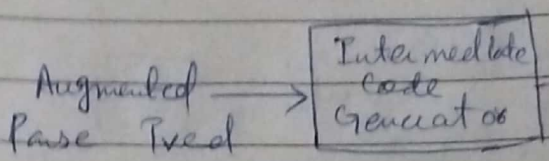
Intermediate Code Generation

Date 9-2



- Last phase of the frontend of a compiler.
- It creates an intermediate representation then the backend will generate targeted machine code.
- The choice of intermediate representation depends on compiler to compiler.
- An intermediate representation can be an actual representation or it may consist of internal data structures.
- C programming language is widely used as an intermediate form because of its flexibility with machine / low-level code.

Variants of Syntax Trees.

- Nodes in a syntax tree represents constructs in the source program, its children represents meaningful components of the construct.
- A directed acyclic graph (DAG) for an expression identifies the common subexpressions of the expression

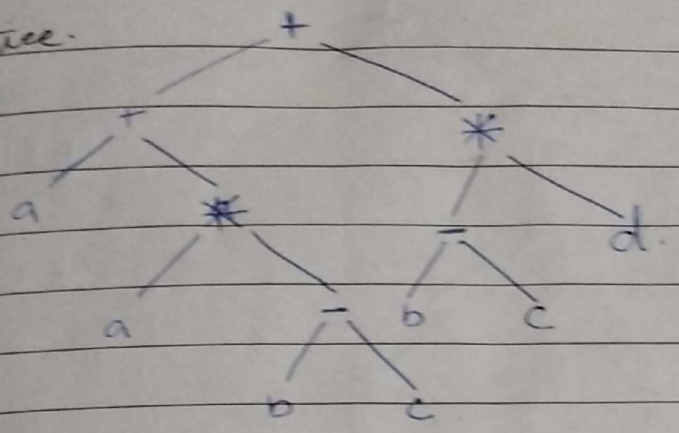
Directed Acyclic Graphs for Expressions

- DAGs can be constructed using the same technique as for Syntax trees.
- A DAG has leaves corresponding to atomic operands & interior nodes for operators.
- Here a node can have more than one parent, if a common subexpression exists

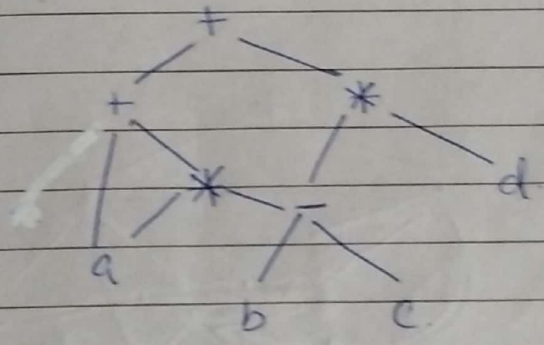
Date ____ 20__

Ex: 6.1 a + a * (b - c) + (b - c) * d.

Syntax tree.



DAG.



Following are semantic rules to construct syntax trees or DAGs.

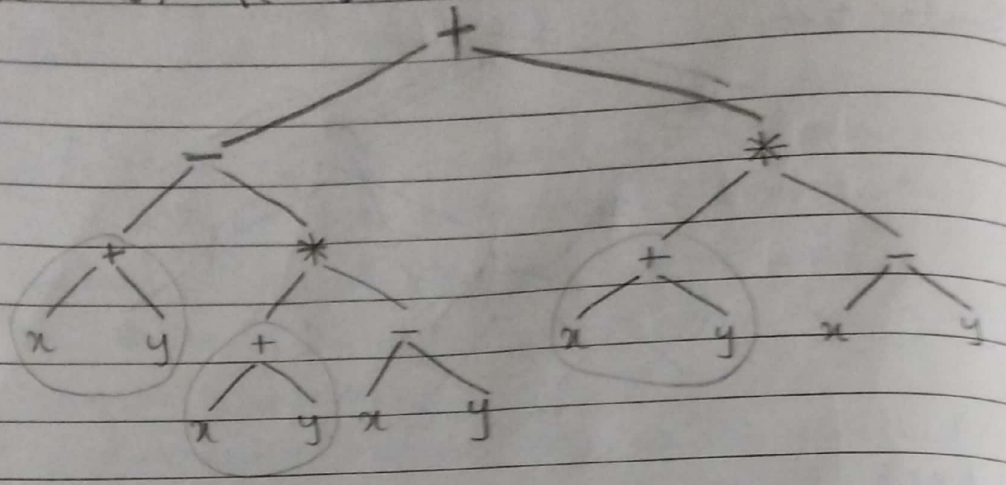
Productions	Semantic rules.
1) $E \rightarrow E_1 + T$	E.node = new Node('+', E1.node, T.node)
2) $E \rightarrow E_1 - T$	E.node = new Node('-', E1.node, T.node)
3) $E \rightarrow T$	E.node = T.node
4) $T \rightarrow (E)$	T.node = E.node.
5) $T \rightarrow id$	T.node = new Leaf(id, id.entry)
6) $T \rightarrow num$	T.node = new Leaf(num, num.val)

