Project 9

File You Will Modify:

Generator.java

New Files:

IR.java (slight changes)
tst directory
runAll
Main.jar

Copy all p8 files into a new directory ("p9").

Didn't finish project 8?

Must finish p8 before starting p9.
Must pass all tests before moving on.

Generating Code For Expressions

All the methods that generate code for expressions...

`genExpr`
`genBinaryOp`
`genUnaryOp`
...etc...

The place will be:
Temporary or normal variable
VarDecl
Formal

...must do two things:

- Generate IR code to evaluate the expression and place the value into some variable
- Return the variable (i.e., return the synthesized "place" attribute)

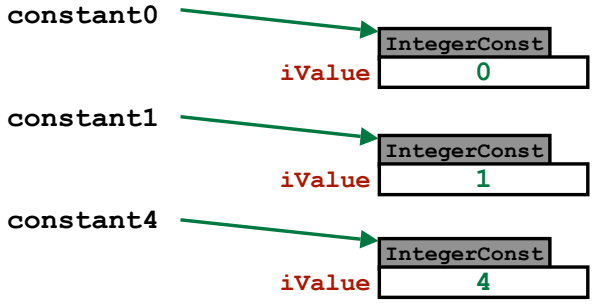
To handle short-circuit code, will add 2 additional parameters.

```
Ast.Node genExpr (Ast.Expr p)
Ast.Node genBinaryOp (Ast.Expr p)
Ast.Node genUnaryOp (Ast.Expr p)
...
```

```
Ast.Node genExpr (Ast.Expr p, String trueLabel, String falseLabel)
Ast.Node genBinaryOp (Ast.Expr p, String trueLabel, String falseLabel)
Ast.Node genUnaryOp (Ast.Expr p, String trueLabel, String falseLabel)
...
```

Handy Constants

Initialize these:



Translating Boolean Expressions

Assume we have these IR instructions...

```
if x < y then goto Label_43
if x <= y then goto Label_43
...
x := y AND z
x := y OR z
x := NOT y
```

Encoding Boolean Values

	Option 1	Option 2	Option 3
FALSE:	0	0	<0
TRUE	1	≠0	≥0

Example Boolean Expressions ("Conditional Expressions")

```
c AND (d OR NOT e)
b AND (x < y)
```

Used as R-Value

```
b := c AND (d OR NOT e);
```

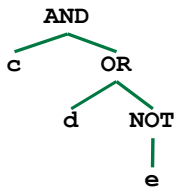
Used to control execution

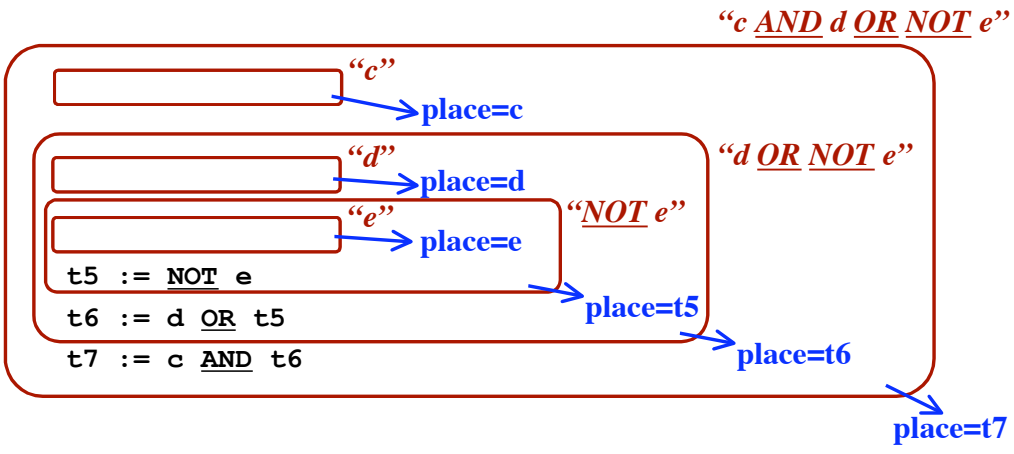
```
if c AND (d OR NOT e) then
   ...
end;
```

Let's start by assuming...

