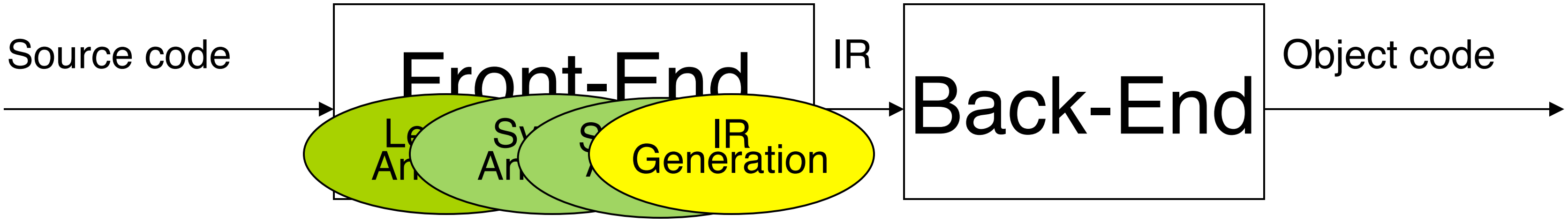
Compiler Design

Lecture 9:
Three-Address Code Generation

Dr. Momtazi
momtazi@aut.ac.ir

based on the slides of the course book

Intermediate Representation Generation.



IR Generation

- Goal: Translate the program into the format expected by the compiler back-end.

Outline

- Introduction
- Syntax-Directed Translation
- Code Generation
- Representations
- **More Structures of Code Generation**

More Structures of Code Generation

- Booleans and Reloperators
- While
- If
- Array
- Function
- Function call

Boolean Expressions

- Boolean expressions compute logical values

- Often used with flow-of-control statements

- Methods of translating boolean expression:
  - Numerical methods
  - Flow-of-control methods

Boolean Expressions: Numerical

- Numerical
  - ƒTrue is represented as 1 and false is represented as 0
  - Nonzero values are considered true and zero values are considered false

Boolean Expressions: Numerical

- Example:
  - a or b and not c

```
t1 = not c
t2 = b and t1
t3 = a or t2
```

Boolean Expressions: Numerical

- Example:
  - $b = x + x < y$

```
t1 = x + x
t2 = t1 < y
b = t2
```

Boolean Expressions: Numerical

- Example:
  - a < b → if a < b then 1 else 0

```
100: if a < b goto 103
101: t1 = 0
102: goto 104
103: t1 = 1
104:
```

Code Generation for Boolean Expressions

PRODUCTION	SEMANTIC RULES

$E \longrightarrow E_1$ **or** E_2	E.place := newtemp;
	emit(E.place ':=' E_1.place 'or'E_2.place)

$E \longrightarrow E_1$ **and** E_2	E.place := newtemp;
	emit(E.place ':=' E_1.place 'and'E_2.place)

$E \longrightarrow$ **not** E_1	E.place := newtemp;
	emit(E.place ':=' 'not' E_1.place)

$E \longrightarrow (E_1)$	E.place := E_1.place;

Code Generation for Boolean Expressions

PRODUCTION	SEMANTIC RULES

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')

Boolean Expressions: Numerical

- Example:
  - a < b or c < d and e < f

```
100: if a < b goto 103
101: t1 = 0
102: goto 104
103: t1 = 1
104: ???
```

Boolean Expressions: Numerical

- Example:
  - a < b or c < d and e < f

```
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
```

Boolean Expressions: Numerical

- Example:
  - a < b or c < d and e < f

```
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
```

Boolean Expressions: Numerical

- Example:
  - a < b or c < d and e < f

```
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
```

Boolean Expressions: Numerical

- Example:
  - a < b or c < d and e < f

```
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
```

Boolean Expressions

- Boolean variables are represented as integers that have zero or nonzero values.
- In addition to the arithmetic operator, TAC supports <, ==, or, and

- Example:
  - b = (x <= y)

Boolean Expressions

- Boolean variables are represented as integers that have zero or nonzero values.
- In addition to the arithmetic operator, TAC supports <, ==, or, and

- Example:
  - b = (x <= y)

```
t0 = x < y;
t1 = x == y;
b = t0 or t1;
```

Control Flow Statements

- The function newlabel will return a new symbolic label each time it is called
- Each boolean expression will have two new attributes:
  - E.true is the label to which control flows if E is true
  - E.false is the label to which control flows if E is false
- Attribute S.next of a statement S:
  - Inherited attribute whose value is the label attached to the first instruction to be executed after the code for S
  - Used to avoid jumps to jumps

Labels

- TAC allows for **named labels** indicating particular points in the code that can be jumped to.
- There are two control flow instructions:
  - Goto *label*;
  - If *value* Goto *label*;
- Note that If is always paired with Goto

Boolean Expressions

- Methods of translating boolean expression:
  - Numerical methods
  - **Flow-of-control methods**

Boolean Expressions: Flow-of-control methods

- Flow-of-control methods:
  - Represent the value of a boolean by the position reached in a program
  - Often not necessary to evaluate entire expression

Code Generation for Boolean Expressions

PRODUCTION	SEMANTIC RULES
E —> 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 II gen(E1.false ':') II E_2.code
E —> 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 II gen(E1 .true ':') II E_2 .code
E —> **not** E_1	E_1.true := E.false; E_1.false := E.true; E.code := E_1.code
E —> (E_1)	E_1.true := E.true; E_1.false := E.false; E.code := E_1.code

Code Generation for Boolean Expressions

PRODUCTION	SEMANTIC RULES

E —> id1 relop id2	E.code := gen('if' id.place
	relop.op id2 .place 'goto‘ E.true) II
	gen('goto' E.false)

E —> **true**	E.code := gen('goto' E.true)
E —> **false**	E.code := gen('goto' E.false)

Boolean Expressions: Flow-of-control

- Example:
  - a < b or c < d and e < f

```
if a < b goto $E_1$true
goto  $E_1$false
```

