Accessing fields in records

→ Compiler tracks both the types and the relative addresses of the fields of a record.

→ Keeping this info. in symbol table entries for the field names → routine for looking up names can also be used for field names.

→ t → pointer to the symbol table entry for a record type

record(t) → returned as T.type.

p↑.info + 1

- p → pointer to a record with a field name 'info'
- type(p) → pointer(record(t))
- type(p↑) → record(t)

→ 'info' is looked up in the symbol table pointed to by t.

Boolean Expressions

- Used as conditional expressions in statements that alter the flow of control.
- operators - 'and', 'or', 'not' applied to boolean variables or relational expressions.
- E → E or E | E and E | not E | (E) | id relop id | true | false
- or, and - left associative precedence: ~~!~~, and, or.

Two methods for translating boolean expressions

(i) Numerical Representation

a or b and not c

t_1 = not c

t_2 = b and t_1

t_3 = a or t_2

a < b ⇔ if a < b then 1
else 0

```
100: if a<b goto 103
101: t = 0
102: goto 104
103: t = 1
104:
```

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' nextstat +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'); }

```
100: if a<b goto 103
101: t1 = 0
102: goto 104
103: t1 = 1
104: if c<d goto 107
105: t2 = 0
106: goto 108
107: t2 = 1
```

```
108: if e<f goto 111
109: t3 = 0
110: goto 112
111: t3 = 1
112: t4 = t2 and t3
113: t5 = t1 or t4
```

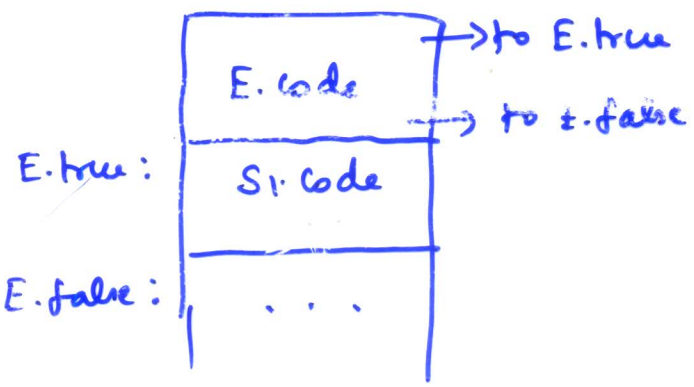
⇒ a<b or c<d and e<f

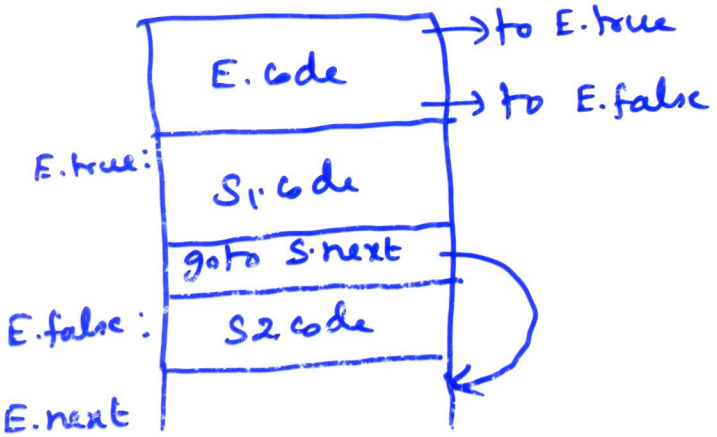
'Short-circuiting'

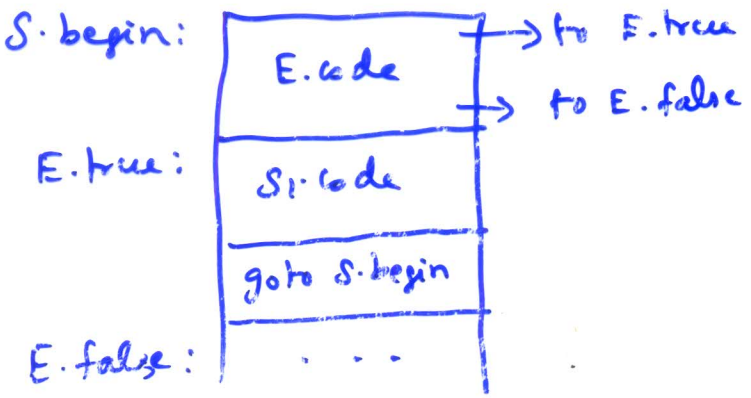
t_1 need not be stored as the control flow through 101 or 103 determines t_1

Flow of Control Statements

S → if E then S_1 | if E then S_1 else S_2 | while E do S_1







$S \rightarrow$ if E then S_1

E.true = newlabel;
E.false = S.next;
S_1.next = S.next
S.code = E.code || gen('E.true':') ||
S_1.code

$S \rightarrow$ if E then S_1 else S_2

E.true = newlabel;
E.false = newlabel;
S_1.next = S.next;
S_2.next = S.next;
S.code = E.code || gen(E.true':') ||
S_1.code || gen('goto' S.next) ||
gen(E.false':') || S_2.code

$S \rightarrow$ while E do S_1

S.begin = newlabel;
E.true = newlabel;
E.false = S.next;
S_1.next = S.begin;
S.code = gen(S.begin':' || E.code ||
gen(E.true':') || S_1.code ||
gen('goto' S.begin)

Example:

a<b or c<d and e<f

Ltrue / Lfalse → true and false exits for the entire expression

if a<b goto Ltrue
goto L1
L1: if c<d goto L2
goto Lfalse

L2: if e<f goto Ltrue
goto Lfalse

$E \rightarrow E_1$ or E_2

$E_1.true = E.true$;

$E_1.false = newlabel$;

$E_2.true = E.true$;

$E_2.false = E.false$;

$E.code = E_1.code \,||\, gen(E_1.false\ ':') \,||\, E_2.code$

$E \rightarrow E_1$ and E_2

$E_1.true = newlabel$;

$E_1.false = E.false$

$E_2.true = E.true$

$E_2.false = E.false$

$E.code = E_1.code \,||\, gen(E_1.true\ ':') \,||\, E_2.code$

$E \rightarrow$ not E_1

$E_1.true = E.false$;

$E_1.false = E.true$;

$E.code = E_1.code$;

$E \rightarrow (E_1)$

$E_1.true = E.true$;

$E_1.false = E.false$;

$E.code = E_1.code$;

$E \rightarrow id_1\ relop\ id_2$

$E.code = gen('if'\ id_1.place\ relop.op\ id_2.place\ 'goto'\ E.true) \,||\, gen('goto'\ E.false)$

$E \rightarrow true$

$E.code = gen('goto'\ E.true)$

$E \rightarrow false$

$E.code = gen('goto'\ E.false)$

```
while a<b do
    if c<d then
        x = y+z
    else
        x = y-z
```

```
L1: if a<b goto L2
    goto Lnext
L2: if c<d goto L3
    goto L4
L3: t1 = y+z
    x = t1
    goto L1
L4: t2 = y-z
    x = t2
    goto L1
Lnext:
```

Mixed mode boolean expressions

E → E+E | E and E | E relop E | id

E.type = arith

if E_1.type = arith and E_2.type = arith then

E.place = newtemp

E.code = E_1.code || E_2.code || gen(E.place '=' E_1.place '+' E_2.place)

end

else if E_1.type = arith and E_2.type = bool then