhotiya
Roll Number: 211030
Date: April 5, 2024

QUESTION 1

1 Annotated parse tree for the given string

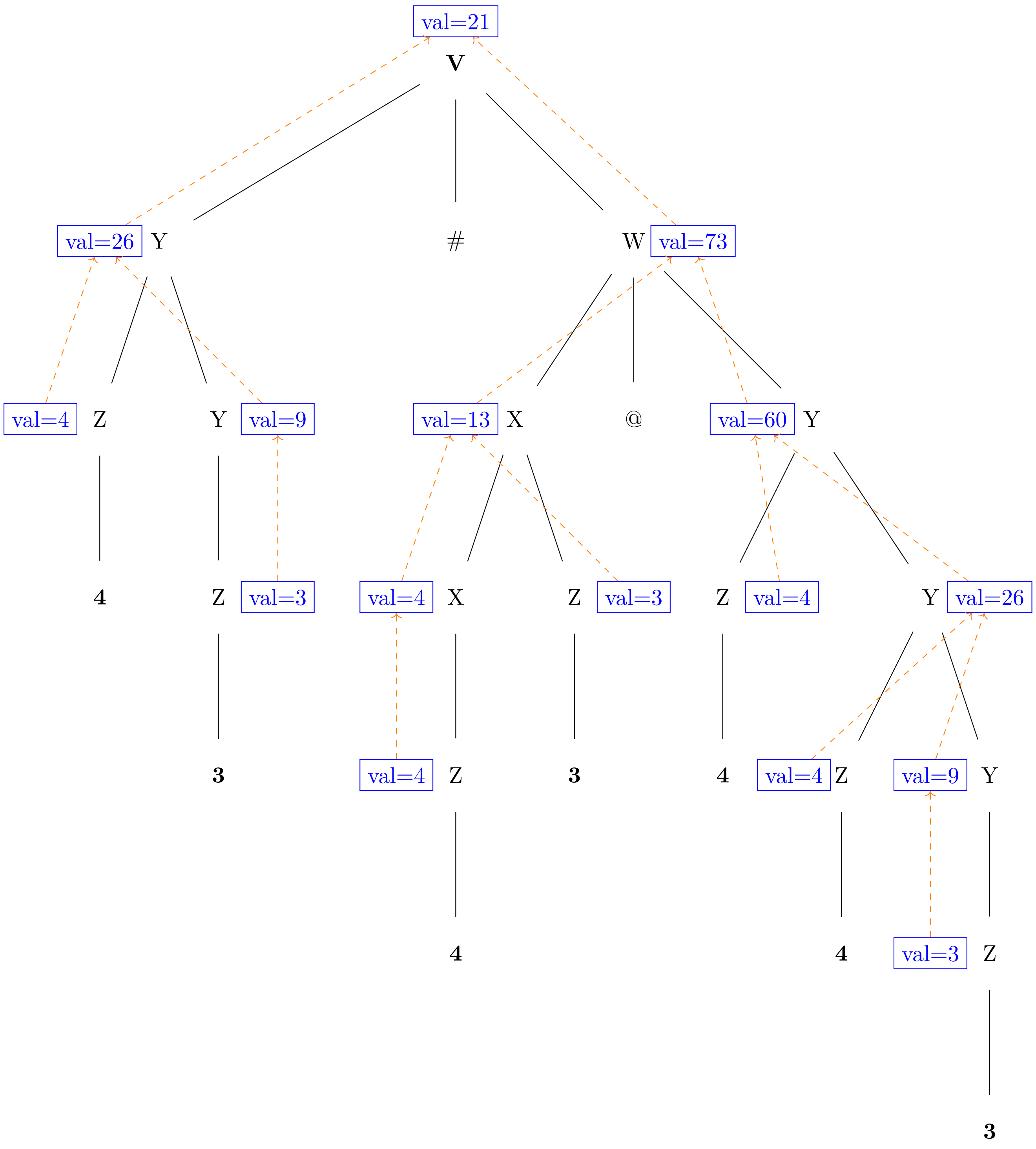

Figure 1: Annotated parse tree for 43#43@443

2 Value of V

As can be seen from the annotated parse tree, the value at V for the string 43#43@443 is **21**.

3 Grammar: S-attributed or L-attributed?

The given grammar is **S-attributed** as the attribute *val* for each non-terminal V, X, Y, Z is synthesized (i.e. depends only on the body of their respective productions). Note that every S-attributed grammar is also L-attributed so it can also be said that the grammar is also L-attributed.