Boolean Expressions: Flow-of-control

- Example:
  - a < b or c < d and e < f

if a < b goto ~~E_1true~~ Etrue

goto ~~E_1false~~ L1

Boolean Expressions: Flow-of-control

- Example:
  - a < b or c < d and e < f

if a < b goto ~~E_1true~~ Etrue

goto ~~E_1false~~ L1

L1: if c < d goto E_2true

goto E_2false

Boolean Expressions: Flow-of-control

- Example:
  - a < b or c < d and e < f

if a < b goto ~~E_1true~~ Etrue

goto ~~E_1false~~ L1

L1: if c < d goto ~~E_2true~~ L2

goto ~~E_2false~~ E_{23}false

Boolean Expressions: Flow-of-control

- Example:
  - a < b or c < d and e < f

if a < b goto ~~E_1true~~ Etrue

goto ~~E_1false~~ L1

L1: if c < d goto ~~E_2true~~ L2

goto ~~E_2false~~ E_{23}false

L2: if e < f goto E_3true

goto E_3false

Boolean Expressions: Flow-of-control

- Example:
  - a < b or c < d and e < f

if a < b goto ~~E_1true~~ Etrue

goto ~~E_1false~~ L1

L1: if c < d goto ~~E_2true~~ L2

goto ~~E_2false~~ E_{23}false

L2: if e < f goto ~~E_3true~~ E_{23}true

goto ~~E_3false~~ E_{23}false

Boolean Expressions: Flow-of-control

- Example:
  - a < b or c < d and e < f

if a < b goto ~~E_1true~~ Etrue

goto ~~E_1false~~ L1

L1: if c < d goto ~~E_2true~~ L2

goto ~~E_2false~~ ~~E_{23}false~~ Efalse

L2: if e < f goto ~~E_3true~~ ~~E_{23}true~~ Etrue

goto ~~E_3false~~ ~~E_{23}false~~ Efalse

Boolean Expressions: Flow-of-control

- Example:
  - a < b or c < d and e < f

```
        if a < b goto Etrue
        goto L1
L1:  if c < d goto L2
        goto Efalse
L2:  if e < f goto Etrue
        goto Efalse
```

Control Flow Statements

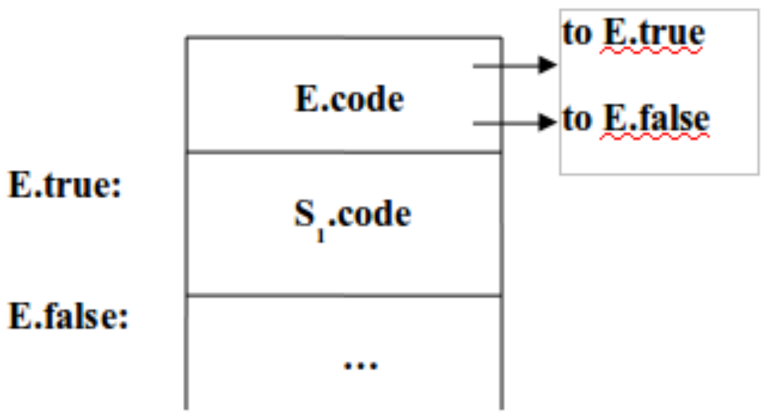
- Suppose we have the following grammar:
  - S → if E then S1
  - S → if E then S1 else S2
  - S → while E do S1
  - S → do S1 while E

Code Generation for Control Flow Statements

PRODUCTION	SEMANTIC RULES

S → if E then S1	{
	E.true := newlabel ;
	E.false := S.next ;
	S1.next := S.next ;
	S.code := E.code ll gen(E.true ':') ll S1.code
	}

Code Generation for Control Flow Statements



if - then

Code Generation for Control Flow Statements

- Example:

```
if ( x <  y )
    z = x;
```

Code Generation for Control Flow Statements

- Example:

```
if ( x <  y )
    z = x;
```

```
if x < y goto Etrue
goto  Efalse
```

Code Generation for Control Flow Statements

- Example:

```
if ( x <  y )
    z = x;
```

if x < y goto ~~Etrue~~ L1

goto ~~Efalse~~ Snext

Code Generation for Control Flow Statements

- Example:

if (x < y)
        z = x;

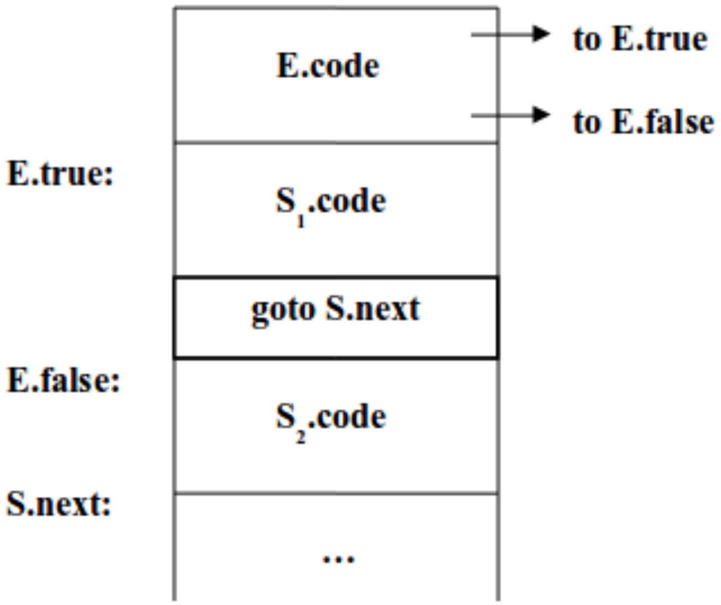
        if x < y goto ~~Etrue~~ L1
        goto ~~Efalse~~ Snext
L1:   z = x

Code Generation for Control Flow Statements

PRODUCTION	SEMANTIC RULES

S → if E then S1 else S2	{
	E.true := newlabel ;
	E.false := newlabel ;
	S1.next := S.next ;
	S2.next := S.next ;
	S.code := E.code II gen(E.true ':') II S1.code
	II gen('goto' S.next) II gen(E.false ':')
	IIS2.code
	}

Code Generation for Control Flow Statements



if – then - else

Code Generation for Control Flow Statements

- Example:

```
if ( x <  y )
    z = x;
else
    z = y;
```

Code Generation for Control Flow Statements

- Example:

```
if ( x <  y )
    z = x;
else
    z = y;
```

```
if x < y goto Etrue
goto  Efalse
```

Code Generation for Control Flow Statements

- Example:

```
if ( x <  y )
    z = x;
else
    z = y;
```

if x < y goto ~~Etrue~~ L1
goto ~~Efalse~~ L2

Code Generation for Control Flow Statements

- Example:

```
if ( x <  y )
      z = x;
else
      z = y;
```

```
        if x < y goto ~~Etrue~~   L1
        goto  ~~Efalse~~   L2
L1:     z = x
        goto Snext
```

