Compilers Assign3

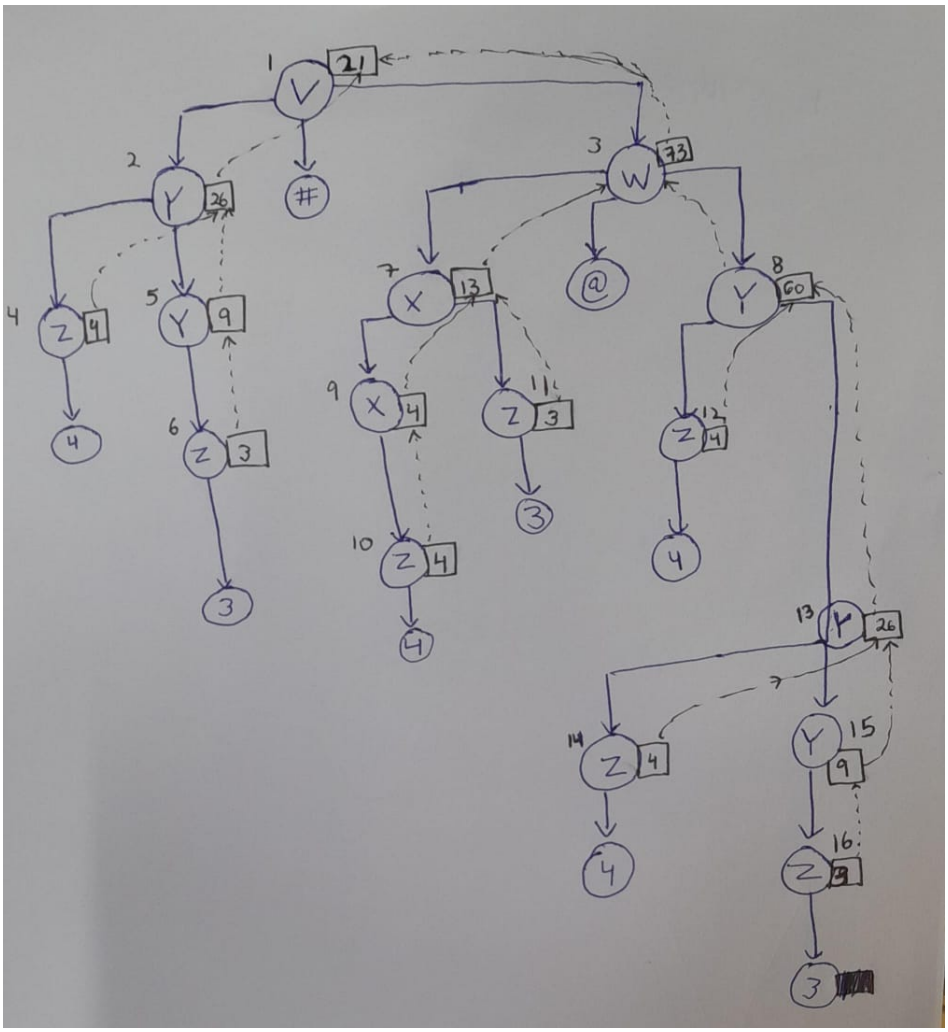
Geetika (210392)

April 5, 2024

1 Question

1.1 Part (Annotated parse tree)

- The values in the square black box represent the values of the respective non-terminal
- The number beside the nodes is used for indexing the translation scheme in part2
- The edges showing flow of attribute values are dotted and black.