Compiler Design (CS335), Spring 2024
Indian Institute of Technology Kanpur
Homework Assignment Number 3

Student Name: Siddhant Jakhotiya
Roll Number: 211030
Date: April 5, 2024

QUESTION
2

1 Annotated parse tree for the given expression

Note: to avoid cluttering the diagram, the square brackets '[' and ']' are not shown as separate nodes but together in the same node with the non-terminal symbol E.

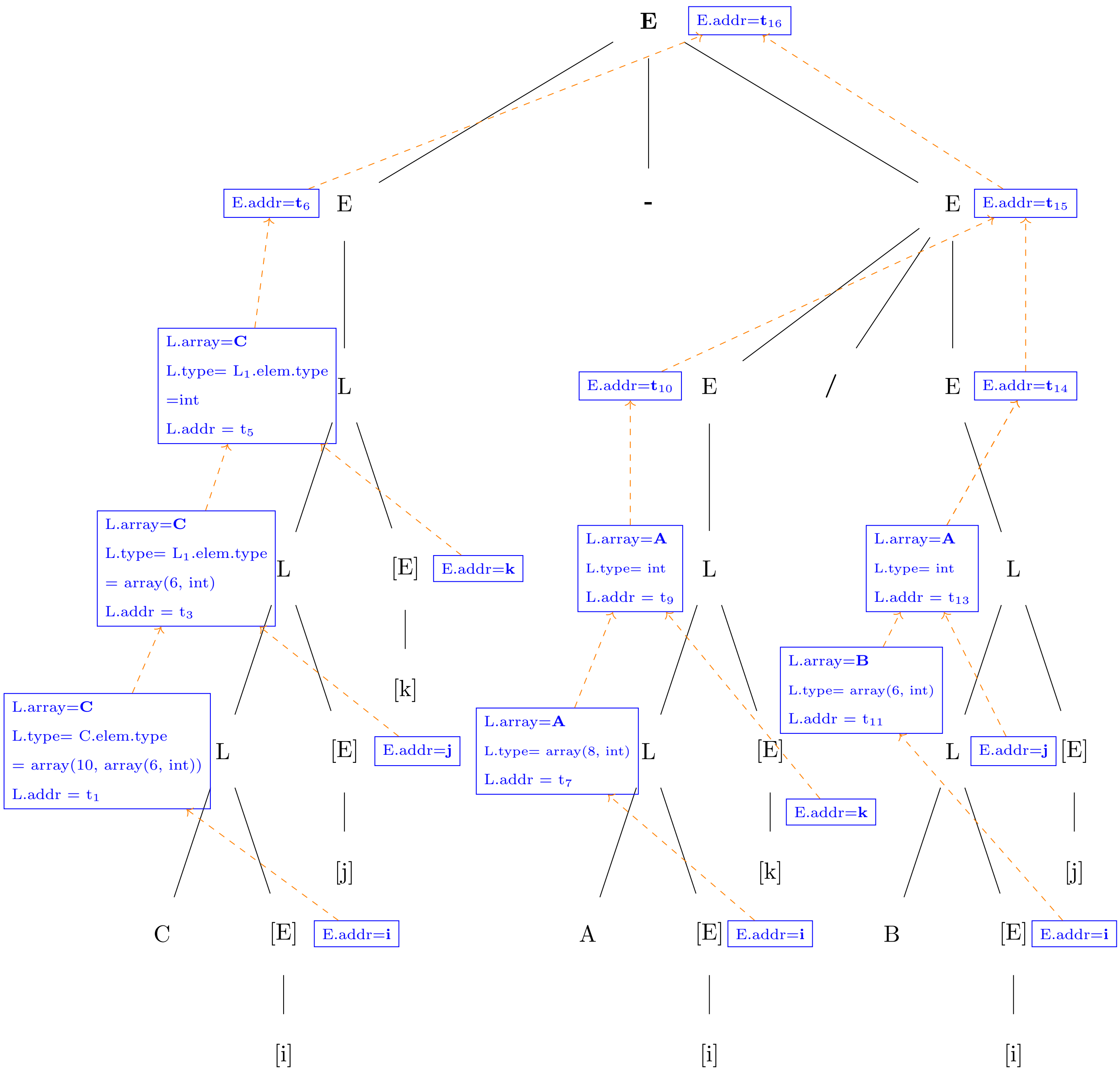

Figure 2: Annotated parse tree for $C[i][j][k] - A[i][k]/B[i][j]$

2 3AC code sequence generated for the expression

Based on the semantic actions, the 3AC code sequence generated for the given expression are given below. Note that although the labels aren't added by the semantic actions, they have been written for clarity.