Code Generation for Control Flow Statements

- Example:

```
if ( x <  y )
      z = x;
else
      z = y;
```

```
       if x < y goto ~~Etrue~~   L1
       goto  ~~Efalse~~   L2
L1:    z = x
       goto Snext
L2:    z = y
```

Code Generation for Control Flow Statements

- Example:

```
if ( x <  y )
      z = x;
else
      z = y;
```

```
        if x < y goto L1
        goto  L2
L1:     z = x
        goto Snext
L2:     z = y
```

Code Generation for Control Flow Statements

- Example:

```
if ( x <  y )
    z = x;
else
    z = y;
z = 2 * z;
```

Code Generation for Control Flow Statements

- Example:

```
if ( x <  y )
      z = x;
else
      z = y;
z = 2 * z;
```

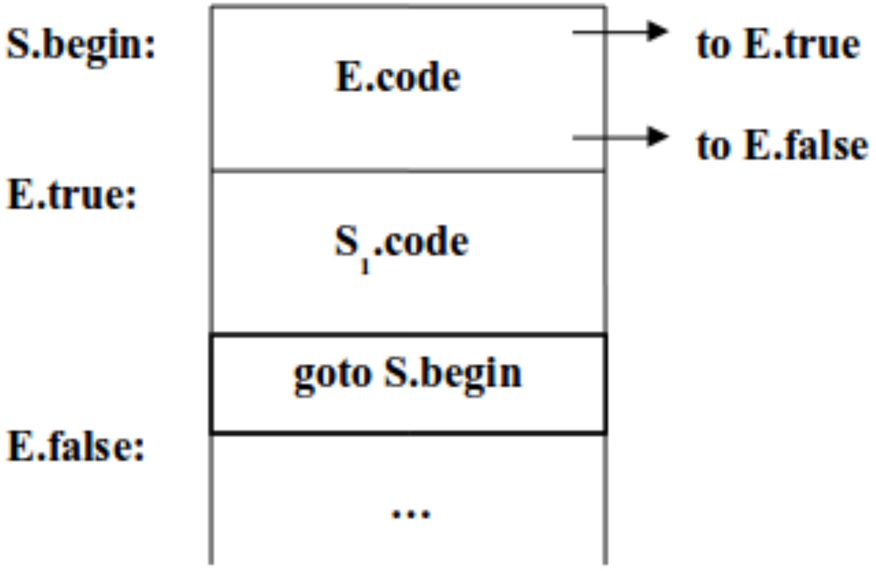
```
            if x < y goto L1
            goto  L2
L1:         z = x
             goto Snext
L2:         z = y
Snext:      z=2*z
```

Code Generation for Control Flow Statements

PRODUCTION	SEMANTIC RULES

S → while E do S1	{
	S.begin := newlabel ;
	E.true := newlabel ;
	E.false := S.next ;
	S1.next := S.begin ;
	S.code := gen(S.begin ‘:’) II E.code II gen(E.true ‘:’)
	II S1.code II gen(‘goto’ S.begin)
	}

Code Generation for Control Flow Statements



while - do

Code Generation for Control Flow Statements

- Example:

```
while ( x <  y )
        x = x + 2;
```

Code Generation for Control Flow Statements

- Example:

```
while ( x <  y )
        x = x + 2;
```

Sbegin => L1

```
L1:    if x < y goto Etrue
       goto  Efalse
```

Code Generation for Control Flow Statements

- Example:

```
while ( x <  y )
      x = x + 2;
```

Sbegin => L1

L1: if x < y goto ~~Etrue~~ L2

goto ~~Efalse~~ Snext

Code Generation for Control Flow Statements

- Example:

```
while ( x <  y )
        x = x + 2;
```

Sbegin => L1

```
L1:    if x < y goto ~~Etrue~~  L2
       goto  ~~Efalse~~  Snext
L2:    x = x + 2
       goto L1
```

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
    if ( c <  d )
            x = y + z;
    else
            x = y - z;
```

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
    if ( c <  d )
            x = y + z;
    else
            x = y - z;
```

Sbegin => L1

```
L1:   if a < b goto Etrue
      goto  Efalse
```

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
    if ( c <  d )
        x = y + z;
    else
        x = y - z;
```

Sbegin => L1

L1: if a < b goto ~~Etrue~~ L2
goto ~~Efalse~~ Snext

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
    if ( c <  d )
        x = y + z;
    else
        x = y - z;
```

Sbegin => L1

L1:   if a < b goto ~~Etrue~~ L2
        goto ~~Efalse~~ Snext
L2:

Code Generation for Control Flow Statements

- Example:

```
if ( x <  y )
    z = x;
else
    z = y;
```

REMINDER

```
        if x < y goto L1
        goto  L2
L1:     z = x
        goto Snext
L2:     z = y
```

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
    if ( c <  d )
            x = y + z;
    else
            x = y - z;
```

Sbegin => L1

L1: if a < b goto ~~Etrue~~ L2
goto ~~Efalse~~ Snext
L2:

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
    if ( c <  d )
            x = y + z;
    else
            x = y - z;
```

Sbegin => L1

```
L1:   if a < b goto ~~Etrue~~  L2
      goto  ~~Efalse~~  Snext
L2:   if c < d goto L3
      goto  L4
L3:   x = y + z
      goto Snext
L4:   x = y - z
```