No Short-Circuit Evaluation

Example: c AND (d OR NOT e)





Syntax Directed Translations

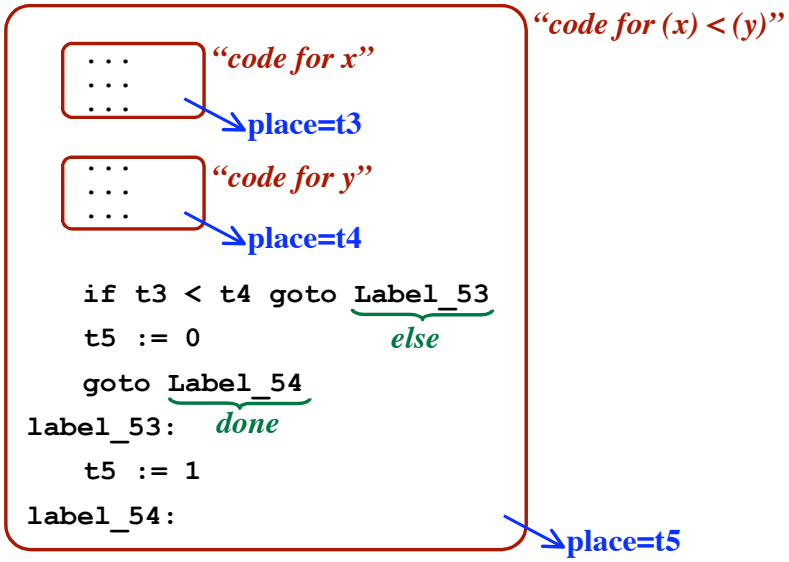
$E_0 \rightarrow E_1 + E_2$	E_0.place := ... E_0.code := ...
$E_0 \rightarrow (E_1)$	E_0.place := E_1.place E_0.code := E_1.code
$E_0 \rightarrow$ TRUE	E_0.place := NewTemp () E_0.code := IR (E_0.place, '**:= 1**')
$E_0 \rightarrow$ FALSE	E_0.place := NewTemp () E_0.code := IR (E_0.place, '**:= 0**')
$E_0 \rightarrow$ **ID**	E_0.place := ID.svalue E_0.code := " "

Syntax Directed Translations

$E_0 \rightarrow E_1$ AND E_2	E_0.place := NewTemp () E_0.code := E_1.code ‖ E_2.code ‖ IR (E_0.place, '**:=**', E_1.place, '**AND**', E_2.place)
$E_0 \rightarrow E_1$ OR E_2	E_0.place := NewTemp () E_0.code := E_1.code ‖ E_2.code ‖ IR (E_0.place, '**:=**', E_1.place, '**OR**', E_2.place)
$E_0 \rightarrow$ NOT E_1	E_0.place := NewTemp () E_0.code := E_1.code ‖ IR (E_0.place, '**:= NOT**', E_1.place)