1.2 Translation Scheme

(1) $V.val = 73\%26 = 21$

(2) $Y.val = 2 * (4 + 9) = 26$

(3) $W.val = 13 + 60 = 73$

(4) $Z.val = 4$

(5) $Y.val = 3 * 3 = 9$

(6) $Z.val = 3$

(7) $X.val = 4 + 3 * 3 = 13$

(8) $Y.val = 2 * (4 + 26) = 60$

(9) $X.val = 4$

(10) $Z.val = 4$

(11) $Z.val = 3$

(12) $Z.val = 4$

(13) $Y.val = 2 * (4 + 9) = 26$

(14) $Z.val = 4$

(15) $Y.val = 3 * 3 = 9$

(16) $Z.val = 3$

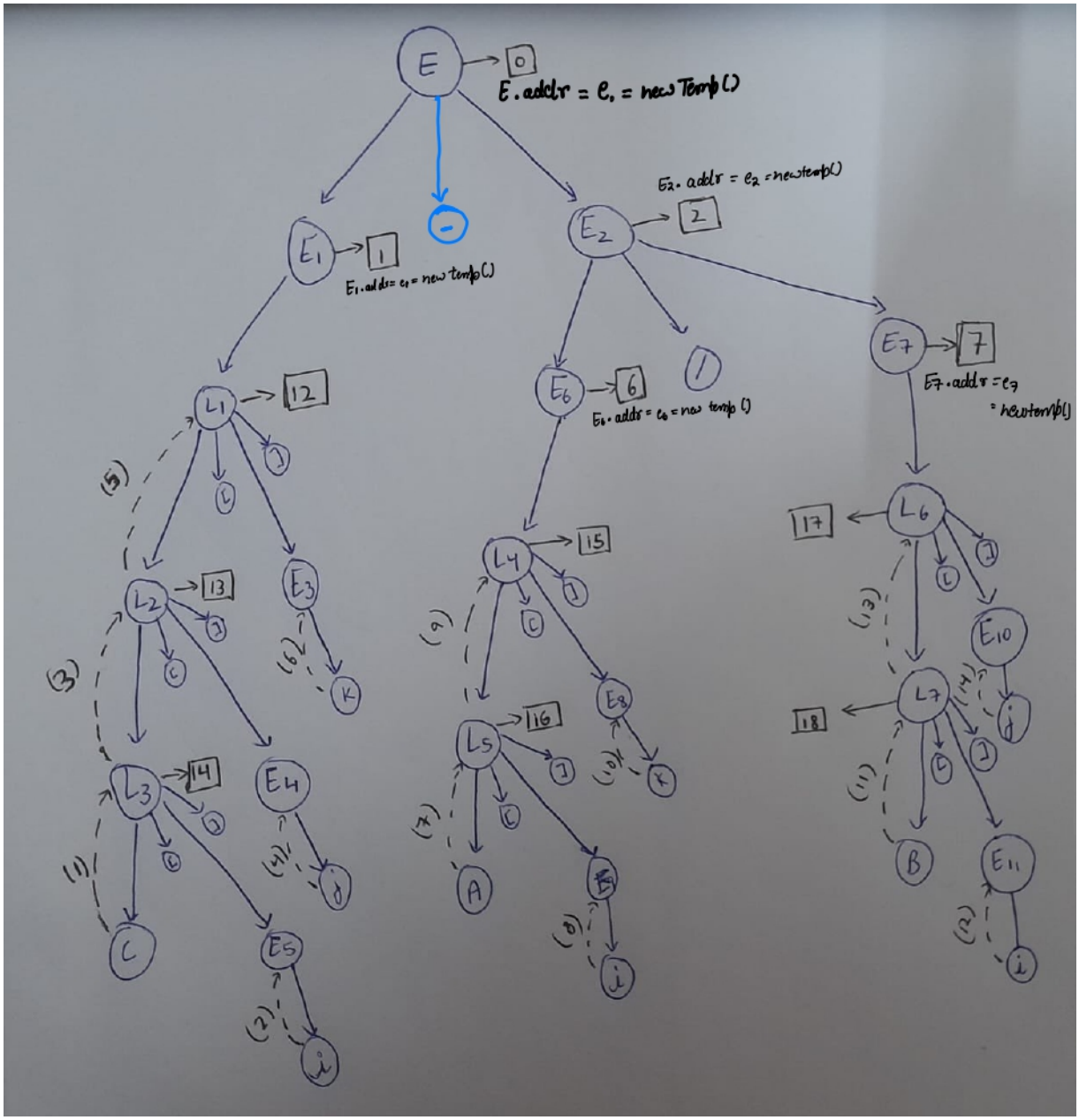
Hence the value of V compted using above translation scheme is **21**.

1.3

The grammar is S- S-attributed since the value of the node depends only on the child nodes and is not dependent on the parent node.

2 Question

- All the nonterminals are represented by circular nodes.
- Since the diagram could not accommodate the code per node it has been indexed by the number in the Rectangular box (black).
- When a nonterminal goes to id, it has been replaced by its lexeme also in the production L− >id[E] id is represented by its respective lexeme.
- Also, the semantic actions other than code generation are indexed by (<NUM>).
- In 3AC, the temporaries corresponding to $L_i.addr$ are denotes by l_i and the temporaries corresponding to $E_i.addr$ are denoted by e_i. In cases where E_i.addr→ id it has been replaced by id's lexeme.



