Translation of Different Statements

Translation of Assignment Statement

- new temp() is a function which returns a new temporary variable.
- Look_up(identifier): checks the identifier value in the symbol table. If it exists then return that.
- The Emit() function is used for appending the three address code to the output file.
- E.val=tells the value of E
- E.place= tells the name that will hold the value of the expression
- E.code= gives the sequence of three address statements evaluating the expression
- E.mode=tells the datatype of E

Translation of Assignment Statement

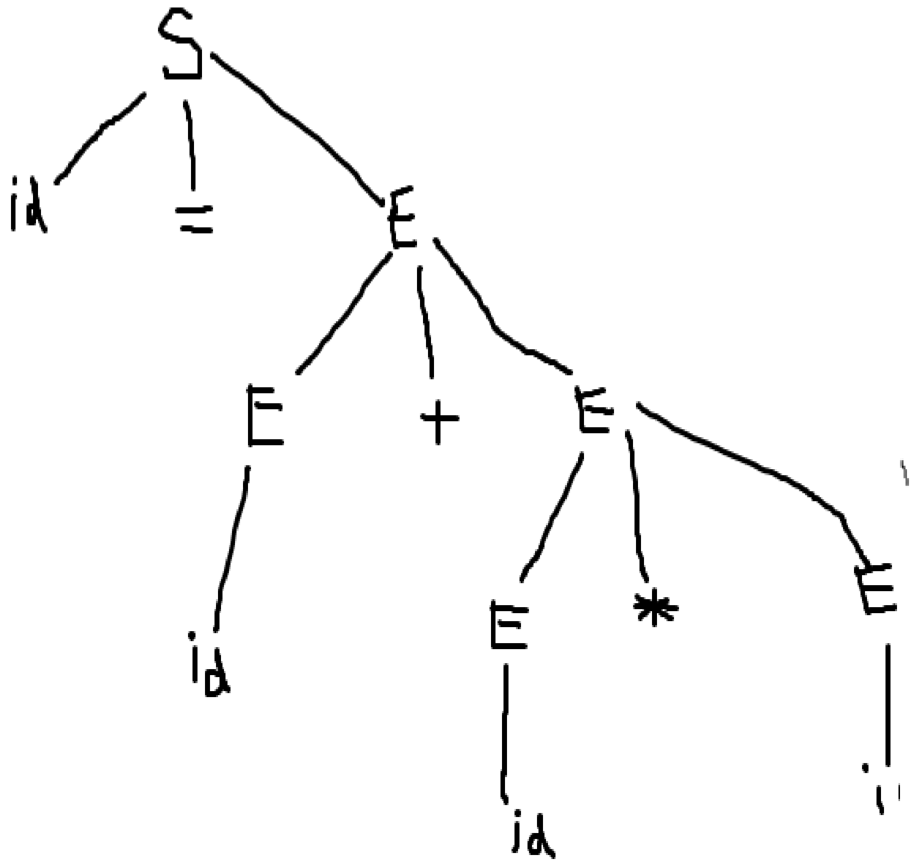
Production rule	Semantic actions

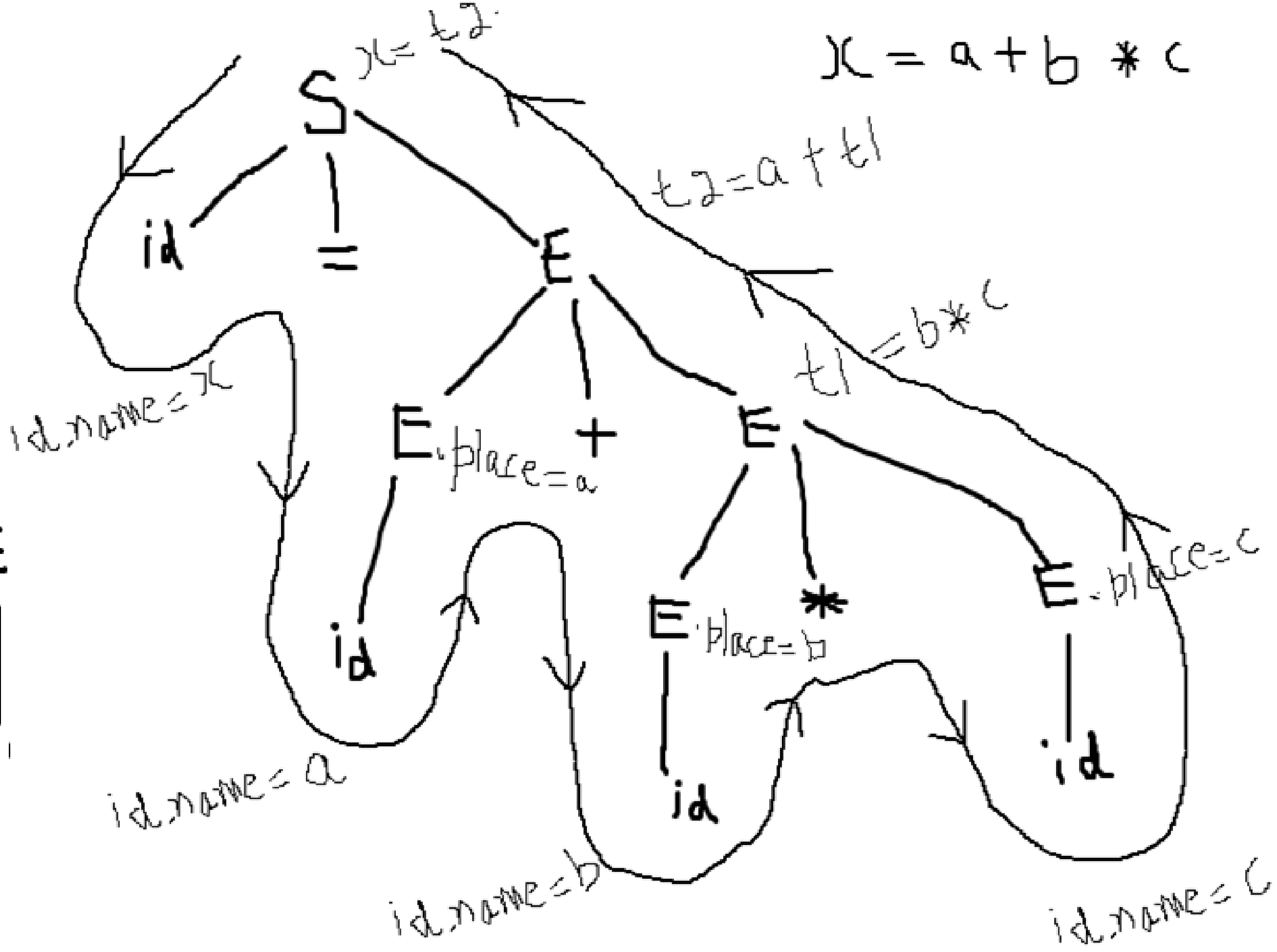
S → id =E	{p = look_up(id.name);
	 If p ≠ nil then
	 Emit (p = E.place)
	 Else
	 Error;}
	p is pointing to name of id in the symbol table and if it exists then it will be updated by the **E.place**

E → E1 + E2	{E.place = newtemp(); Emit (E.place = E1.place '+' E2.place) } i.e. **E.place** will hold the name of that temporary variable whilch will hold the value of E+E
E → E1 * E2	{E.place = newtemp(); Emit (E.place = E1.place '*' E2.place) } i.e. **E.place** will hold the name of that temporary variable whilch will hold the value of E*E
E → (E1)	{E.place = E1.place}

E → id	{p = look_up(id.name);
	 If p ≠ nil then
	 Emit (p = E.place)
	 Else
	 Error;}
	i.e. if **id.name** exists in the symbol table then **E.place** will hold the name which will hold the value of the expression

