Name: Chitwan Goel
Roll No. 210295

CS335 Assignment 3

1 Problem 1

Given SDD:

$V \to Y\#W$	$\{V.val = W.val \,\%\, Y.val\}$
$W \to X@Y$	$\{W.val = X.val + Y.val\}$
$X \to X_1 Z$	$\{X.val = X_1.val + 3 \times Z.val\}$
$X \to Z$	$\{X.val = Z.val\}$
$Y \to ZY_1$	$\{Y.val = 2 \times (Z.val + Y_1.val)\}$
$Y \to Z$	$\{Y.val = 3 \times Z.val\}$
$Z \to 3$	$\{Z.val = 3\}$
$Z \to 4$	$\{Z.val = 4\}$

1.1 Show the annotated parse tree for the input string 43#43@443.

Figure 1 shows the annotated parse tree for the given SDD and given input string.

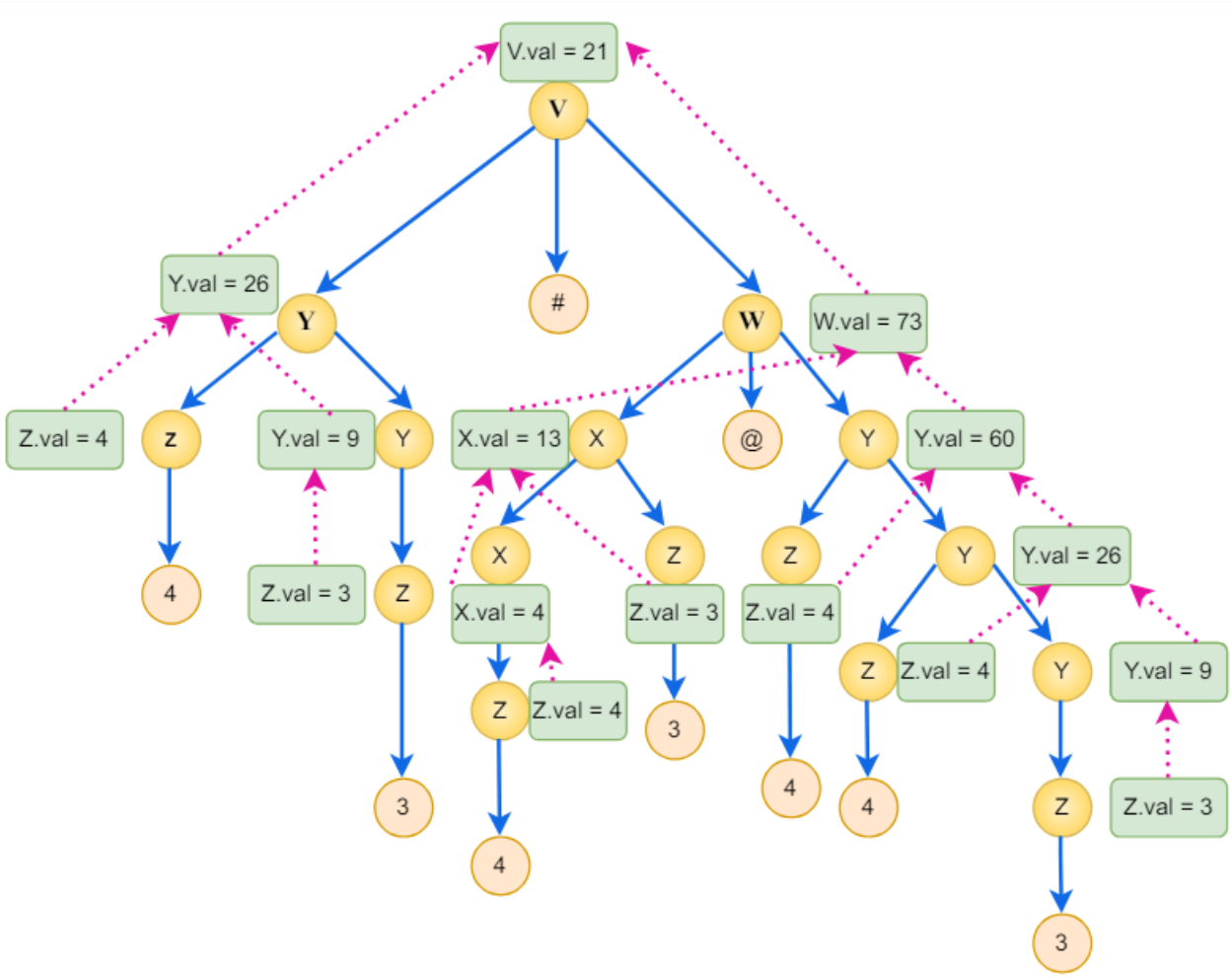

Figure 1: Annotated Parse Tree

1.2 What is the value at V computed by the translation scheme for the above input string.

As clear from the annotated parse tree shown in *Figure 1*, the value at V is 21.

1.3 Explain whether the grammar is S-attributed or L-attributed.

The attributes of head of every production in the grammar is computed using the attributes of symbols in the body of production. So the grammar is $S - attributed$.

2 Problem 2

Consider the following extended grammar with semantic translation.

$S \rightarrow \textbf{id} = E$	$\{gen(symtop.get(\textbf{id}.lexeme) \text{ “} = \text{ ”} E.addr)\}$
$S \rightarrow L = E$	$\{gen(L.array.base \text{ “[”} L.addr \text{ “]” “} = \text{ ”} E.addr)\}$
$E \rightarrow E_1 - E_2$	$\{E.addr = \textbf{new } Temp(); gen(E.addr \text{ “} = \text{ ”} E_1.addr \text{ “} - \text{ ”} E_2.addr)\}$