Label	Code	Semantic Action
1:	t_1 = i * 240	L.addr "=" E.addr "*" L.type.width
2:	t_2 = j * 24	t "=" E.addr "*" L.type.width
3:	t_3 = t_1 + t_2	L.addr "=" L1.addr "+" t
4:	t_4 = k * 4	t "=" E.addr "*" L.type.width
5:	t_5 = t_3 + t_4	L.addr "=" L1.addr "+" t
6:	t_6 = C[t_5]	E.addr = L.array.base"["L.addr"]"
7:	t_7 = i * 32	L.addr "=" E.addr "*" L.type.width
8:	t_8 = k * 4	t "=" E.addr "*" L.type.width
9:	t_9 = t_7 + t_8	L.addr "=" L1.addr "+" t
10:	t_{10} = A[t_9]	E.addr "=" L.array.base"["L.addr"]"
11:	t_{11} = i * 24	L.addr "=" E.addr "*" L.type.width
12:	t_{12} = j * 4	t "=" E.addr "*" L.type.width
13:	t_{13} = t_{11} + t_{12}	L.addr "=" L1.addr "+" t
14:	t_{14} = B[t_{13}]	E.addr "=" L.array.base"["L.addr"]"
15:	t_{15} = t_{10}/t_{14}	E.addr "=" E1.addr"/"E2.addr
16:	t_{16} = t_6 - t_{15}	E.addr "=" E1.addr"-"E2.addr

Compiler Design (CS335), Spring 2024
Indian Institute of Technology Kanpur
Homework Assignment Number 3

Student Name: Siddhant Jakhotiya
Roll Number: 211030
Date: April 5, 2024

QUESTION 3

1 Semantic Actions for the proposed translation

```
S → id = E             gen(symtop.get(id .lexeme) "=" E.addr)
S → L = E              if(L.array = null) then
                         gen(L.addr "=" E.addr)
                       else
                         gen(L.array.base"["L.addr"]" "=" E.addr)
E → E1 + E2            E.addr =  new Temp(); gen(E.addr "=" E1.addr "+" E2.addr)
E → L                  if(L.array = null) then
                         E.addr = L.addr
                       else
                         E.addr = new Temp(); gen(E.addr "=" L.array.base"["L.addr"]")
L →  id                L.array = null; L.addr = symtop.get(id.lexeme)
L →  id[Elist          name = id .lexeme
                       L.array = symtop.get(id.lexeme); L.type = getBaseType(id.lexeme)
                       L.addr = new Temp(); gen(L.addr "=" E.addr "*" L.type.width)
Elist → E]             Elist.pos = 1; Elist.addr = new Temp()
                       gen(Elist.addr = E.addr "*" get_width(name, Elist.pos))
Elist → E, Elist1      Elist.pos = Elist1.pos + 1; Elist.addr = new Temp();
                       t = new Temp(); gen(t "=" E.addr "+" Elist1.addr)
                       gen(Elist.addr "=" t "*" get_width(name, Elist.pos))
```

2 Attributes and Auxiliary Functions Used

2.1 Attributes

- **S**: The non-terminal S has no attributes.
- **E**: The non-terminal E has the attribute addr, which denotes the address that holds the value of E.
- **L**: The non-terminal L has the attributes- array, which is a struct that stores the array attributes if L is an array and is null otherwise, type which denotes the base data type of the array elements and is null otherwise and addr which denotes the relative offset from the base address of the array which stores the desired value if L.array is not null and stores the address that holds the value of L if L.array is null. L.type has a sub-attribute that denotes the width of the type.
- **Elist**: The non-terminal Elist has the attributes- pos, which denotes the number of indices we've encountered so far, numbered from the right end. Elist also has the attribute addr which denotes the relative offset (in terms of number of elements) that we've calculated

so far. At the end, it will denote the total relative offset (in terms of number of elements) that we're trying to access of the array.

2.2 Auxiliary Functions

- Temp(): creates a new temporary, required for the 3AC generation. (same as problem 2)
- gen(): generates the code given as its parameter. (same as problem 2)
- getBaseType(): gets the base data type (eg. int, bool) that the array (its name being provided as the argument) is composed of.
- get_width(): gets the width that we need to multiply by for an array, given the index that we are currently looking at. For example, if we are looking at a 3-dimensional array A of size 10x20x12, and we are looking at A[i, j, k]. When we call the function with the second argument as 1, it will return 200 (as in column-major representation, we need to multiply 10x20 times the index we have provided, in this case k). Similarly, when we pass the second argument as 2, it will return 10. At the end, in the production of L, this will get multiplied by the element size.