Translation of Assignment Statement





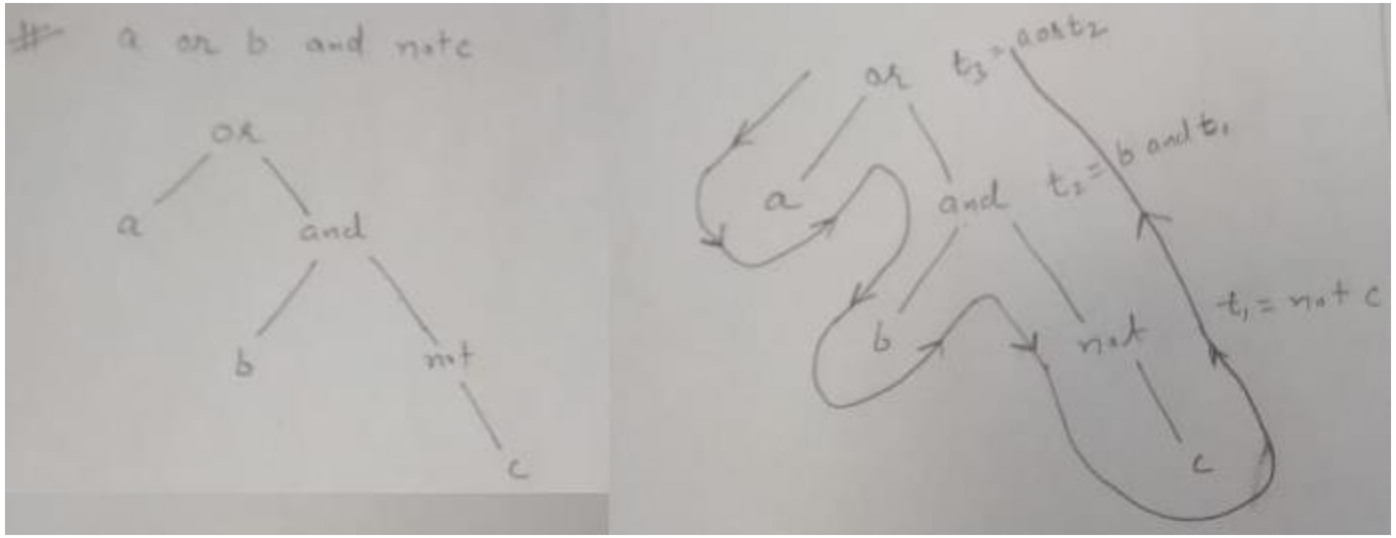
Translation of Boolean Statement

Production rule	Semantic actions
E → E1 OR E2	{E.place = newtemp(); Emit (E.place ':=' E1.place 'OR' E2.place) }
E → E1 AND E2	{E.place = newtemp(); Emit (E.place ':=' E1.place 'AND' E2.place) }
E → NOT E1	{E.place = newtemp(); Emit (E.place ':=' 'NOT' E1.place) }
E → (E1)	{E.place = E1.place}
E → id1 relop id2	{E.place = newtemp(); Emit ('if' id1.place relop.op id2.place 'goto�too nextstar + 3); EMIT (E.place ':=' '0') EMIT ('goto' nextstat + 2) EMIT (E.place ':=' '1') }
E → TRUE	{E.place := newtemp(); Emit (E.place ':=' '1') }
E → FALSE	{E.place := newtemp(); Emit (E.place ':=' '0') }

Translation of Boolean Statement

Consider the grammar:

E → E OR E
E → E AND E
E → NOT E
E → (E)
E → id relop id
E → TRUE
E → FALSE



The relop is denoted by <, >, <, >.
a or b and not c

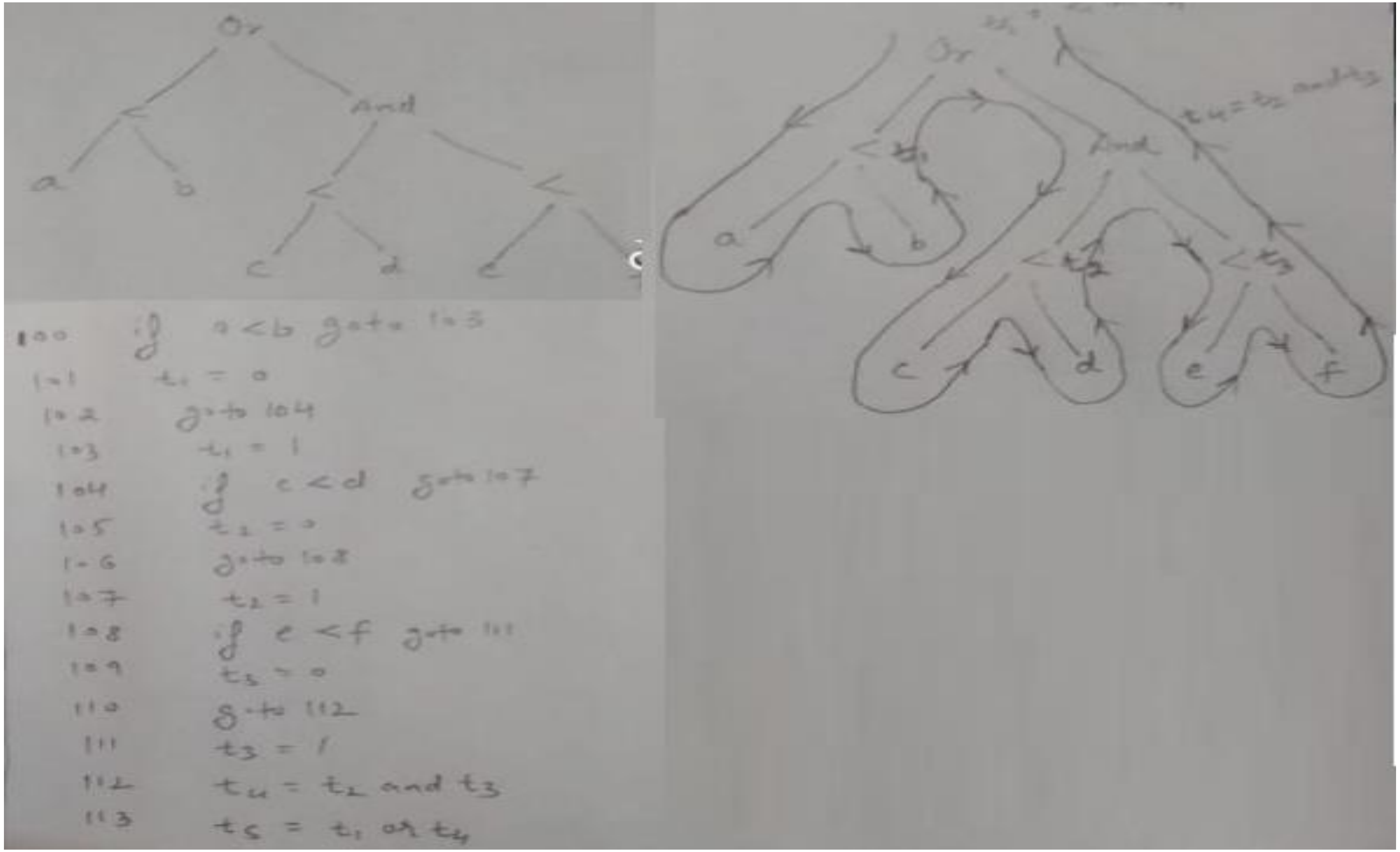
Translation of Boolean Statement

Consider the grammar:

E → E OR E
E → E AND E
E → NOT E
E → (E)
E → id relop id
E → TRUE
E → FALSE

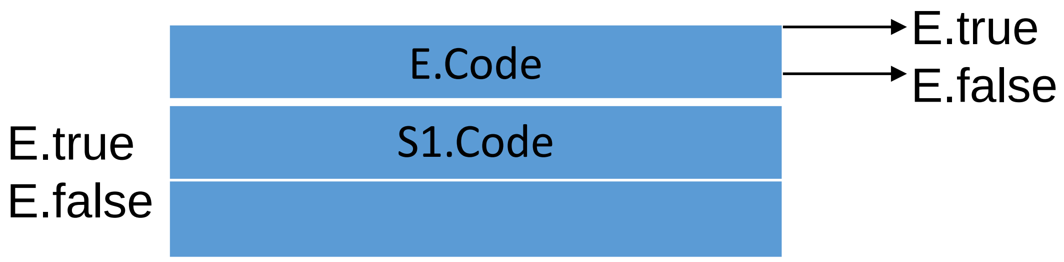
The relop is denoted by <, >, <, >.