$E \rightarrow E_1/E_2$	$\{E.addr = \textbf{new } Temp(); gen(E.addr \text{ “} = \text{ ”} E_1.addr \text{ “/”} E_2.addr)\}$
$E \rightarrow \textbf{id}$	$\{E.addr = symtop.get(\textbf{id}.lexeme)\}$
$E \rightarrow L$	$\{E.addr = \textbf{new } Temp(); gen(E.addr \text{ “} = \text{ ”} L.array.base \text{ “[”} L.addr \text{ “]”})\}$
$E \rightarrow *E_1$	$\{E.addr = \textbf{new } Temp(); gen(E.addr \text{ “} = \text{ ” “} * \text{ ”} E_1.addr)\}$
$L \rightarrow \textbf{id}[E]$	$\{L.array = symtop.get(\textbf{id}.lexeme); L.type = L.array.type.elem;$ $L.addr = \textbf{new } Temp(); gen(L.addr \text{ “} = \text{ ”} E.addr \text{ “} * \text{ ”} L.type.width)\}$
$L \rightarrow L_1[E]$	$\{L.array = L_1.array; L.type = L_1.type.elem;$ $t = \textbf{new } Temp(); L.addr = \textbf{new } Temp();$ $gen(t \text{ “} = \text{ ”} E.addr \text{ “} * \text{ ”} L.type.width); gen(L.addr \text{ “} = \text{ ”} L_1.addr \text{ “} + \text{ ”} t); \}$

3AC for the given expression $C[i][j][k] - A[i][k]/B[i][j]$ is written below:

```
t1 = i * 240
t2 = j * 24
t3 = t1 + t2
t4 = k * 4
t5 = t3 + t4
t6 = C[t5]
t7 = i * 32
t8 = k * 4
t9 = t7 + t8
t10 = A[t9]
t11 = i * 24
t12 = j * 4
t13 = t11 + t12
t14 = B[t13]
t15 = t10 / t14
t16 = t6 - t15
```

Figure 2 shows the annotated parse tree for the given grammar and input expression.