2.1 Semantic Actions other than the code generation.

(1) L3.array = symtop.get(C)
L3.type = int[10][6]
L3.addr=l_3=new temp()

(2) E5.addr = symtop.get(i)

(3) L2.array = C
L2.type = int[6]
L2.addr=l_2=new temp()
t_2=new temp()

(4) E4.addr = symtop.get(j)

(5) L1.array = C
L2.type = int
L1.addr=l_1=new temp()
t_1=new temp()

(6) E3.addr = symtop.get(k)

(7) L5.array = symtop.get(A)
L5.type = int[8]
L5.addr=l_5=new temp()

(8) E9.addr = symtop.get(i)

(9) L4.array = A
L5.type = int
L4.addr=l_4=new temp()
t_4=new temp()

(10) E8.addr = symtop.get(k)

(11) L7.array = symtop.get(B)
L7.type = int[6]
L7.addr=l_7=new temp()

(12) E11.addr = symtop.get(i)

(13) L6.array = B
L6.type = int
L6.addr=l_6=new temp()
t_6=new temp()

(14) E10.addr = symtop.get(j)

2.2 3AC

3AC corresponding to nodes

[0]($E -> E_1 - E_2$)
$e_0 = e_1 - e2$

[1]($E_1 -> L_1$)
$e_1 = C[l_1]$

[2]($E_2 -> E_6/E_7$)
$e_2 = e_6/e7$

[6]($E_6 -> L_4$)
$e_6 = A[l_4]$

[7]($E_7 -> L_6$)
$e_7 = B[l_6]$

[12]($L_1 -> L_2[E_3]$)
$t_1 = k * 4$
$l_1 = l_2 + t_1$

[13]($L_2 -> L_3[E_4]$)
$t_2 = j * 24$
$l_2 = l_3 + t_2$

[14]($L_3 -> C[E_5]$)
$l_3 = i * 240$

[15]($L_4 -> L_5[E_8]$)
$t_4 = k * 4$
$l_4 = l_5 + t_4$

[16](L5 $\rightarrow$ A[E9])
$l_5 = i * 32$

[17]($L_6 -> L_7[E_{10}]$)
$t_6 = j * 4$
$l_6 = l_7 + t_6$

[18]($L_7 -> B[E_{11}]$)
$l_7 = i * 24$

Ordered 3AC

$l_3 = i * 240$
$t_2 = j * 24$
$l_2 = l_3 + t_2$
$t_1 = k * 4$
$l_1 = l_2 + t_1$
$e_1 = C[l_1]$
$l_5 = i * 32$
$t_4 = k * 4$
$l_4 = l_5 + t_4$
$e_6 = A[l_4]$
$l_7 = i * 24$
$t_6 = j * 4$
$l_6 = l_7 + t_6$
$e_7 = B[l_6]$
$e_2 = e_6/e7$
$e_0 = e_1 - e2$

3 Question

3.1 Semantic Actions

$S \rightarrow \textbf{id} = E$	`{gen(symtop.get(id.lexeme) "=" E.addr)}`
$S \rightarrow L = E$	`{gen(L.addr "=" E.addr)}`
$E \rightarrow E_1 + E_2$	`{ E.addr = new Temp();` `gen(E.addr "=" E_1.addr "+" E_2.addr)}`
$E \rightarrow L$	`{E.addr = L.addr}`
$L \rightarrow id$	`{L.addr = get(symtop.get(id.lexeme)}`

$L \rightarrow id[Elist$	`{L.array=symptop.get(id.lexeme)`
	`L.type=L.array.type.element`
	`L.addr=new temp()`
	`t=new temp()`
	`gen(L.addr"="0)`
	`for(auto Eaddr : Elist.indexes) {`
	`gen(t"="L.type.width*Eaddr);`
	`gen(Laddr"="L.addr"+"t);`
	`L.type=L.type.element;}`
	`gen(L.addr"="L.array.base"["L.addr"]");`