a<b or c<d and e<f



Translation of Statement that alter the flow of control

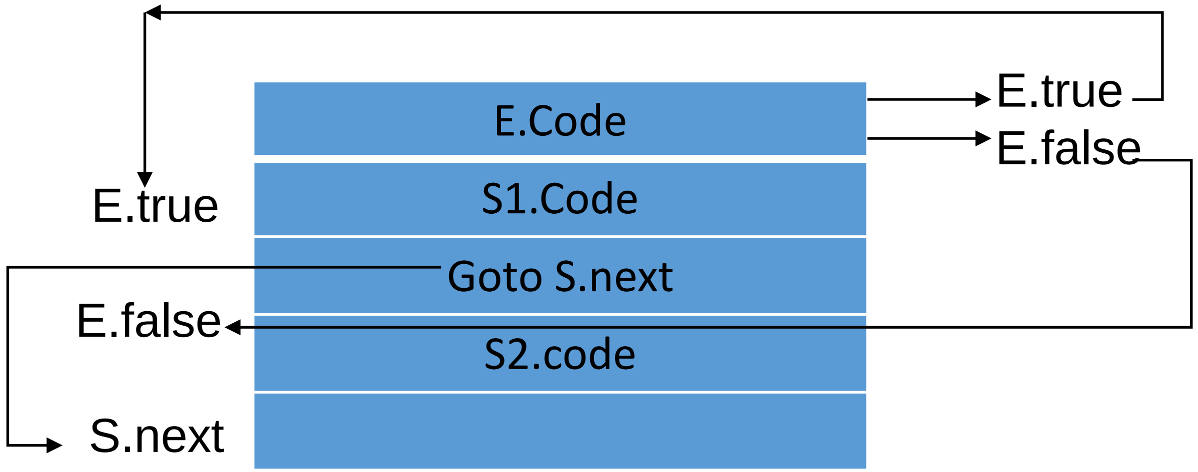
- If statement, if-else statement, loop statement, goto statements are example of statements which alter the flow of control.
- For implementation of these statements we require Label for a statement where these statements want to transfer the control.
- S->if E then S1
- In this grammar E is the Boolean expression, if its true then S1 will be executed otherwise next statement after S will be executed



- Semantic action for S->if E then S1
- E.true=newlabel
- E.false=S.next
- S1.next=S.next
- S.code=E.code||gen(E.true':')||S1.code

Translation of Statement that alter the flow of control

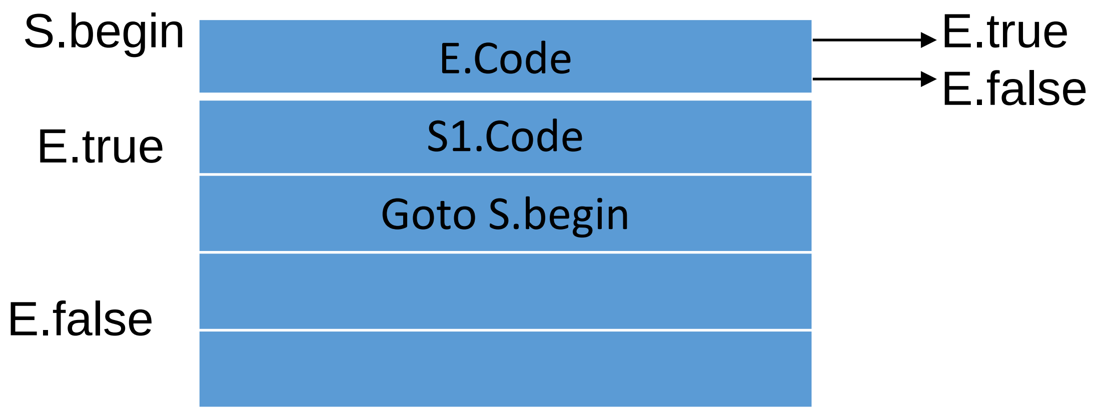
- S->if E then S1 else S2
- In this grammar E is the Boolean expression, if its true then S1 will be executed otherwise S2 will be executed and after execution of either of S1 or S2, next statement after S will be executed.



- Semantic action for S->if E then S1 else S2
- E.true=newlabel
- E.false=newlabel
- S1.next=S.next
- S2.next=S.next
- S.code=E.code||gen(E.true':')||S1.code||gen(E.false':')||S2.code

Translation of Statement that alter the flow of control

- S->While E do S1
- In this grammar E is the Boolean expression, statement S1 will be executed till the expression E is true. It will stop the execution when E becomes false and execute the statement next to S.