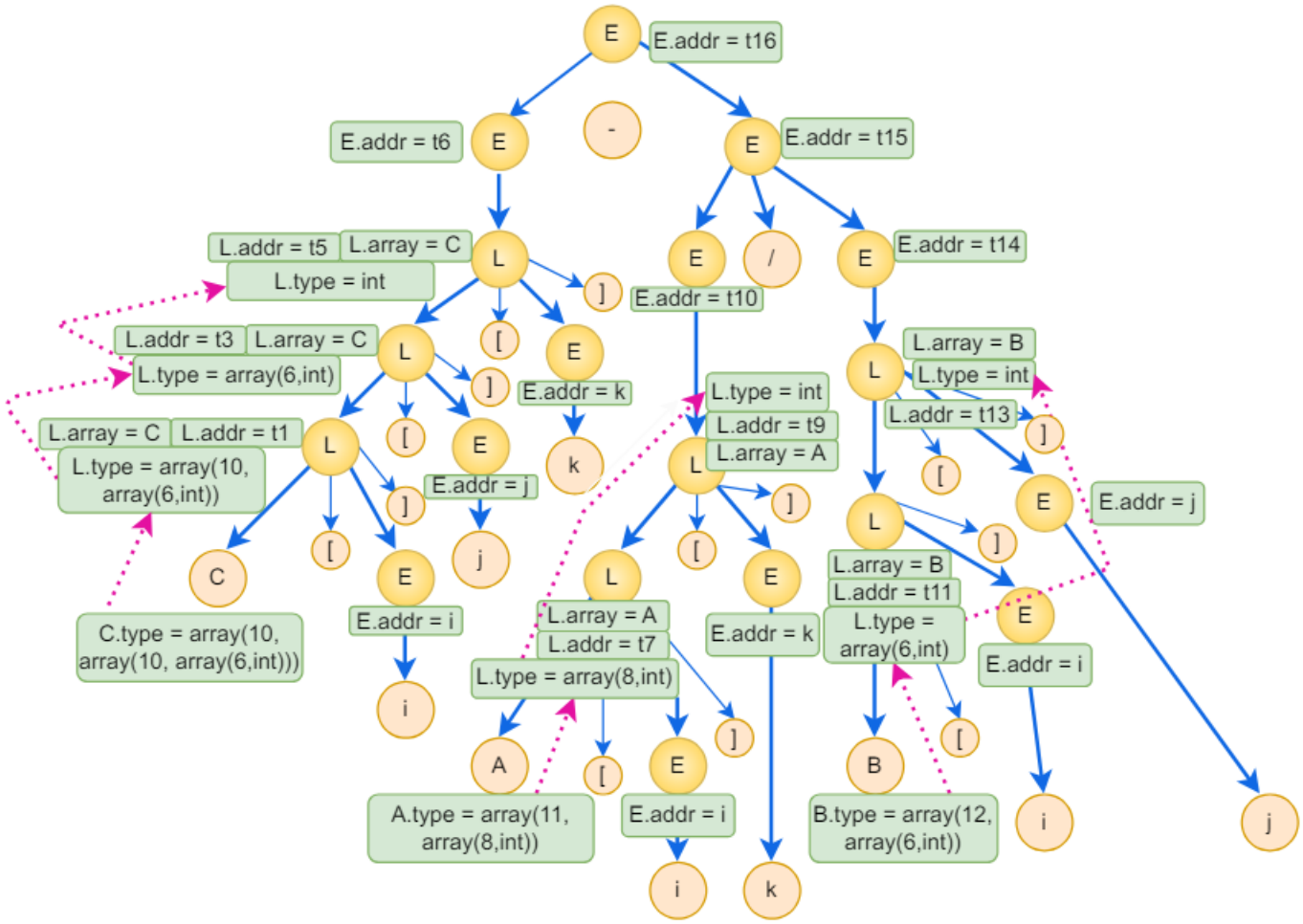

Figure 2: Annotated Parse Tree
For identifers, the node is labelled as **id**.*lexeme* itself like A, i, etc..

