Three Address Code Generation for Control Statements

RobinAnil (04CS3005)

- INTRODUCTION :-
  - The basic idea of converting any flow of control statement to a three address code is to simulate the "branching" of the flow of control.
  - This is done by skipping to different parts of the code (label) to mimic the different flow of control branches.
  - Flow of control statements may be converted to three address code by use of the following functions:-
    - ✓ newlabel – returns a new symbolic label each time it is called.
    - ✓ gen () – "generates" the code (string) passed as a parameter to it.
  - The following attributes are associated with the non-terminals for the code generation:-
    - ✓ code – contains the generated three address code.
    - ✓ true – contains the label to which a jump takes place if the Boolean expression associated (if any) evaluates to "true".
    - ✓ false – contains the label to which a jump takes place if the Boolean expression (if any) associated evaluates to "false".
    - ✓ begin – contains the label / address pointing to the beginning of the code chunk for the statement "generated" (if any) by the non-terminal.

- EXAMPLES:-

  Lets try converting the following c code

FOR LOOP	**in 3 TA code**

a=3;
b=4;
for(i=0;i<n;i++){
  a=b+1;
  a=a*a;
}
c=a;
	a=3;
	b=4;
	i=0;
	L1:
	VAR1=i<n;
	if(VAR1) goto L2;
	goto L3;
	L4: i++;
	goto L1;
	L2: VAR2=b+1;
	a=VAR2;

	VAR3=a*a; a=VAR3; goto L4 L3: c=a;

WHILE Loop
 	
a=3;
b=4;
i=0;
while(i<n){
a=b+1;
a=a*a;
i++;
}
c=a;
	in 3 TA code
	
	a=3;
	b=4;
	i=0;
	L1:
	VAR1=i<n;
	if(VAR1) goto L2;
	goto L3;
	
	L2: VAR2=b+1;
	a=VAR2;
	VAR3=a*a;
	a=VAR3;
	i++;
	goto L1
	
	L3: c=a;

DO WHILE Loop
 	
a=3;
b=4;
i=0;
do{
a=b+1;
a=a*a;
i++;
}while(i<n);
c=a;
	in 3 TA code
	
	a=3;
	b=4;
	i=0;
	L1:
	VAR2=b+1;
	a=VAR2;
	VAR3=a*a;
	a=VAR3;
	i++;
	
	VAR1=i<n;
	if(VAR1) goto L1;
	goto L2;
	
	L2: c=a;

Suppose we have the following grammar:-

S --> if E then S_1
S --> if E then S_1 else S_2
S --> while E do S_1

- SEMANTIC RULES:-

  - S --> if E then S_1

```
{
    E.true := newlabel ;
    E.false := S.next ;
    S1.next := S.next ;
    S.code := E.code || gen(E.true ':') || S1.code
}
```

  - S --> if E then S_1 else S_2

```
{
    E.true := newlabel ;
    E.false := newlabel ;
    S1.next := S.next ;
    S2.next := S.next ;
    S.code := E.code || gen(E.true ':') || S1.code || gen('goto' S.next) || gen(E.false ':') ||
    S2.code
}
```

  - S --> while E do S_1

```
{
    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)
 }
```