⇒ Node (op, left, right) → It'll first check whether there exist a node with value op & left & right child, if true, it'll returns that node, else creates a new Node.

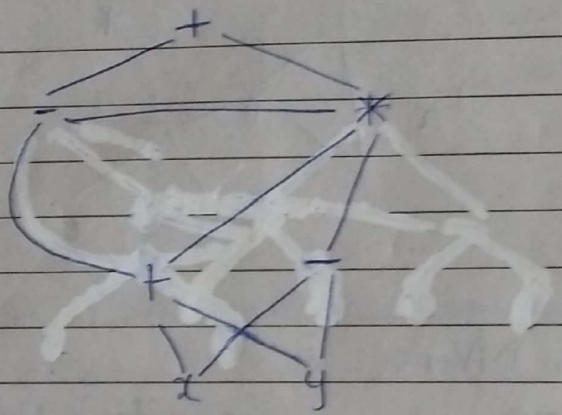
Date ____ 20

Exercise 6.1.1: Construct the DAG for the expression

((x + y) - ((x + y) * (x - y))) + ((x + y) * (x - y))

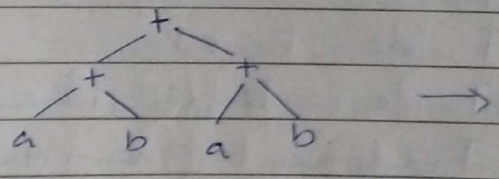


DAG



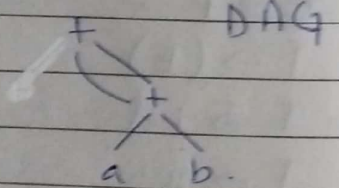
Exercise: 6.1.2: Construct a DAG ~~or identify the value numbers for the subexpressions~~ of the following expression assuming + associates from the left.

a) a + b + (a + b)