Relational Operators

Example: `(...x...) < (...y...)`

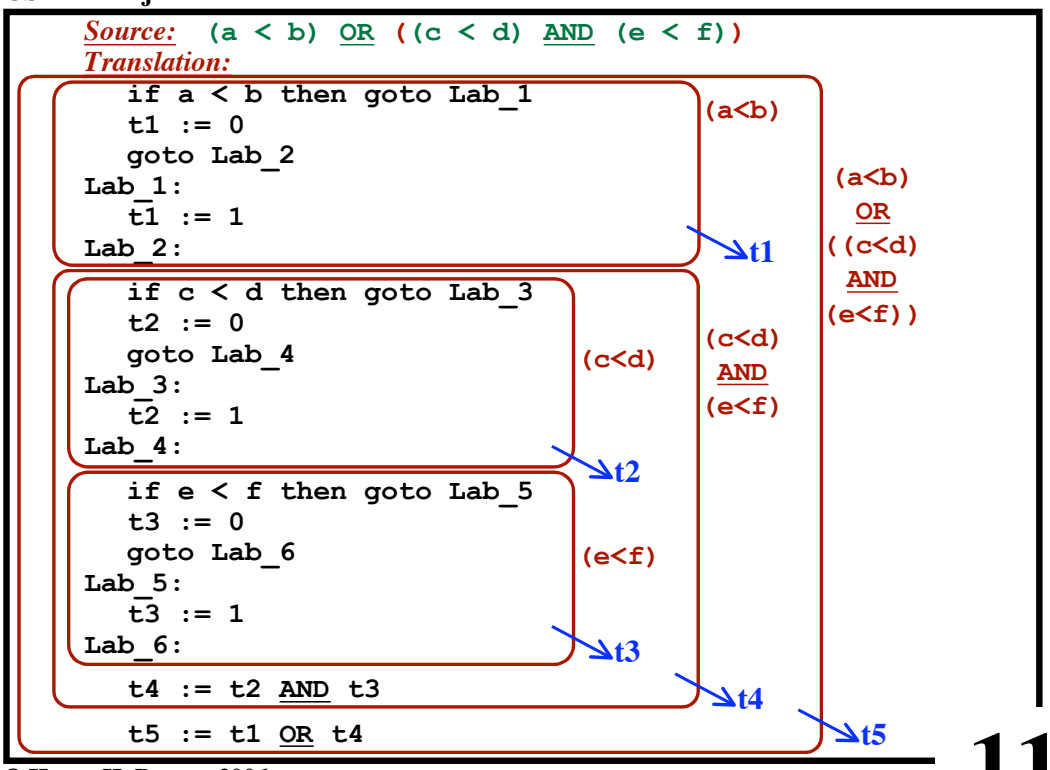


$E_0 \rightarrow E_1$ RELOP E_2

E_0.place := NewTemp ()
LabElse := NewLabel ()
LabEnd := NewLabel ()
E_0.code := E_1.code ||
E_2.code ||
IR (E_1.place, '**RELOP**', E_2.place,
'**then goto**', LabElse) ||
IR (E_0.place, '**:= 0**') ||
IR ('**goto** ', LabDone) ||
IR (LabElse, '**:**') ||
IR (E_0.place, '**:= 1**') ||
IR (LabDone, '**:**')

Our Code Will Look More Like This:

```
t0 = newTemp();
labElse = newLabel();
labDone = newLabel();
t1 = genExpr(expr1);
t2 = genExpr(expr2);
IR.gotoiLT(t1,t2,labElse);
IR.assign(t0,constant0);
IR.goto(labDone);
 ...etc...
return t0;
```



GenExpr is passed an expression and will generate code for it.
GenExpr will return "place" attribute:

- The variable holding the result

IDEA:

We'll pass 2 more parameters to GenExpr.
"inherited attributes"
These will be labels to use during code generation.

E.trueLabel
E.falseLabel

Java Strings
e.g., "`Label_58`"

Case 1: Both labels are NULL.

GenExpr will work as before.
(GenExpr will return name of variable holding result.)

Case 2: Both labels will be non-NULL.

GenExpr will generate code such that...
when executed the last instruction to be executed
will branch to either
"trueLabel" (if the condition is TRUE)
"falseLabel" (if the condition is FALSE)
GenExpr will return NULL (i.e., "place" will not be used).

Source: "c < d"

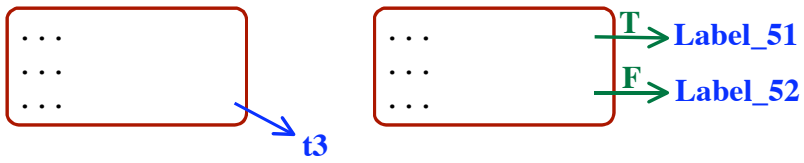
Case 1: Both labels are NULL

```
        if c < d then goto Label_51
        t7 := 0
        goto Label_52
   Label_51:
        t7 := 1
   Label_52:
```