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
    if ( c <  d )
        x = y + z;
    else
        x = y - z;
```

Sbegin => L1
S1.next := S.begin ;

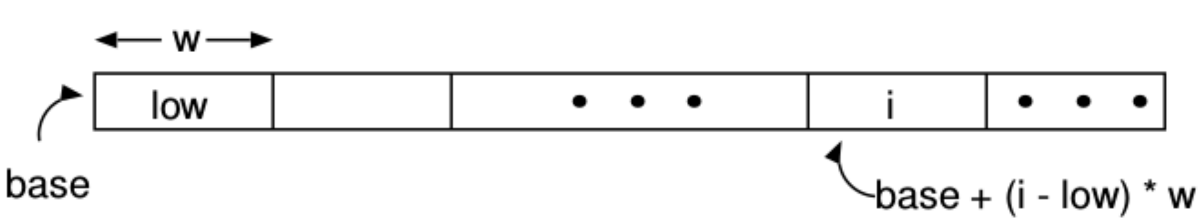
L1: if a < b goto ~~Etrue~~ L2
goto ~~Efalse~~ Snext
L2: if c < d goto L3
goto L4
L3: x = y + z
goto ~~Snext~~ L1
L4: x = y - z

Code Generation for Array

- Addressing Array Elements

- Two-Dimensional Arrays

Addressing Array Elements

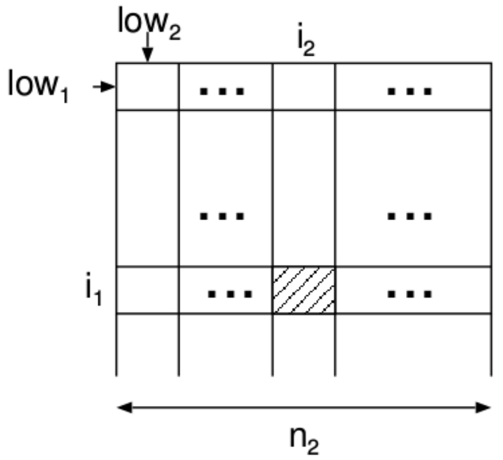


$$\begin{aligned} \text{Address of } A[i] &= base + w \times (i - low) \\ &= \underbrace{(base - w \times low)}_{c} + w \times i \end{aligned}$$

- First format
  - w is the width of each element
  - low is the lower bound of the subscript
  - base is the relative address of A[low]
- Second format
  - The subexpression in parentheses is a constant
  - That subexpression can be evaluated at compile time

Two-Dimensional Arrays

- Stored in row-major form



Address of $A[i_1, i_2]$

$$\begin{aligned} &= base + \\ &\quad w \times (i_1 - low_1) \times n_2 + \\ &\quad w \times (i_2 - low_2) \\ &= w \times (i_1 \times n_2 + i_2) + \\ &\quad \underbrace{\overbrace{base - w \times (low_1 \times n_2 + low_2)}} \end{aligned}$$

- First format
  - n2 = high2 – low2 + 1
- Second format
  - The last term can be computed at compile time

Code Generation for Array

- Example:
  - c + a[i][j]

  A is a 2x3 array of integers

```
t1 = i * 12
t2 = j * 4
t3 = t1 + t2
t4 = a [ t3 ]
t5 = c + t4
```

```
t1 = i * 12
t2 = j * 4
t3 = t1 + t2
t4 = addr(a)
t5 = t4 [ t3 ]
t6 = c + t5
```

Code Generation for Array:
Another Approach (not used in this course)

- Example
  - arr[1] = arr[0] * 2

```
t0 = 1
t1 = 4
t2 = t1 * t0
t3 = arr + t2
t4 = 0
t5 = 4
t6 = t5 * t4
```

```
t7 = arr + t6
t8 = *(t7)
t9 = t8 * 2
*(t3) = t9
```

Code Generation for Functions

- Function definition
- Function call

Function definition

- Functions consist of the following items:
  - A **label** identifying the start of the function.
  - A **BeginFunc** ***N*;** instruction reserving **N** bytes of space for locals and temporaries.
  - The body of the function.
  - **Return** (if needed)
  - An **EndFunc;** instruction marking the end of the function.
    - When reached, cleans up stack frame and returns.

Code Generation for Functions

```
void main()
{
        int a;
        a = 2 + a;
}
```

```
main:
    BeginFunc 12
    t1 = 2 + a
    a = t1
    EndFunc
```

Function definition

- Calling functions consists of three pieces:
  - PushParam
  - LCall
  - PopParams

Code Generation for Functions

```
void main()
{
        int a;
        a = 2 + a;
        Print(a);
}
```

```
main:
    BeginFunc 12
    t1 = 2 + a
    a = t1
    PushParam a
    LCall _PrintInt
    PopParams 4
    EndFunc
```

Code Generation for Functions

```
int foo(int a, int b){
        return a + b;
}

void main(){
        int c;
        int d;
        foo(c, d);
}
```

```
_foo:
   BeginFunc 4
   t0 = a + b
   Return t0
   EndFunc

main:
   BeginFunc 12
   PushParam d
   PushParam c
   LCall _foo
   PopParams 8
   EndFunc
```

Code Generation for Functions

```
void main()
{
        int b;
        int a;
        b = 3;
        a = 12;
        a = (b + 2)-(a*3)/6;
}
```

```
main:
    BeginFunc 44;
    _t0 = 3;
    b = _t0;
    _t1 = 12;
    a = _t1;
    _t2 = 2;
    _t3 = b + _t2;
    _t4 = 3;
    _t5 = a * _t4;
    _t6 = 6;
    _t7 = _t5 / _t6;
    _t8 = _t3 - _t7;
    a = _t8;
    EndFunc;
```

Compiling Function Calls

```
void SimpleFn(int z) {
    int x, y;
    x = x * y * z;
}

void main() {
   SimpleFunction(137);
}
```

Compiling Function Calls

```
void SimpleFn(int z) {
    int x, y;
    x = x * y * z;
}

void main() {
   SimpleFunction(137);
}
```

```
_SimpleFn:
   BeginFunc 16;
   _t0 = x * y;
   _t1 = _t0 * z;
   x = _t1;
   EndFunc;
```

Compiling Function Calls

```
void SimpleFn(int z) {
    int x, y;
    x = x * y * z;
}

void main() {
   SimpleFunction(137);
}
```

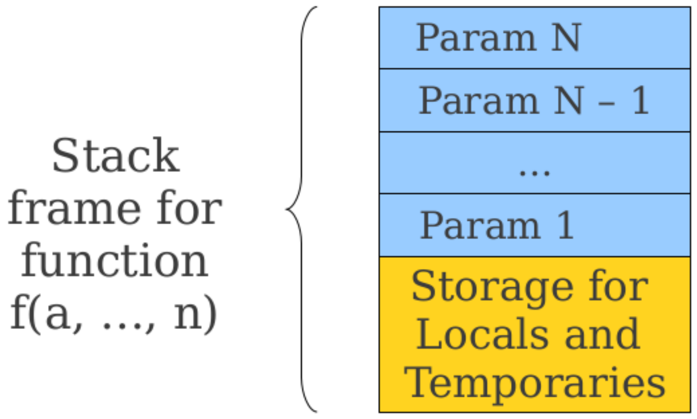
```
_SimpleFn:
   BeginFunc 16;
   _t0 = x * y;
   _t1 = _t0 * z;
   x = _t1;
   EndFunc;