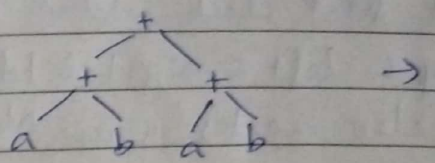


→

DAG

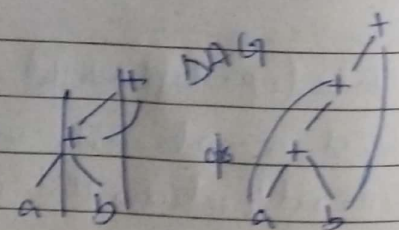


b) a + b + a + b



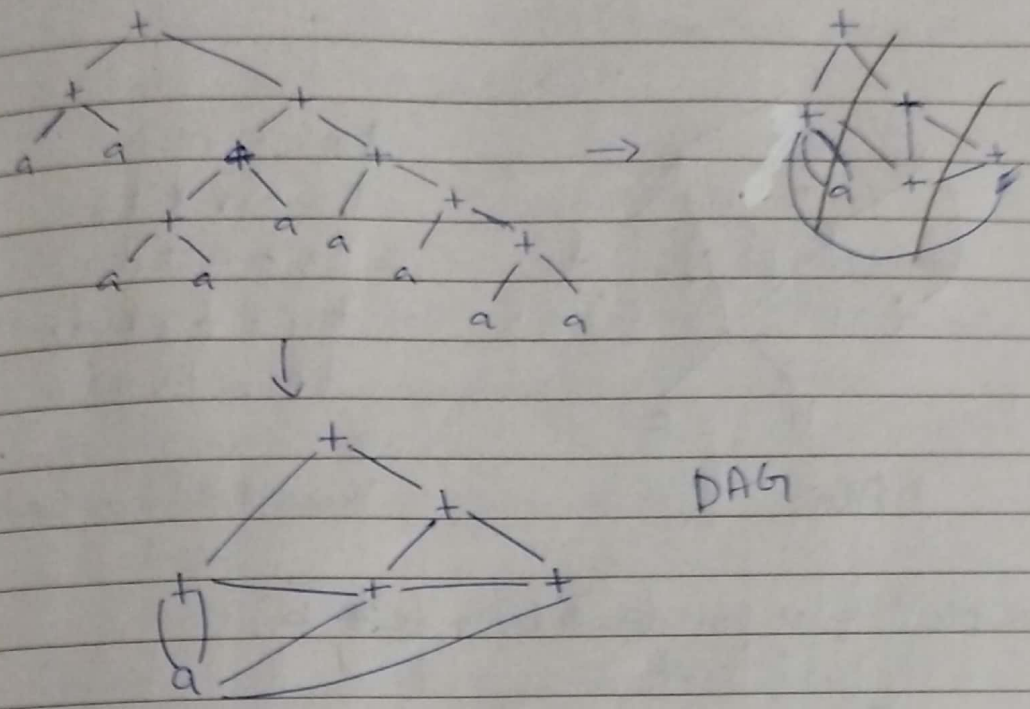
→

DAG



Date 20

a) a+a+((a+a+a+(a+a+a+a)))

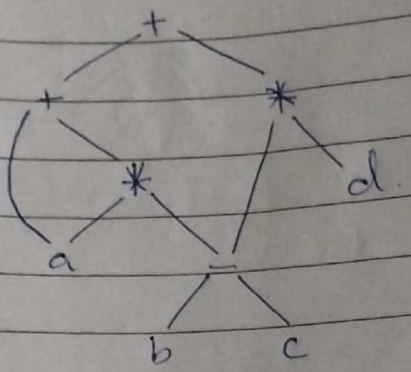


THREE ADDRESS CODE

- At most one operator on the right side of an instruction, no built up arithmetic expressions are permitted.
  ⇒ x + y * z might be translated into
  ⇒ $t_1 = y * z$
  ⇒ $t_2 = x + t_1$
- here t_1 & t_2 are temporary names generated by the compiler.
- This multi-operator arithmetic expressions & nested flow of control statements make three-address code desirable for targeted machine code.
- Three address code is a linearized representation of a syntax tree or a DAG, where the temporary names e.g. t_1 and t_2 represents intermediate nodes.

Date

Example: $a + a * (b-c) + (b-c) * d$



DAG

$t_1 = b - c$

$t_2 = a * t_2$

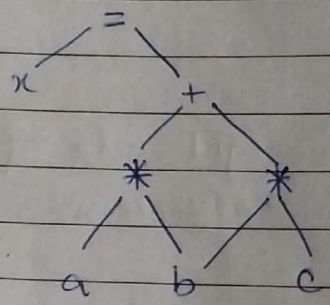
$t_3 = a + t_2$

$t_4 = t_1 * d$

$t_5 = t_3 + t_4$

Three Address Co

Example: $x = (a * b) + (c * b)$



DAG

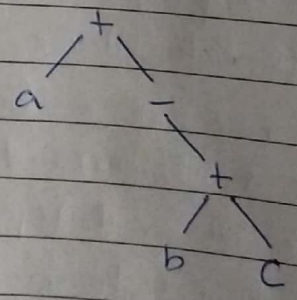
$t_1 = a * b$

$t_2 = c * b$

$x = t_1 + t_2$

Three address code.