Case 2: Both labels are non-NULL

```
        if c < d then goto trueLabel
        goto falseLabel
```

Notation:



$E_0 \rightarrow E_1$ RELOP E_2

if E_0.trueLabel == NULL **then**

Case 1

E_0.place := NewTemp ()
LabElse := NewLabel ()
LabEnd := NewLabel ()
E_0.code := E_1.code ||
E_2.code ||
IR (E_1.place, '**RELOP**', E_2.place,
'**then goto**', LabElse) ||
IR (E_0.place, '**:= 0**') ||
IR ('**goto** ', LabDone) ||
IR (LabElse, '**:**') ||
IR (E_0.place, '**:= 1**') ||
IR (LabDone, '**:**')

Case 2

else
...
endIf

$E_0 \rightarrow E_1$ RELOP E_2

Case 1

if E_0.trueLabel == NULL **then**

...

else

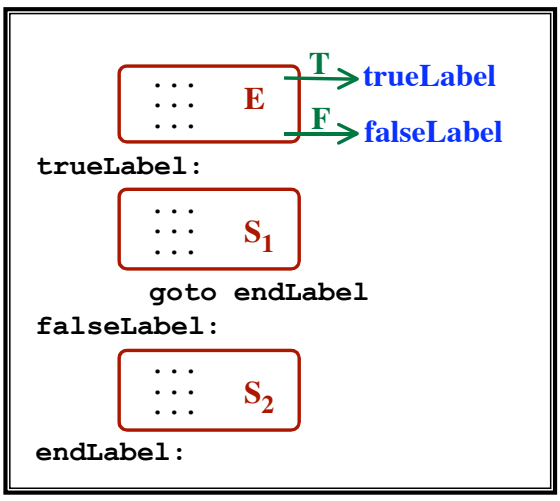
Case 2

E_0.code := E_1.code ||
E_2.code ||
IR (E_1.place, '**RELOP**', E_2.place,
'**then goto**', E_0.trueLabel) ||
IR ('**goto** ', E_0.falseLabel)

endIf

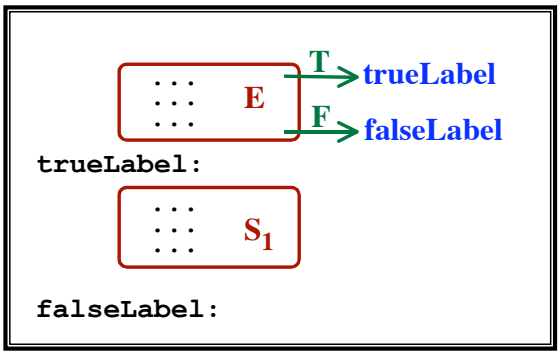
$S_0 \rightarrow$ if E then S_1 else S_2 endIf

E.trueLabel := NewLabel ()
E.falseLabel := NewLabel ()
endLabel := NewLabel ()
S_0.code := E.code ||
IR (E.trueLabel, '**:**') ||
S_1.code ||
IR ('**goto** ', endLabel) ||
IR (E.falseLabel, '**:**') ||
S_2.code ||
IR (endLabel, '**:**')



$S_0 \rightarrow$ if E then S_1 endIf

E.trueLabel := NewLabel ()
E.falseLabel := NewLabel ()
S_0.code := E.code ||
IR (E.trueLabel, ':') ||
S_1.code ||
IR (E.falseLabel, ':')



Source:

```
if a < b then
   ...Stmts...
end;
```

Translation:

```
if a<b goto Label_5
goto Label_6
Label_5:
   ...
   ...Stmts...
   ...
Label_6:
```


Short-Circuit Evaluation

Don't evaluate an operand unless necessary.

- `x AND y` ← *Don't evaluate if x = ???*
- `x OR y` ← *Don't evaluate if x = ???*
- `NOT x` ← *No optimization*

Short-Circuit Evaluation

Don't evaluate an operand unless necessary.