main:
   BeginFunc 4;
   _t0 = 137;
   PushParam _t0;
   LCall _SimpleFn;
   PopParams 4;
   EndFunc;
```

Stack Management in TAC

- BeginFunc *N*;
  - Instruction only needs to reserve room for local variables and temporaries.
- EndFunc;
  - Instruction reclaims the room allocated with BeginFunc *N*;
- PushParam *var*
  - A single parameter is pushed onto the stack by the caller
- PopParams *N*;
  - Space for parameters is reclaimed by the caller
  - *N* is measured in *bytes*, not number of arguments.

Calling Sequence



Stack frame for function f(a, …, n)

Param N

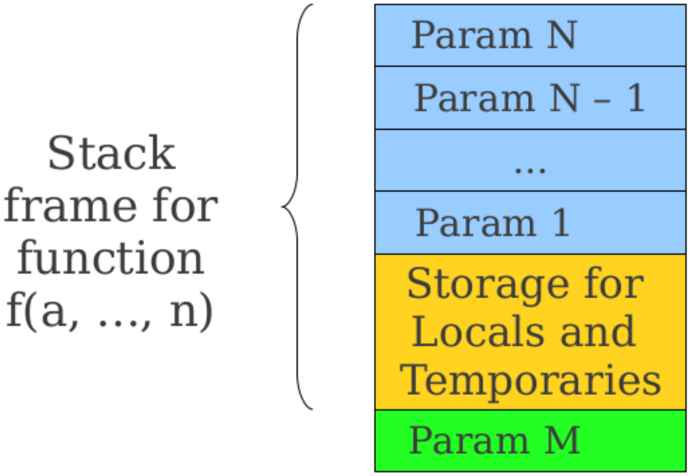
Param N – 1

…

Param 1

Storage for Locals and Temporaries

Calling Sequence



Stack frame for function f(a, …, n)

Param N

Param N – 1

…

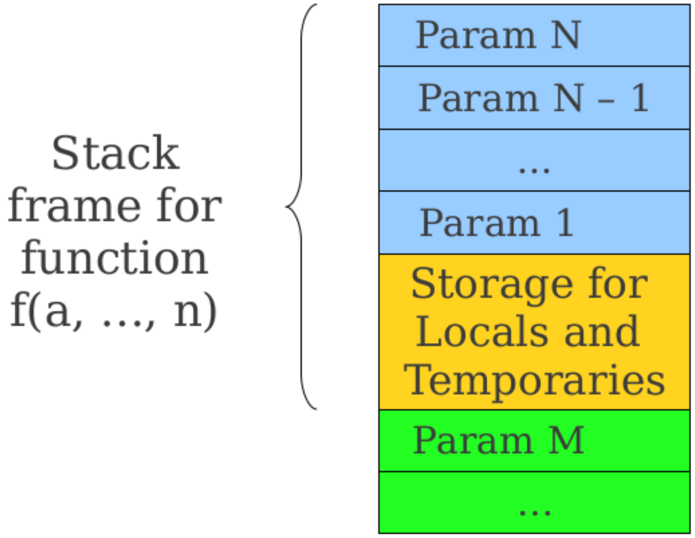
Param 1

Storage for Locals and Temporaries

Param M