Ex: 6.2.1: Translate the arithmetic expression $a + -(b+c)$ into a syntax tree.



Date ____ 20__

Addresses & Instructions.

- Three address code is built from two concepts: addresses and instructions.
- In object-oriented way, it corresponds to the concept of classes & subclasses.
- Three-address code can be implemented using ~~fields~~ & records. with fields for addresses & records called quadruples & tripples.
- An address can b
  1. A name (source program names)
  2. A constant
  3. A compiler-generated temporary. e.g. t_1 & t_2
- Common three-address instruction form
  1. Assignment instructions $x = y \text{ op } z$.
     op: binary, arithmetic or logical operation & x, y, z are addresses
  2. Assignment => $x = \text{op } z$, op is a unary operation such as negation, unary minus, shift etc.
  3. Copy instructions => $x = y$.
  4. Unconditional jump goto L. (L is label)
  5. Conditional jump => if x goto L.
  6. or => if x relop y goto L, relop is relational operators (>, <, == etc).
  7. Procedure calls & return statements.
     param x => for parameters.
     call p, n y = call p, n. (Procedural function calls)
     return y.
  8. Indexed copy instruction => $x = y[i]$, $x[i] = y$
  9. Addresses & pointers => $x = \&y$, $x = *y$.

Date ____ 20

Ex: 6.5. do i = i + 1; while (a[i] < v);

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

Symbolic Labels Representation

Position Numbers Representation.

Quadruples:

- A representation is a data structure of three-address
- A quadruple (or quad) has four fields: op, arg1, arg2, and result.
- A three address instruction in quad would

x = y + z ⇒

op	arg1	arg2	result
+	y	z	x

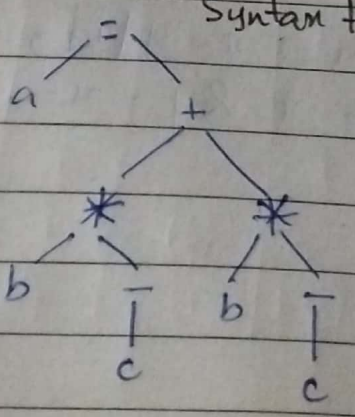
① Instructions with unary operator ⇒ x = minus y or x = y, do not use arg2. A in copy stat x = y, op is =

② Operators such as param don't use arg2 & result.

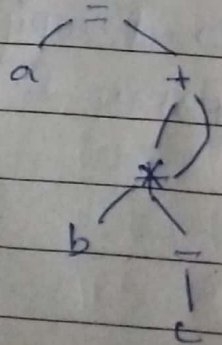
③ Conditional & Unconditional jumps, put label in result.

Ex: 6.6. a = b * -c + b * -c;

Syntax tree



Dag



Date 20

t_1 = minus c
t_2 = b * t_1
t_3 = t_2 + t_2
a = t_3

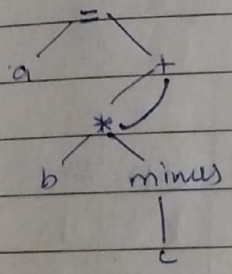
	op	arg_1	arg_2	result
0	minus	c		t_1
1	*	b	t_1	t_2
2	+	t_2	t_2	t_3
3	=	t_3		a

Three-address code Triples. Quadruple.

Triple:

- A tiple has only three fields, op, arg1, and arg2.
- Using tuples, we refer to the result of an operation x op y by its position (indexes) rather than temporary names. ~~positions~~
- Positions will be like (0). Parenthesized numbes represent pointers into the tuple data structure itself

Ex: 6.7 a = b * -c + b * -c.



	op	arg_1	arg_2
0	minus	c	
1	*	b	(0)
2	+	(1)	(1)
3	=	a	(2)

- Triples take less space.
- Quadruples can switch statements.
- In quadruples, we store result in temporary say t, if we move that instruction, we don't have to change t.
- In tiples, result of an instruction is referred by pointer (4) if we move the instruction, the references will have to be changed.
- This problem of tiple is solved by indirect triples.

Date

Indirect tuples.

