UNIT-3

1. SDTS of Assignment Statement into TAC

A→ id := E

E →E+E | E*E | E-E | (E) | id

Production	Semantic Rule
A →id : = E	A .code = E. code || gen(id.place ' =' E . place)
E→ E1+E2	T= newTemp( ); E.place : = T; E.code : = E1.code || E2.code || gen(E.place ' =' E1.place '+' E2.place)
E→ E1*E2	T= newTemp( ); E.place : = T; E.code : = E1.code || E2.code || gen(E.place ' =' E1.place '*' E2.place)
E→ E1-E2	T= newTemp( ); E.place : = T; E.code : = E1.code || E2.code || gen(E.place '=' E1.place '-' E2.place)
E→ -E1	T= newTemp( ); E.place : = T; E.code : = E1.code || gen(E.place ' =' '-' E1.place)
E→(E)	E.place : = E1.place E.code : = E1.code
E→id	E.place : = id.place E.code : = null

2. SDTS for Boolean expressions (Short Circuit Method)

Grammar: E→E or E | E and E | not E | (E) | id relop id | true | false

Production	Semantic Rule
E→E1 or M1 E2	backpatch (E1.false,M.quad): E.false=E2.false E.true= merge(E1.true, E2. true);
E→E1 and M1 E2	backpatch (E1.true,M.quad); E.true=E2.true; E.false= merge(E1.false, E2.false);
E→not E1	E.true=E1.false; E.false= E.true;
M→ε	M.quad = nextquad

Translation scheme for other statements

Production	Semantic Rule
E→id	E.place= id.place;

E→E1 relop E2	E.true = makelist(nextquad); E.false= makelist(nextquad+1); gen (if E1.place relop.val E2.place goto____); gen(goto ____);
E→(E)	E.true = E1.true; E.false= E1.false;
A →id : = E	A .code = E. code || gen(id.place ' =' E . place)

3. SDTS into Three-Address Code for programming language control structures using backpatching

Production	Semantic Rule
S→if E then M S1	backpatch(E.true,M.quad); S.next= merge(E.false,S1.next);

S→if E then M1 S1 N else M2 S2	backpatch(E.true,M1.quad);
	backpatch(E.false,M2.quad);
	S.next=merge(S1.next, merge(N.next, S2.next);

S→while M1 E do M2 S1	backpatch(S1.next, M1.quad);
	backpatch(E.true,M2.quad);
	S.next= E.false; gen('goto' M1.quad)

S→do M1 S1 while M2 E	backpatch(S1.next,M2.Quad);
	backpatch(E.true = M1.Quad);
	S.next = E.false;

M→ε	M.quad = nextquad;
N→ε	N.next= makelist(nextquad), gen(goto __)

Translation scheme for other statements

Production	Semantic Rule
S→begin L end	S.next = L.next;
S→A	S.next = nil;
L→L1; M S	backpatch(L1.next , M.quad); L.next = S.next;
L→S	L.next = S.next;
L →ε	L . next = S.next;

Translation scheme- for loop, repeat-until

Production	Semantic Rule

S→ for (A1; M1 E ; M2A2)M3 S1	backpatch(E.true,M3.quad);
	backpatch(S1.next, M2.quad);
	backpatch(M3.next, M1.quad);
	S.next= E.false;
	Gen('goto' M2.quad);

S→Repeat M1 S1 until M2 E	backpatch(S1.next = M2.Quad);
	backpatch(E.False = M1.Quad);
	S.next = E.True;

M1→ε	M1.quad = nextquad;
M2→ε	M2.quad = nextquad;
M3→ε	M3.next= makelist(nextquad); gen(goto __); M3.quad=nextquad;

4. Translation scheme for arrays

Production	Semantic Rule

A→L=E	If (L.offset = null) then
	gen(L.place '=' E.place)
	else
	gen(L.place'['L.offset'] ' '=' E. place);

E→E+E	T=newTemp( ); E.place=T; gen(E.place '=' E1.place '+' E2.place)
E→(E)	E.place= E1.place

E→ L	if (L.offset = null) then
	E.place = L.place
	else begin
	T = Newtemp ( );
	E.place=T;
	gen(T '=' L.place '[' L.offset ']');
	end

L→elist]	T= newtwmp( ); U = newTemp( ); L.place = T; L.offset = U; gen (T '=' elist.Array-c) gen (U '=' w * elist.place); where c = [d2 * d3 *…dk + d3 * d4….dk+…..dk+1] * w.
L→ id	L.place = id.place L.offset = null

Elist → elist, E	T = Newtemp ();
	m= elist.NDIM + 1;
	gen (T= elist1.place '*' Limit(elist1.array,m));
	gen (T=T+E.place);
	elist.array = elist1.array;
	elist.place=T;
	elist.NDIM = m;

Elist → id [E	elist.place = E.place;
	elist.array = id.place;
	elist.ND1M = 1;

5. Translation scheme for switch-case

Production	**Semantic Rule**

S→ Switch E N begin caselist end	backpatch(N.next,nextquad);
	for each entry in caselist.Q
	gen(if E.place=caselist.Q.V goto caselist.Q.L;
	if caselist.d ≠ null
	gen(goto caselist.d)
	S.Next = caselist.next;

caselist → case V : S	caselit. next =makelist(nextquad);
	caselist.next = merge(caselist.next, S.next);
	enter(caselist.Q,V.place,V.next);
	gen(goto__);

caselist → caselist case V : S	caselist.next = makelist(nextquad);
	caselist.next = merge(caselist.next,Caselist1.next, S.next);
	gen (goto__);
	Enter(caselist.Q,V.place,V.next);

caselist → caselist default M : S	caselist.next = makelist(nextquad)
	caselist.next = merge(caselist.next,caselist1.next, S.next);
	gen(goto__);
	caselist.d = M.quad;

V→id	V.place = id.place; V.next = nextquad;
N→ε	N.next = nextquad; gen(goto__);
M→ε	M.quad = Nextquad;

6. **Translation scheme for procedure call**

Production	Semantic Rule
S→ call id(elist)	for each item p an queue do gen (‘parameter’ p); gen (‘call’ id place);
elist→ elist , E	append E, place to end of Queue
elist→ E	initialize Queue to contain only E.place

7. **SDTS for declarations**

Production	Semantic Rule
D→ integer id	Enter (id.type=integer); D.attr = integer;
D→ real id	Enter (id.type=real); D.attr = real;
D→ D , id	Enter (id.place= D.attr); D.attr = D1.attr;