```
PushParam var;
```

Calling Sequence



Stack frame for function f(a, …, n)

Param N

Param N – 1

…

Param 1

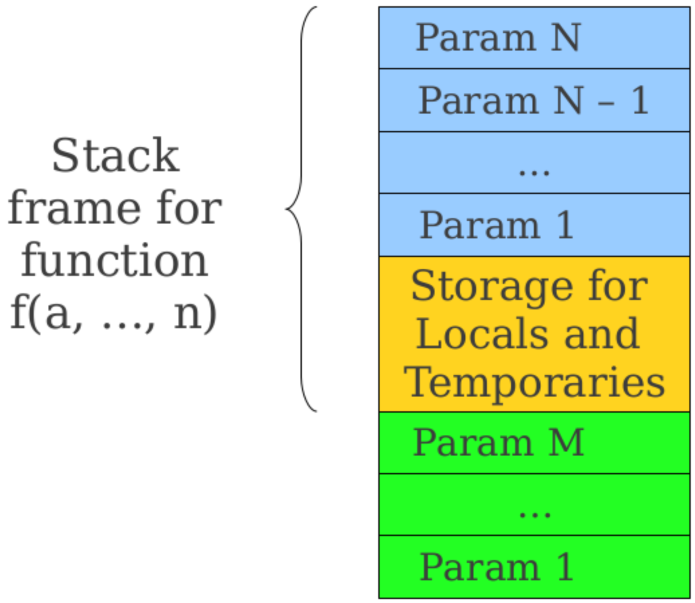
Storage for Locals and Temporaries

Param M

…

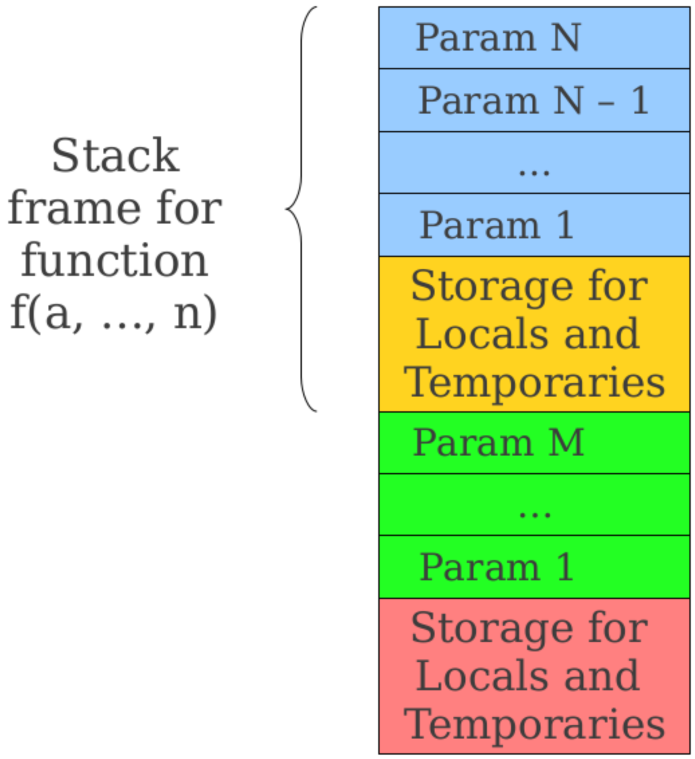
```
PushParam var;
PushParam var;
```

Calling Sequence


Stack frame for function f(a, …, n)
Param N
Param N – 1
…
Param 1
Storage for Locals and Temporaries
Param M
…
Param 1

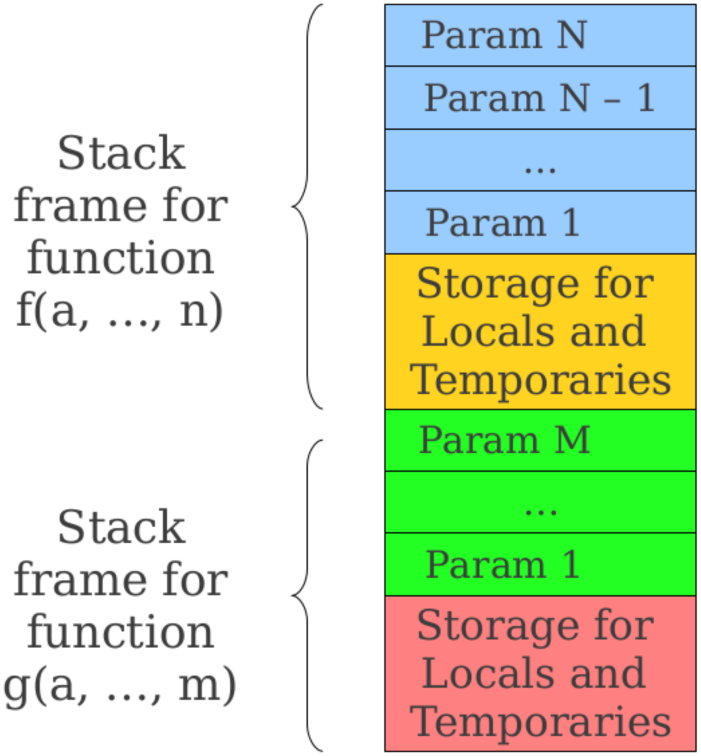
```
PushParam var;
PushParam var;
PushParam var;
```

Calling Sequence


Stack frame for function f(a, …, n)
Param N
Param N – 1
…
Param 1
Storage for Locals and Temporaries
Param M
…
Param 1
Storage for Locals and Temporaries

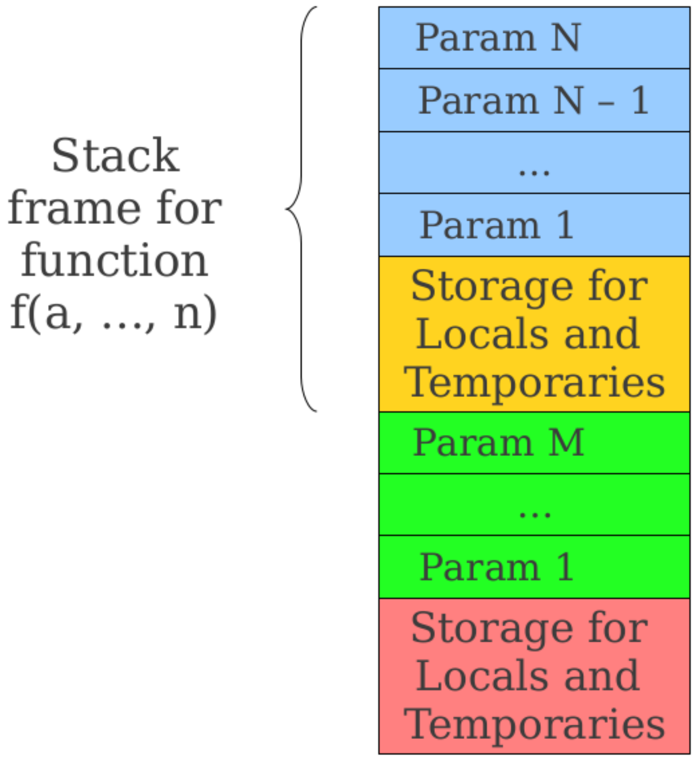
```
PushParam var;
PushParam var;
PushParam var;
BeginFunc N;
```

Calling Sequence


Stack frame for function f(a, …, n)
Param N
Param N – 1
…
Param 1
Storage for Locals and Temporaries
Stack frame for function g(a, …, m)
Param M
…
Param 1
Storage for Locals and Temporaries

```
PushParam var;
PushParam var;
PushParam var;
BeginFunc N;
```

Calling Sequence



Stack frame for function f(a, …, n)

Param N

Param N – 1

…

Param 1

Storage for Locals and Temporaries

Param M

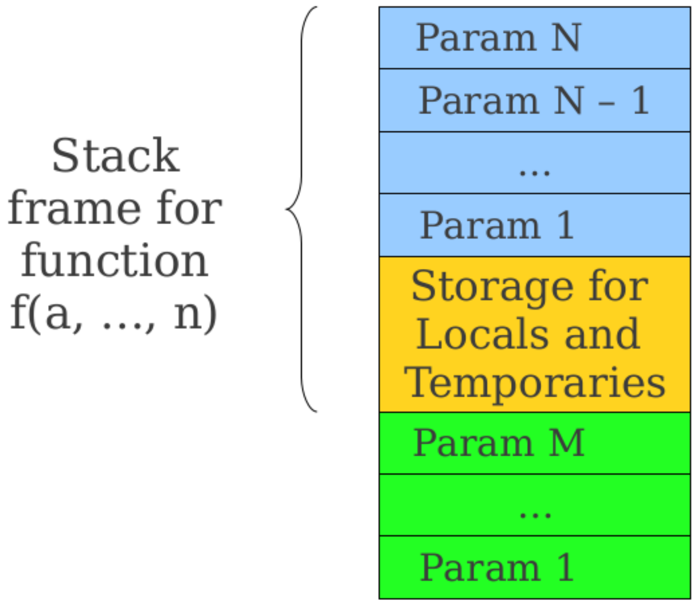
…

Param 1

Storage for Locals and Temporaries

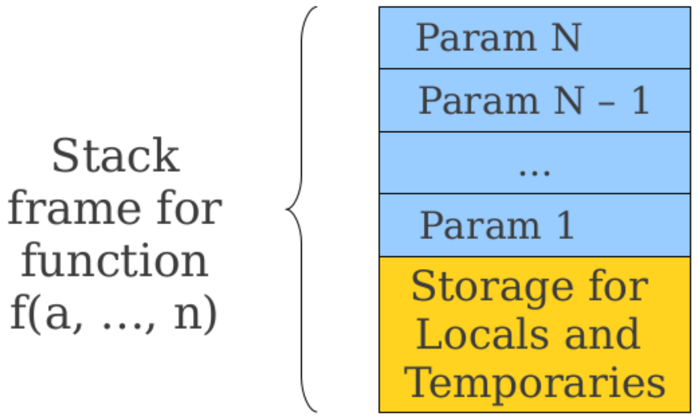
```
EndFunc;
```

Calling Sequence


Stack frame for function f(a, …, n)
Param N
Param N – 1
…
Param 1
Storage for Locals and Temporaries
Param M
…
Param 1

`PopParams N;`

Calling Sequence


Stack frame for function f(a, …, n)
Param N
Param N – 1
…
Param 1
Storage for Locals and Temporaries

Code Generation for Functions

- Example