- Listing of pointers to tuples, rather than listing of tuples themselves
- Let's use an array instruction to list pointers to tuples in a desired order.

The indirect tuple representation of previous example:

instruction	
35	(0)
36	(1)
37	(2)
38	(3)

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

Using indirect tuples, an instruction can be reordered ~~using~~ in instruction list.

Example: $-(a*b)+(c*d+e)$.

$t_1 = a * b$
$t_2 = -t_1$
$t_3 = c * d$
$t_4 = t_3 + e$
$t_5 = t_2 + t_4$

Three address code.

	op	arg1	arg2	result
0	*	a	b	t_1
1	-	t_1		t_2
2	*	c	d	t_3
3	+	t_3	e	t_4
4	+	t_2	t_4	t_5

Quadruple.

	op	arg1	arg2
0	*	a	b
1	-	(0)	
2	*	c	d
3	+	(2)	e
4	+	(1)	(3)

Triple

instruction		op	
101	(0)		(2)
102	(1)		(1)
103	(2)		(0)
104	(3)		(3)
105	(4)		(4)

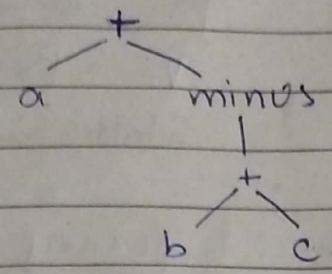
Date ____ 20 ____

Static Single Assignment Form.

Exercise 6.2.1 : a + -(b+c)

Translate this arithmetic expression into.

(a) A syntax tree.



(b) Quadruples.

	op	arg1	arg2	result
0	+	b	c	t_1
1	-	t_1		t_2
2	+	a	t_2	t_3

(c) Triples

	op	arg1	arg2
0	+	b	c
1	-	(0)	
2	+	a	(1)

(d) Indirect Triples.

100	(0)
101	(1)
102	(2)

Static Single Assignment form. (SSA)

- An intermediate representation that provides code optimization
- There are two aspects that distinguishes SSA from three address code.

① All assignments in SSA are ~~two~~ to variables with distinct names (hence static single assignment). Each variable is defined once only but may be used multiple times. (i.e. variables cannot be repeated on L.H.S of assignment).

Date

Exampte:

$P = a+b$	$P_1 = a+b$
$q = p-c$	$q_1 = P_1 - c$
$P = q * d.$	$P_2 = q_1 * d$
$P = e-P$	$P_3 = e - P_2$
$q = P+q$	$q_2 = P_3 + q_2$
Three address code	Static Single assignm

minimum total variables : temporary variables.

$\{ P_1, P_2, P_3, q_1, q_2 \}$.

$+ \{ a, b, c, d, e \}$.

Total 10 variables.

Example: Consider the following three address code.

$x = a/b$	$x_1 = a / b$
$y = c+d$	$y_1 = c + d$
$y = y-x$	$y_2 = y_1 - x_1$
$x = d+y$	$x_2 = d + y_2$
$z = f+y$	$z_1 = f + y_2$
$z = z+a.$	$z_2 = z_1 + a.$

Min Temporary variables used = 6.

Min Total variables use = 6 + 5 = 11

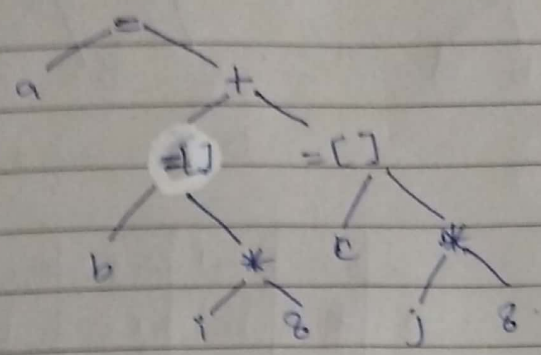
Date ____ 20__

Ex16.2.2

Example: Translate the code segment into
(a) A Syntax tree (b) Quadruples (c) Triples.
(d) Indirect Triples.

(i) $a = b[i] + c[j]$.

(a) Syntax tree.