- `x AND y` ← *Don't evaluate if x = ???*
- `x OR y` ← *Don't evaluate if x = ???*
- `NOT x` ← *No optimization*

Exclusive-Or (XOR)

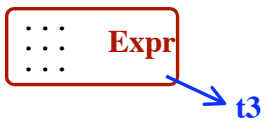
x	y	x XOR y
0	0	0
0	1	1
1	0	1
1	1	0

Equivalence (=)

x	y	x = y
0	0	1
0	1	0
1	0	0
1	1	1

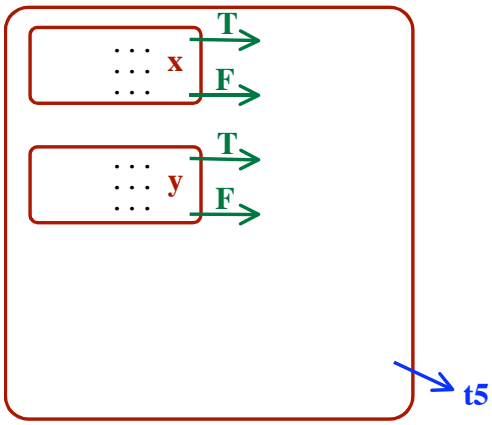
Can we use short-circuit evaluation
... for XOR?
... for EQUIVALENCE?

Notation:

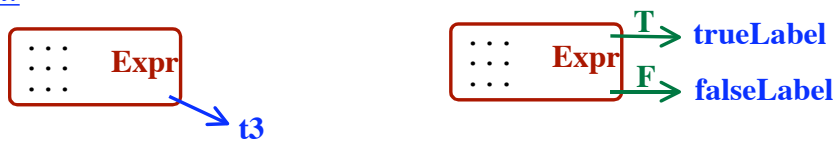


Short-Circuit Code for `(...x...) AND (...y...)`

Case 1: Both labels are NULL

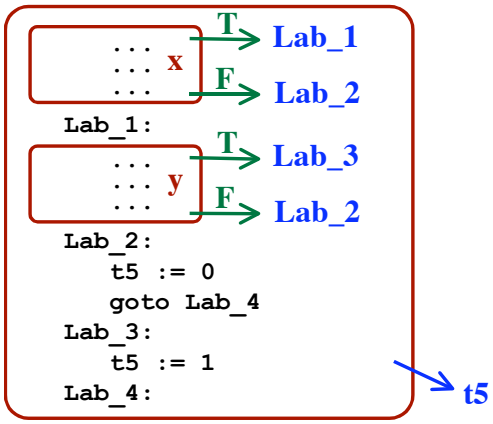


Notation:



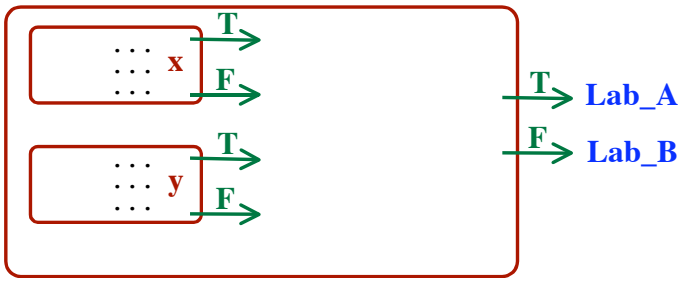
Short-Circuit Code for `(...x...) AND (...y...)`

Case 1: Both labels are NULL



Short-Circuit Code for `(...x...) AND (...y...)`

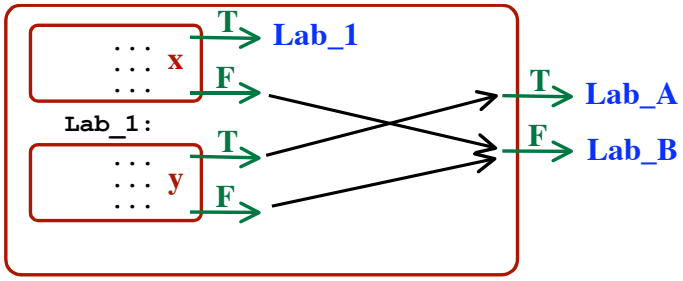
Case 2: Both labels are non-NULL
(say "Lab_A" and "Lab_B")



The Code for "OR" is similar!