```
int fact(int n){
      if (n==0) return 1;
      else return (n * fact(n-1));
}
```

Code Generation for Functions

- Example

```
int fact(int n){
    if (n==0) return 1;
    else return (n * fact(n-1));
}
```

```
fact:
    beginFunc  4
    if (n==0) goto L1
    goto L2
L1: return 1
    goto Lnext
L2: t1 = n-1
    PushParam t1
    t2 = LCall fact
    PopParams 4
    t3 = n * t2
    return t3
    goto Lnext
Lnext: endFunc
```

Case (Switch)

```
switch(E) {
        case $V_1$ : $S_1$
        case $V_2$ : $S_2$
        …
        case $V_{n-1}$ : $S_{n-1}$
        default: $S_n$
}
```

Case (Switch)

- Implemented as:
  - Sequence of if statements
  - Jump table

Case (Switch)

```
switch(E) {
        case $V_1$ : $S_1$
        case $V_2$ : $S_2$
        ...
        case $V_{n-1}$ : $S_{n-1}$
        default: $S_n$
}
```

```
        t = code to evaluate E
        goto Ltest
L1:     code for S1
        goto Lnext
L2:     code for S2
        goto Lnext
...
Ln-1:   code for Sn-1
        goto Lnext
Ln:     code for Sn
        goto Lnext
Ltest:  if t = V1  goto  L1
        if t = V2   goto  L2
        ...
        if t = Vn-1 goto Ln-1
        goto Ln
Lnext:
```

Case (Switch)

- The definition of a label is treated as a declaration of the label
- Labels are typically entered into the symbol table
  - Entry is created the first time the label is seen
  - This may be before the definition of the label if it is the target of any forward goto
- When a compiler encounters a goto statement:
  - It must ensure that there is exactly one appropriate label in the current scope
  - If so, it must generate the appropriate code; otherwise, an error should be indicated

Backpatching

- A key problem when generating code for Boolean expressions and flow-of-control statements is that of matching a jump instruction with the target of the jump.
- Two passes required to replace symbolic addresses (labels) in jump instructions by actual addresses

Backpatching

- Solution:
  - Putting all (forward) jump statements that have the same target on a list
  - Filling in actual address for each statement on list when the target address is known

Backpatching

- Backpatching uses lists of jumps which are passed as synthesized attributes.
- When a jump is generated, the target of the jump is temporarily left unspecified.
- Each such jump is put on a list of jumps whose labels are to be filled in when the proper label can be determined.
- Attributes:
  - E.tlist – all jumps (conditional / unconditional) to E.true
  - E.flist, S.nlist – analogous

Backpatching

- Generate instructions into an instruction array, and labels will be indices into this array. To manipulate lists of jumps, three functions are used:
- makelist(i)
- merge(pl , p2)
- backpatch (p, i)

Backpatching

- makelist(i)
  - creates a new list containing only i, an index into the array of instructions;
  - makelist returns a pointer to the newly created list.
- merge(pl , p2)
  - concatenates the lists pointed to by pl and p2 , and returns a pointer to the concatenated list.
- backpatch (p, i)
  - inserts i as the target label for each of the instructions on the list pointed to by p.

Code Generation for Boolean Expressions

PRODUCTION	SEMANTIC RULES
E —> 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(E1.false ':') || E_2.code
E —> 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(E1 .true ':') || E_2 .code
E —> **not** E_1	E_1.true := E.false; E_1.false := E.true; E.code := E_1.code
E —> (E_1)	E_1.true := E.true; E_1.false := E.false; E.code := E_1.code

Code Generation for Boolean Expressions

PRODUCTION	SEMANTIC RULES

E —> id1 relop id2	E.code := gen('if' id.place
	relop.op id2 .place 'goto‘ E.true) II
	gen('goto' E.false)

E —> **true**	E.code := gen('goto' E.true)
E —> **false**	E.code := gen('goto' E.false)

Backpatching for Boolean Expressions

PRODUCTION	SEMANTIC RULES

$E \rightarrow E_1$ **or** $M\ E_2$	backpatch(E_1.flist, M.quad);
	E.tlist = merge(E_1.tlist, E_2.tlist);
	E.flist = E_2.flist;

$E \rightarrow E_1$ **and** $M\ E_2$	{ backpatch(E1.tlist, M.quad);
	E.tlist = E2.tlist;
	E.flist = merge(E1.flist, E2.flist); }

$E \rightarrow$ **not** E_1	E.tlist = E1.flist; E.flist = E1.tlist;
$E \rightarrow (E_1)$	E.tlist = E1.tlist; E.flist = E1.flist;

E —> id1 relop id2	E.tlist = makelist(next);
	E.flist = makelist(next+1);
	gen("if id1.place relop.op id2.place goto -");
	gen("goto -");

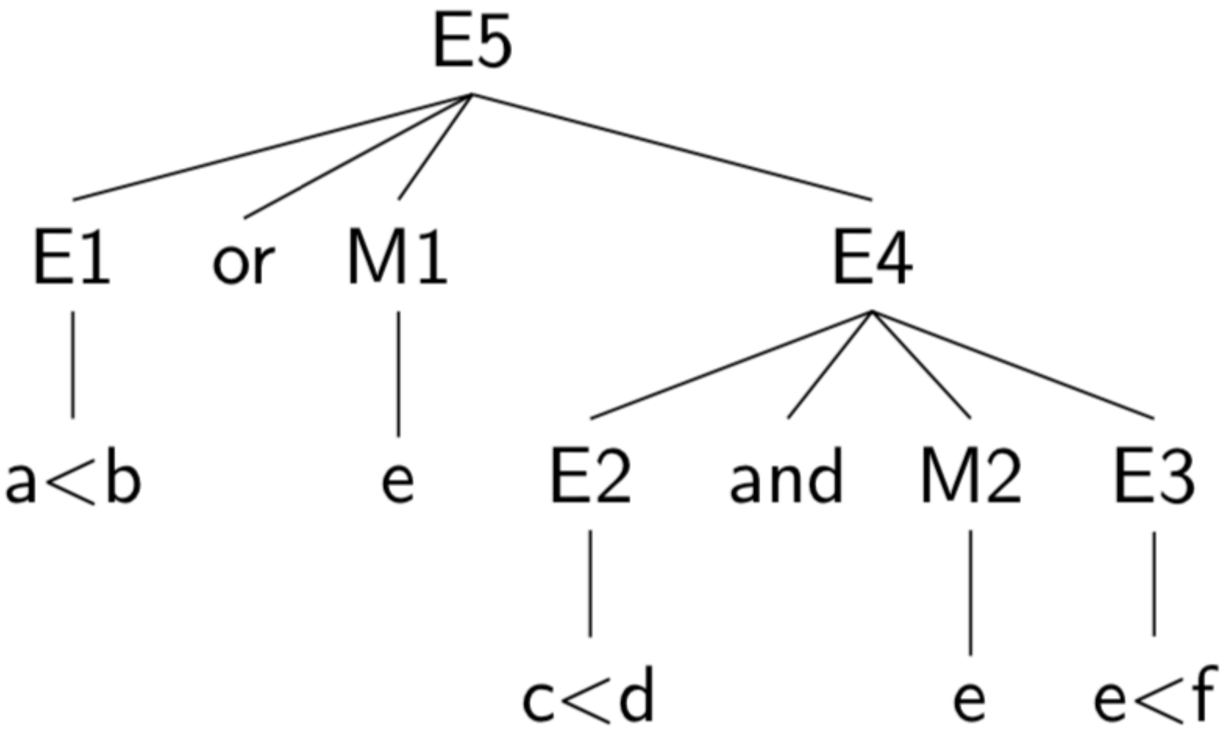
E —> **true**	E.tlist = makelist(next); gen("goto -");
E —> **false**	E.flist = makelist(next); gen("goto -");
$M \rightarrow \varepsilon$	M.quad = next;

Backpatching for Boolean Expressions

- Example:
  - a < b or c < d and e < f

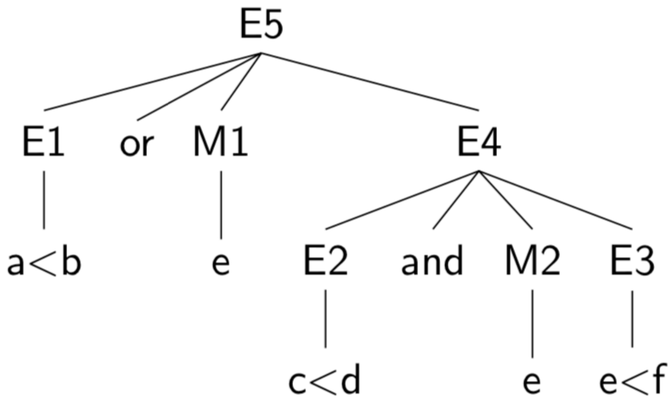
Backpatching for Boolean Expressions

- Example:
  - a < b or c < d and e < f



Backpatching for Boolean Expressions

- Example:
  - a < b or c < d and e < f

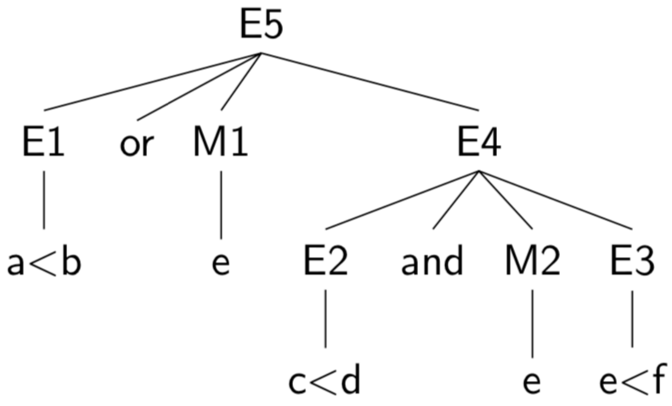


E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