$t_1 = i * 8$.
$t_2 = b[t_1]$
$t_3 = j * 8$.
$t_4 = c[t_3]$
$t_5 = t_2 + t_4$
$a = t_5$

Three address code.

(b) Quadruple.

	op	arg1	arg2	result
0	*	i	8	t_1
1	=[]	b	t_1	t_2
2	*	j	8	t_3
3	=[]	c	t_3	t_4
4	+	t_2	t_4	t_5
5	=	t_5		a

(c) Triples

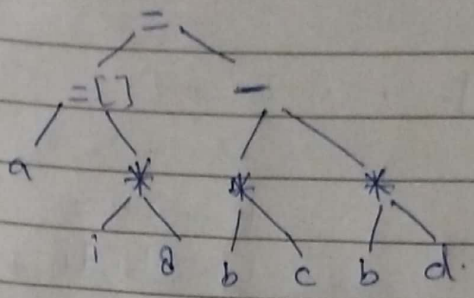
	op	arg1	arg 2
0	*	i	8
1	=[]	b	(0)
2	*	j	8
3	=[]	c	(2)
4	+	(1)	(3)
5	=	a	(4)

(d) Indirect Triples.

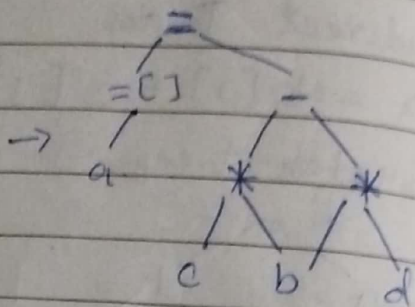
instructions	
100	(0)
101	(1)
102	(2)
103	(3)
104	(4)
105	(5)

Date

(ii) a[i] = b * c - b * d.



Syntax tree



Dag.

$t_1 = b * c$
$t_2 = b * d$.
$t_3 = t_1 - t_2$.
$t_4 = i * 8$.
$t_5 = a[t_4] = t_3$

Three address code

	op	arg1	arg2	result
0	*	b	c	t_1
1	*	b	d	t_2
2	-	t_1	t_2	t_3
3	*	i	8	t_4
4	[]=	a	t_4	t_5
5	=	t_3		t_5

Quadruples.

	op	arg1	arg2
0	*	b	c
1	*	b	d
2	-	(0)	(1)
3.	*	i	8
4.	[]=	a	(3)
5	=	( )	(2)

Triples

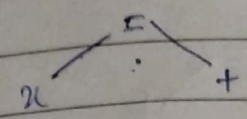
indirect triples

101	(0)
102	(1)
103	(2)
104	(3)
105	(4)
106	(5)

Indirect Triples.

Date ____ 20____

(iii) $x = f(y+1) + 2$.



f(y+1) is a function
(y+) is parameter to f.

$t_1 = y + 1$
$t_2 = f(t)$
$t_3 = t_2 + 2$
$x = t_3$.

~~Interm~~ Three address code.

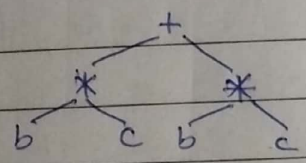
(b) Quadruples

	op	arg1	arg2	result
0	+	y	1	t_1
1	param	t_1		
2	call	f	1	t_2
3	+	t_2	2	t_3
4	=	t_3		x.

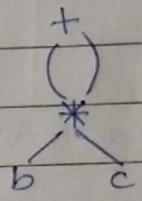
Triples.

	op	arg 1	arg 2
0	+	y	1
1	param	(0)	
2	call	f	1
3	+	(2)	2
4	=	x	(3)

① (b*c) + (b*c).