Short-Circuit Code for `(...x...) AND (...y...)`

Case 2: Both labels are non-NULL
(say "Lab_A" and "Lab_B")

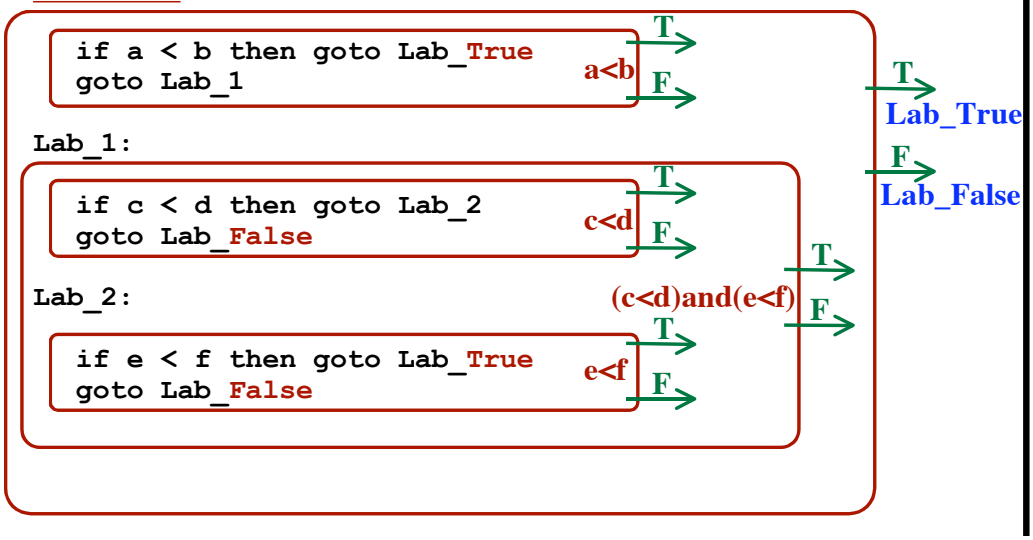


The Code for "OR" is similar!

Example

Source: `(a < b) OR ((c < d) AND (e < f))`

Translation:



Example

Source: `(a < b) OR ((c < d) AND (e < f))`

Translation:

```
   if a < b then goto Lab_True
   goto Lab_1
Lab_1:

   if c < d then goto Lab_2
   goto Lab_False
Lab_2:

   if e < f then goto Lab_True
   goto Lab_False
```

Easy to optimize

Can we optimize this?

Eliminating Unnecessary "Goto"s

Given:

```
   goto Lab_XXX
Lab_XXX:
```

Re-write as:

```
Lab_XXX:
```

Leave this in (unless sure that there are no branches to it)

Opposites:

<	↔	>=
<=	↔	>
>	↔	<=
>=	↔	<
=	↔	!=
!=	↔	=

Given:

```
   if AAA < BBB then goto Lab_XXX
   goto Lab_YYY
Lab_XXX:
```

Re-write as:

```
   if AAA >= BBB then goto Lab_YYY
Lab_XXX:
```

Leave this in (unless sure that there are no branches to it)

Example

Source: `(a < b) OR ((c < d) AND (e < f))`

Translation:

```
   if a < b then goto Lab_True
   goto Lab_1

Lab_1:
```

Easy to optimize

```
   if c < d  then goto Lab_2
   goto Lab_False

Lab_2:
```

Can we optimize this?

```
   if e < f then goto Lab_True
   goto Lab_False
```


Example

Source: `(a < b) OR ((c < d) AND (e < f))`

Translation:

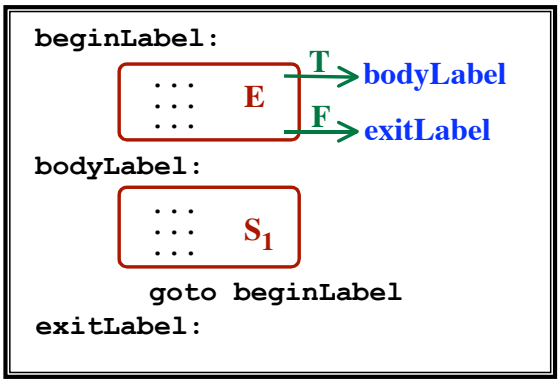
```
   if a < b then goto Lab_True



   if c >= d then goto Lab_False



   if e < f then goto Lab_True
   goto Lab_False
```

14 instructions ⟶ 4 instructions!