3 Annotated Parse Tree for x = c + A[i, j]

Note: the square brackets and comma are not drawn as separate nodes in the tree so that the diagram is not cluttered.

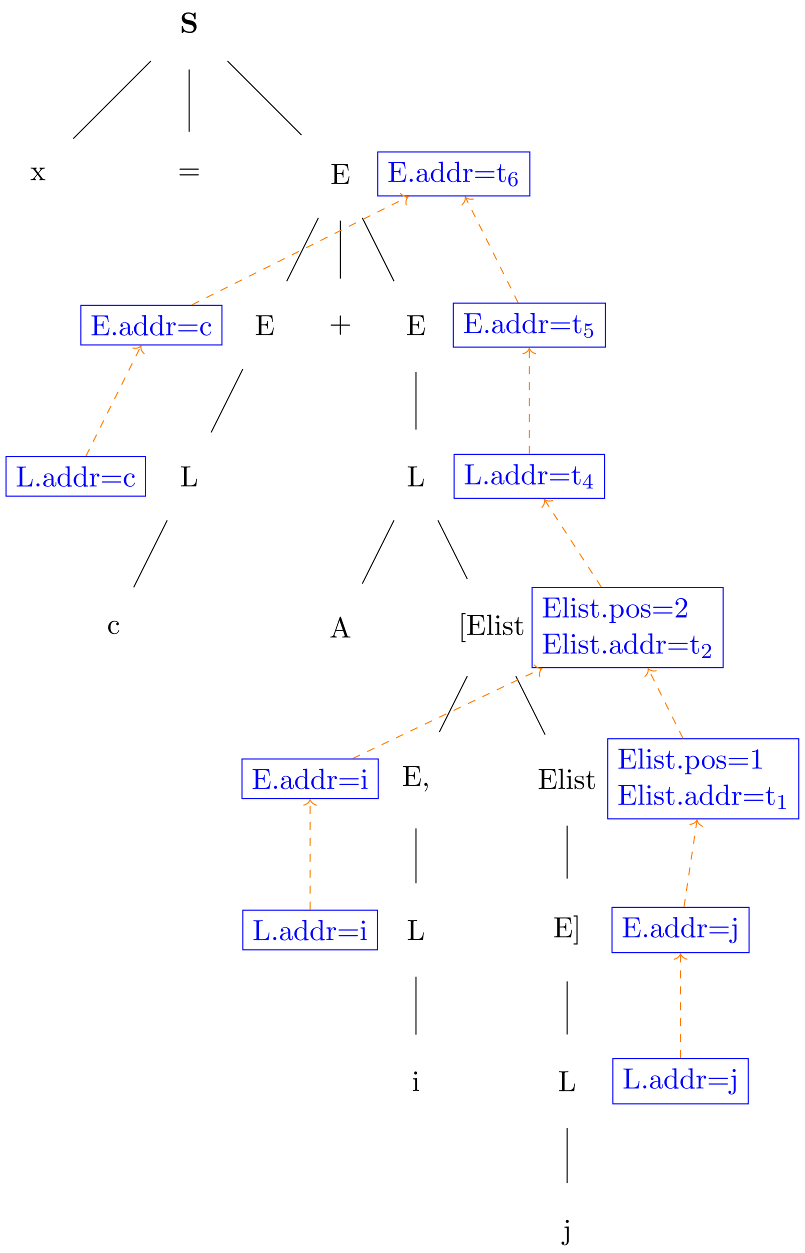

Figure 3: Annotated parse tree for $x = c + A[i][j]$

4 Generated 3AC code

```
Label    Code         Semantic Action
   1:    t1 = j * 10  Elist.addr "=" E.addr "*" get_width(A, 1)
   2:    t3 = i + t1  t "=" E.addr "*" Elist1.addr
   3:    t2 = t3 * 1  Elist.addr "=" t "*" get_width(A, 2)
   4:    t4 = t2 * 4  L.addr "=" Elist.addr "*" L.type.width
   5:    t5 = A[t4]   E.addr "=" L.array.base"["L.addr"]"
   6:    t6 = c + t5  E.addr "=" E1.addr "+" E2.addr
   7:    x = t6       symtop.get(id.lexeme) "=" E.addr
```