3 Problem 3

3.1 Construct an SDT translation scheme for array expressions using column-major organization of arrays. Show the semantic actions for your proposed translation.

Below is the extended grammar with semantic actions:

$S \rightarrow \textbf{id} = E$	$\{gen(symtop.get(\textbf{id}.lexeme) \text{ “} = \text{”} E.addr)\}$

$S \rightarrow L = E$	{
	 if$(L.dims > 0)$ **then**
	 $gen(L.array.base \text{ “[”} L.addr \text{ “]”} \text{ “} = \text{”} E.addr)$;
	 else $gen(L.addr \text{ “} = \text{”} E.addr)$
	 }

$E \rightarrow E_1 + E_2$	$\{E.addr = \textbf{new}\ Temp(); gen(E.addr \text{ “} = \text{”} E_1.addr \text{ “} + \text{”} E_2.addr)\}$

$E \rightarrow L$	{
	 if$(L.dims > 0)$ **then**
	 $E.addr = \textbf{new}\ Temp(); gen(E.addr \text{ “} = \text{”} L.array.base \text{ “[”} L.addr \text{ “]”})$;
	 else $gen(E.addr \text{ “} = \text{”} L.addr)$
	 }

$L \rightarrow \textbf{id}$	$\{L.addr = symtop.get(\textbf{id}.lexeme); L.dims = 0\}$

$L \rightarrow \textbf{id}[Elist$	{
	 $L.array = symtop.get(\textbf{id}.lexeme); L.dims = sizeof(Elist.addrlist)$;
	 $L.addr = \textbf{new}\ Temp(); tmp = \textbf{new}\ Temp(); gen(L.addr \text{ “} = \text{”} 0)$;
	 $cur\_type = L.array.type.elem$; $prev\_type = L.array.type$; $width = 4$
	 $for\ (adr\ in\ Elist.addrlist)$ **do**
	 $gen(tmp \text{ “} = \text{”} adr \text{ “} * \text{”} width)$
	 $gen(L.addr \text{ “} = \text{”} L.addr \text{ “} + \text{”} tmp)$
	 $width = width * (prev\_type.width/curr\_type.width)$;
	 $prev\_type = curr\_type$; $curr\_type = curr\_type.elem$;
	 done
	 }

$Elist \rightarrow E]$	$\{Elist.addrlist = newlist(E.addr)\}$
$Elist \rightarrow E, Elist_1$	$\{tmp = newlist(E.addr); Elist.addrlist = concatlist(tmp, Elist_1.addrlist);\}$

3.2 Explain the attributes and auxiliary functions in your SDT.

Attributes: Below is the specification of the attributes used.

1. *E.addr :=* Holds the value of expression E (can be a name, a constant, or a temporary).
2. *Elist.addrlist :=* Is a list of addresses used for indexing the array
3. *L.addr :=* Is used for computing the offset for array reference

4. *L.array :=* Points to the symbol table entry for the array name. *L.array.base* gives the base address of the array.
5. *L.dims :=* Stores the dimension of *L.array*
6. *type :=* For an array a, *a.type* stores its type. For an array of type t, *t.width* gives the width of t and *t.elem* gives the type of element.
7. *symtop :=* Points to the current symbol table

Functions: Below is the specification of the functions used.

1. *gen :=* creates a 3AC instruction and appends it to an instruction stream
2. *sizeof :=* Returns the number of elements in a list
3. *newlist(x) :=* Returns a list containing only 1 element x
4. *concatlist(a,b) :=* Returns a merged list with elements of a followed by those of b
5. *new Temp() :=* Gives a new temporary

3.3 Show the annotated parse tree for the expression $x = c + A[i, j]$. Assume that A is a 10×20 array of integers, the size of an integer is 4 bytes, and the arrays are zero-indexed.