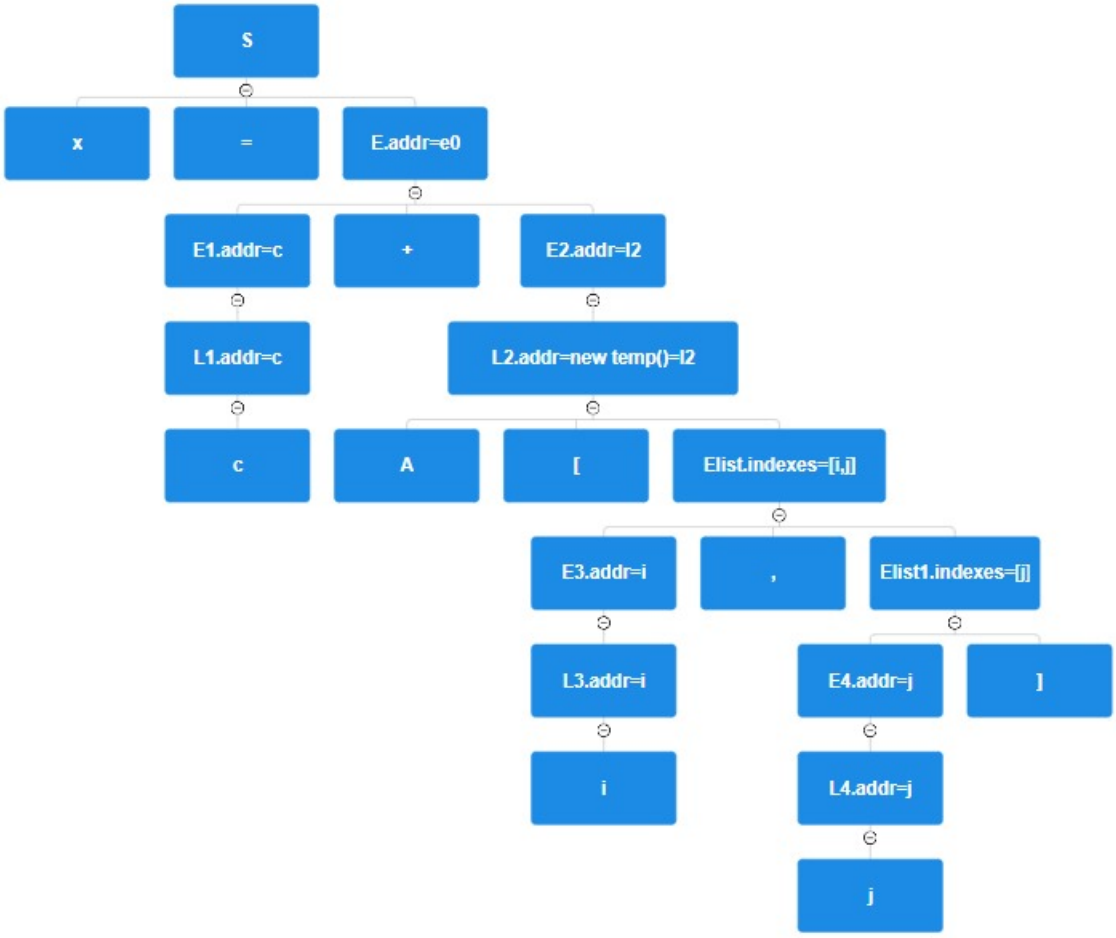
Elist $\rightarrow E]$	`{Elist.indexes.push(E.addr)}`
Elist $\rightarrow E, Elist_1$	`{Elist.indexes.push(E.addr)` `Elist.copy(Elist1.indexes)}`

3.2 Attributes and Auxiliary functions

Assumption : Type constructor for an array like int [3][4][5] is stored in the symbol table as array(5,array(4,array(3,int))). Hence, type.element for this will be array(4,array(3,int)) and type.element.width is 4*3*4 = 48. And the structure of array is same as the prev ques.

- Attributes
  - **addr** As covered in class, it can refer temporaries, constant or identifier.
  - **array** Is same as previous question and has attributes like
    array.type (stores the type of the array),
    array.base (gives the base address of the array).
  - **type** gives the type corresponding the nonterminal, and has the relevant fields corresponding the nonterminal. In case of array type.element gives type of element in the array and type.width gives the size of the datatype.
  - **indexes** is a vector of E.addr which has operations like push(E.addr e_i) (which pushes the e_i in the current vector of indexes)
- Auxiliary functions
  - **gen(***strings***)**: generates 3AC (As done in class)
  - **symtop.get(***id.lexeme***)**: Returns the symbol table entry corresponding to the identifier lexeme. (As done in class)
  - **Elist.copy(***Vector* $<$ *E.addr* $>$**)**: It takes the vector of E.addr and pushes the element in order in currently existing Elist.indexes vector
  - **new temp()**: returns a new temporary

3.3 Annotated Parse Tree



3.4 3AC

In 3AC,
l_2 corresponds to temporay of L2.addr and later its assigned to E2.addr.
t is the temporary generated in $L_2 -> [Elist$
e_0 is the temporary corresponding to E_0

3AC corresponding to nodes

- $L_2 -> [Elist$
  $l_2 = 0$
  $t = i * 4$
  $l_2 = l_2 + t$
  $t = j * 40$
  $l_2 = l_2 + t$
  $l_2 = A[l2]$
- $E -> E_1 + E_2$
  $e_0 = c + l_2$
- $S \rightarrow x = E$
  x=e_0

Ordered 3AC
$l_2 = 0$
$t = i * 4$
$l_2 = l_2 + t$
$t = j * 40$
$l_2 = l_2 + t$
$l_2 = A[l2]$
$e_0 = c + l_2$
$x = e_0$