P-> {offset=0} D S

{S.next= newlabel()
P.code= S.code || label(S.next)}

$P \rightarrow$ { $offset = 0$; } D

$D \rightarrow T$ **id** ; { $top.put(\mathbf{id}.lexeme, T.type, offset); offset = offset + T.width;$ } D_1

$D \rightarrow \epsilon$

$T \rightarrow B$ { $t = B.type; w = B.width;$ } C {T.type = C.type; T.width = C.width}

$T \rightarrow$ **record** '{' { $Env.push(top); top =$ **new** $Env(); Stack.push(offset); offset = 0;$ } D '}' { $T.type = record(top); T.width = offset; top = Env.pop(); offset = Stack.pop();$ }

$B \rightarrow$ **int** { $B.type = integer; B.width = 4;$ }

$B \rightarrow$ **float** { $B.type = float; B.width = 8;$ }

$C \rightarrow \epsilon$ { $C.type = t; C.width = w;$ }

$C \rightarrow$ [**num**] C_1 { $C.type = array(\mathbf{num}.value, C_1.type); C.width = \mathbf{num}.value \times C_1.width;$ }

$S \rightarrow \mathbf{id} = E$; { $gen(top.get(\mathbf{id}.lexeme) '=' E.addr);$ }

s.code= L.code||

| $L = E$; { $gen(L.addr.base '[' L.addr ']' '=' E.addr);$ }

$E \rightarrow E_1 + E_2$ { $E.addr =$ **new** $Temp();$
$gen(E.addr '=' E_1.addr '+' E_2.addr);$ }

| **id** { $E.addr = top.get(\mathbf{id}.lexeme);$ }

L.code||

| L { $E.addr =$ **new** $Temp();$
$gen(E.addr '=' L.array.base '[' L.addr ']');$ }

$L \rightarrow \mathbf{id}$ [E] { $L.array = top.get(\mathbf{id}.lexeme);$
$L.type = L.array.type.elem;$
$L.addr =$ **new** $Temp();$
$gen(L.addr '=' E.addr '*' L.type.width);$ }

| L_1 [E] { $L.array = L_1.array;$
$L.type = L_1.type.elem;$
$t =$ **new** $Temp();$
$L.addr =$ **new** $Temp();$
$gen(t '=' E.addr '*' L.type.width);$ }
$gen(L.addr '=' L_1.addr '+' t);$ }

Figure 6.22: Semantic actions for array references

PRODUCTION	SEMANTIC RULES
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \mid\mid label(S.next)$
$S \rightarrow$ **assign**	$S.code =$ **assign**$.code$
$S \rightarrow$ **if** (B) S_1	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \mid\mid label(B.true) \mid\mid S_1.code$
$S \rightarrow$ **if** (B) S_1 **else** S_2	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ S.next)$ $\mid\mid label(B.false) \mid\mid S_2.code$ || label(S.next)
$S \rightarrow$ **while** (B) S_1	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ begin)$ || label(B.false)
$S \rightarrow S_1\ S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \mid\mid label(S_1.next) \mid\mid S_2.code$

Figure 6.36: Syntax-directed definition for flow-of-control statements.

PRODUCTION	SEMANTIC RULES
$B \rightarrow B_1 \mid\mid B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.false) \mid\mid B_2.code$
$B \rightarrow B_1 \;\&\&\; B_2$	$B_1.true = newlabel()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \mid\mid label(B_1.true) \mid\mid B_2.code$
$B \rightarrow ! B_1$	$B_1.true = B.false$ $B_1.false = B.true$ $B.code = B_1.code$
$B \rightarrow E_1$ **rel** E_2	$B.code = E_1.code \mid\mid E_2.code$ $\mid\mid gen('\texttt{if}'\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ '\texttt{goto}'\ B.true)$ $\mid\mid gen('\texttt{goto}'\ B.false)$
$B \rightarrow$ **true**	$B.code = gen('\texttt{goto}'\ B.true)$
$B \rightarrow$ **false**	$B.code = gen('\texttt{goto}'\ B.false)$

Figure 6.37: Generating three-address code for booleans

PRODUCTION	SEMANTIC RULES
$S \rightarrow \mathbf{id} = E\ ;$	$S.code = E.code \mid\mid$ $gen(top.get(\mathbf{id}.lexeme)\ '='\ E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = \mathbf{new}\ Temp()$ $E.code = E_1.code \mid\mid E_2.code \mid\mid$ $gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$
$\mid\ - E_1$	$E.addr = \mathbf{new}\ Temp()$ $E.code = E_1.code \mid\mid$ $gen(E.addr\ '='\ '\mathbf{minus}'\ E_1.addr)$
$\mid\ (\ E_1\ )$	$E.addr = E_1.addr$ $E.code = E_1.code$
$\mid\ \mathbf{id}$	$E.addr = top.get(\mathbf{id}.lexeme)$ $E.code = ''$

Figure 6.19: Three-address code for expressions