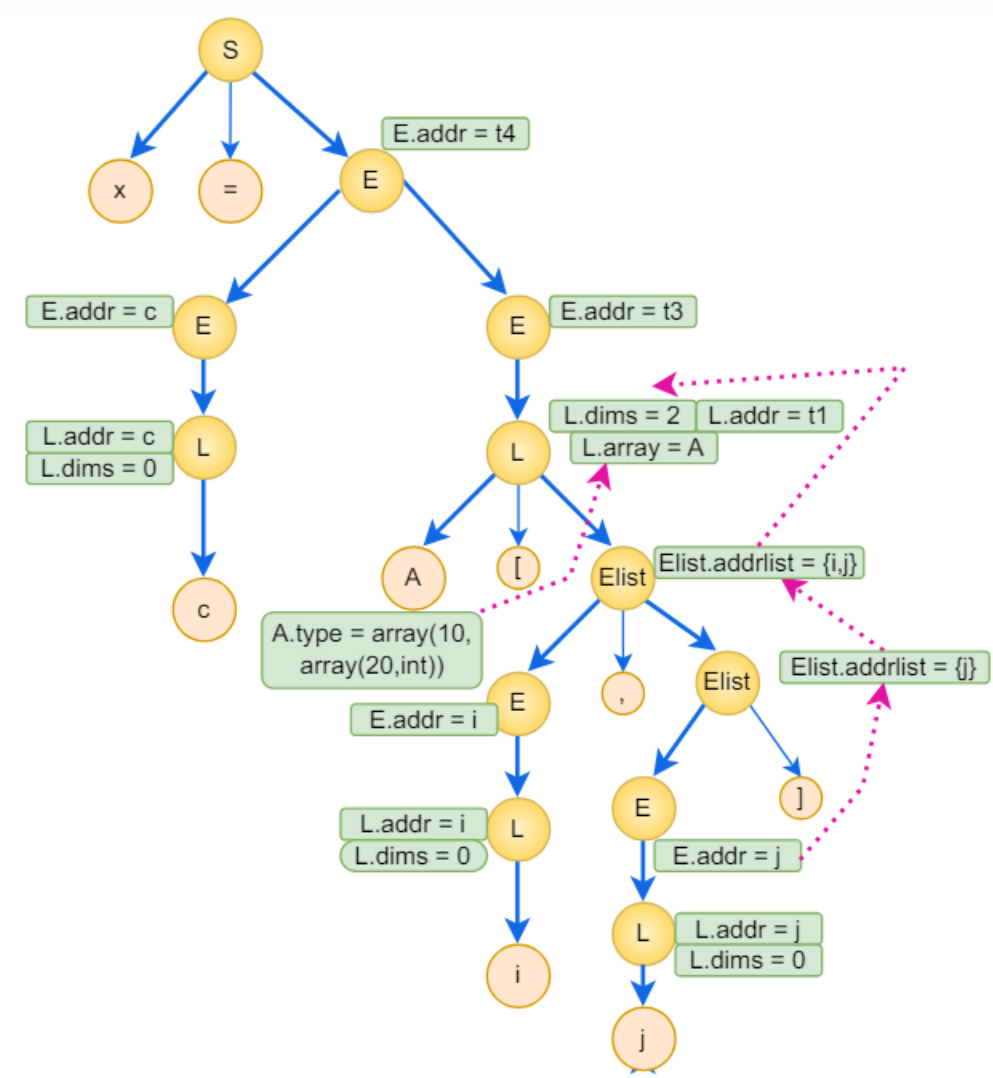

Figure 3: Annotated parse tree
For identifers, the node is labelled as **id**.*lexeme* itself like A, i, etc..

3.4 Show the generated 3AC for the above expression.

Below is the generated 3AC.

```
t1 = 0
t2 = i * 4
t1 = t1 + t2
t2 = j * 40
t1 = t1 + t2
t3 = A[t1]
t4 = c + t3
x = t4
```