- Semantic action for S->while E do S1
- S.begin=newlabel
- E.true=newlabel
- E.false=S.next
- S1.next=S.begin
- S.code=Gen(S.begin ”:”)||E.code||Gen(E.true “:”)||S1.code||Gen(goto S.begin)

Postfix Translation

- By far the simplest SDD implementation occurs when we can parse the grammar bottom-up and the SDD is S-attributed.
- In that case, we can construct an SDT in which each action is placed at the end of the production and is executed along with the reduction of the body to the head of that production.
- SDT's with all actions at the right ends of the production bodies are called postfix SDT's

One Dimensional Array reference in arithmetic expression

System Generated Base Address B

Width 4 bytes (w)

Actual Address in Memory	1100	1104	1108	1112	1116
Element	235	33	345	786	188
Address with respect to indexing	0	1	2	3	4

Lower Bound LB

Address of A[i]=B+w(i-LB)

e.g. A[3]=1100+4(3-0)=1112

One Dimensional Array reference in arithmetic expression

Example:

```
int i, a[10];
i=1
while(i<10)
{
    a[i]=0;
    i=i+1;
}
```

1. i=1
2. If i<10 goto (4)
3. goto (8)
4. t1=4*i
5. a[t1]=0
6. i=i+1
7. goto (2)
8. stop

Two Dimensional Array reference in arithmetic expression

	0	1	2	3
0	8	6	3	2
1	4	5	9	1
2	6	3	2	4

A[M][N]

8	6	3	2	4	5	9	1	6	3	2	4

Row major representation

$A[i][j] = B + w*[N*(i-Lr)+(j-Lc)]$ here B is base address, w is width, N is number of columns, Lr is lower bound of row and Lc is lower bound of column

8	4	6	6	5	3	3	9	2	2	1	4

Column major representation

$A[i][j] = B + w*[(i-Lr)+M*(j-Lc)]$ here B is base address, w is width, M is number of rows, Lr is lower bound of row and Lc is lower bound of column

Two Dimensional Array reference in arithmetic expression

Example:

```
int i, a[10][10];
i=0
while(i<10)
{
        a[i][i]=0;
        i=i+1;
}
```

1. i=0
2. If i<10 goto (4)
3. goto (8)
4. t1=44*i
5. a[t1]=0
6. i=i+1
7. goto (2)
8. stop

Procedure Call

Procedure call of the form P(x1,x2,…xn) with the parameters x1,x2,….xn is represented in 3 address format as:

  - Param x1
  - Param x2
  - .
  - .
  - .
  - Param xn
  - Call P, n
- Here param is representing parameters, P is the name of procedure which is taking n number of actual parameters

Procedure Call

Example:

```
Void main()
{
        Int x,y;
        Swap(&x,&y);
}
Void swap(int *a, int *b)
{
        Int i;
        i=*b;
        *b=*a;
        *a=i;
}
```

1. Call main
2. Param &x
3. Param &y
4. Call swap,2
5. i=*b
6. *b=*a
7. *a=i
8. stop

Declaration

Whenever we encounter declaration statement, we need to lay out storage for the declared variables. For every local name in a procedure, we create a ST(Symbol Table) entry containing:

- The type of the name
- How much storage the name requires

Production rule	Semantic action
D → integer, id	ENTER (id.PLACE, integer) D.ATTR = integer
D → real, id	ENTER (id.PLACE, real) D.ATTR = real
D → D1, id	ENTER (id.PLACE, D1.ATTR) D.ATTR = D1.ATTR

- ENTER is used to make the entry into symbol table and ATTR is used to trace the data type.

Case statement

Example:

```
Switch(ch)
{
    Case '1': c=a+b;
    Break;
    Case '2': c=a*b;
    Break;
    Case '3': c=a-b;
    Break;
    Default: c=a/b;
    Break;
}
```

1. If ch='1' goto 5
2. If ch='2' goto 7
3. If ch='3' goto 9
4. goto 11
5. c=a+b
6. goto 12
7. c=a*b
8. goto 12
9. c=a-b
10. goto 12
11. c=a/b
12. stop