AST



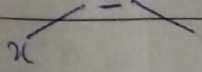
DAG

bc* bc* +

Post fix

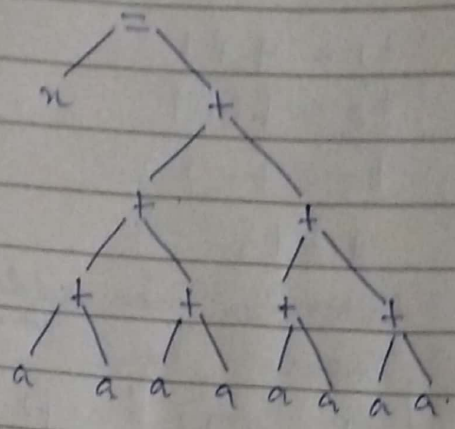
$t_1 = b * c$
$t_2 =$

Q) $x = (((a+a) + (a+a)) + ((a+a) + (a+a)))$

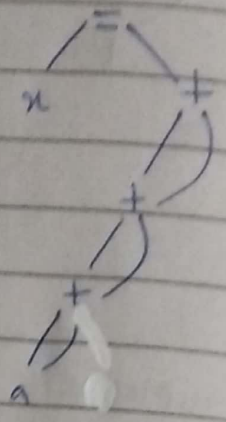


Date

Q) $x = (((a+a)+(a+a)) + (a+a)+(a+a))$



AST

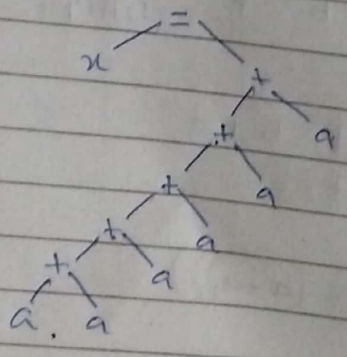


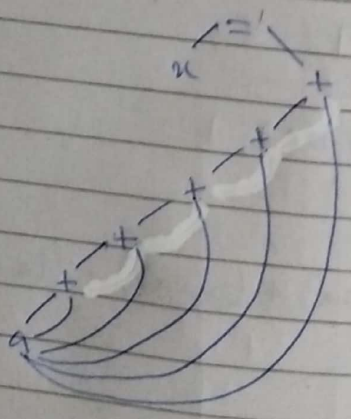
DAG.

$t_1 = a+a$
$t_2 = a+a$
$t_3 = t_1+t_2$
$t_4 = a+a$
$t_5 = a+a$
$t_6 = t_4+t_5$
$t_7 = t_3+t_6$
$x = t_7$

Three adress codes.

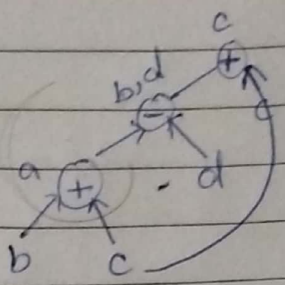
Q) $x = a+a+a+a+a+a$





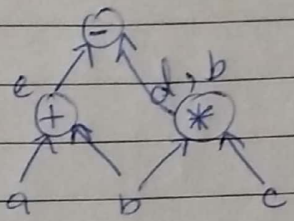
Date ____ 20 ____

Q) $a = b + c$
$b = a - d$
$c = b + c$
$d = a - d$

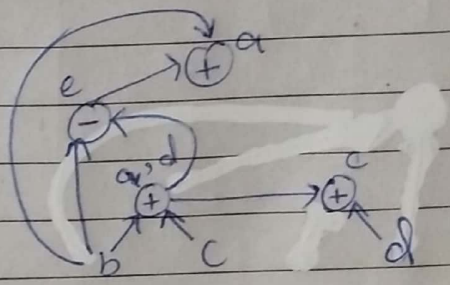


DAG.

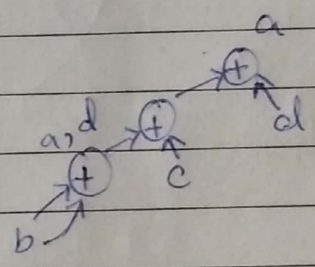
Q) $d = b * c$
$e = a + b$
$b = b * c$
$a = e - d$



Q) $a = b + c$
$c = a + d$
$d = b + c$
$e = d - b$
$a = e + b$



$a = e + b$
$= d - b + b$
$= d$
$= b + c$
$= b + a + d$
$a = b + b + c + d \rightarrow$



Q) $a = b * c$
$d = b$
$e = d * c$
$b = e$
$f = b + c$
$g = f + d$