Translating While Statements

$S_0 \rightarrow$ while E do S_1 endWhile

```
beginLabel := NewLabel ()
E.trueLabel := NewLabel ()    -- bodyLabel
E.falseLabel := NewLabel ()   -- exitLabel
S0.code :=  IR (beginLabel, ':') ||
            E.code ||
            IR (E.trueLabel, ':') ||
            S1.code ||
            IR ('goto ', beginLabel) ||
            IR (E.falseLabel, ':')
```



Translating "For" Stmts

Source:

```
for L := E1 to E2 [ by E3 ] do Stmts end ;
```

It is okay to modify the loop index variable:

```
for j := x+4 to y*foo() by z-6 do
  ...
  if ... then
    j := j + w;
  end;
  ...
end;
```

Could involve arbitrary computation
(Must evaluate expressions EXACTLY once.)

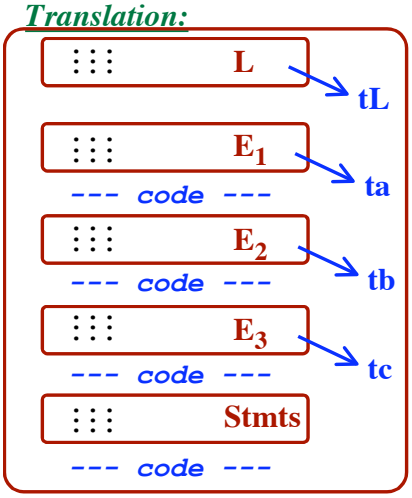
In other langauges, the L-Value may be something complex:

```
for a[i] := ... to ... by ... do
  ...
end;
```

First: get a pointer to the L-Value.
Then: use that whenever we access the variable

Translating "For" Stmts

Translation:



For Stmt

Source:

```
for L := E1 to E2 [ by E3 ] do Stmts end ;
```

We will also need:

3 temporaries

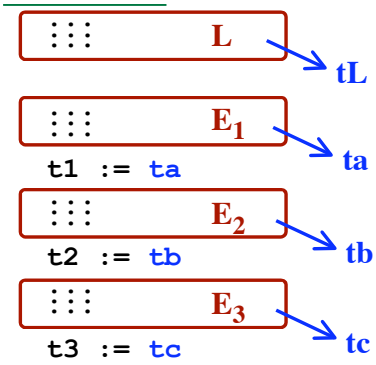
(t1, t2, t3)

2 labels

(loopLabel, exitLabel)

Translating "For" Stmts

Translation:




```
   *tL := t1
Loop_Label:
   if t1 > t2 then goto Exit_Label
```


```
   t1 := *tL
   t1 := t1+t3
   *tL := t1
   goto Loop_Label
Exit_Label:
```

Source:

```
for L := E1 to E2 [ by E3 ] do Stmts end ;
```

Translating "For" Stmts

Source:

for L := E_1 **to** E_2 [**by** E_3] **do** Stmts **end** ;

Translation:

```
   :::              L   → tL
   :::              E1  → ta
   t1  := ta
   :::              E2  → tb
   t2  := tb
   :::              E3  → tc
   t3  := tc
   *tL  := t1
Loop_Label:
   if t1 > t2 then goto Exit_Label
   :::              Stmts
   t1  := *tL
   t1  := t1+t3
   *tL  := t1
   goto Loop_Label
Exit_Label:
```

If E_3 is missing, generate this:

```
t3  := 1
```