```
100: if a < b goto _
101: goto _
```

Backpatching for Boolean Expressions

- Example:
  - a < b or c < d and e < f



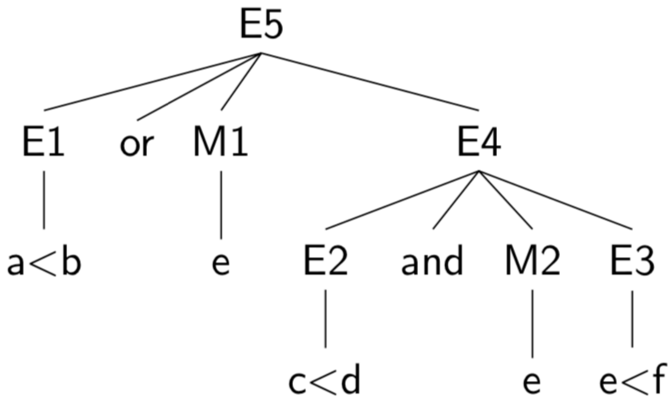
E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

M1 = nextquad = 102

100: if a < b goto _

101: goto _

Backpatching for Boolean Expressions



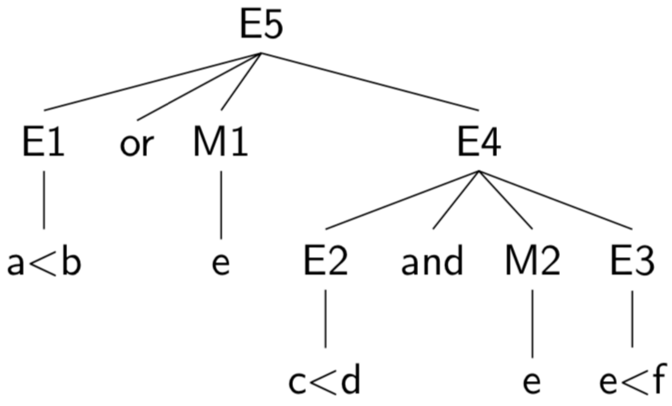
- Example:
  - a < b or c < d and e < f

E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

M1 = nextquad = 102
E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

```
100: if a < b goto _
101: goto _
102: if c < d goto _
103: goto _
```

Backpatching for Boolean Expressions



- Example:
  - a < b or c < d and e < f

E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

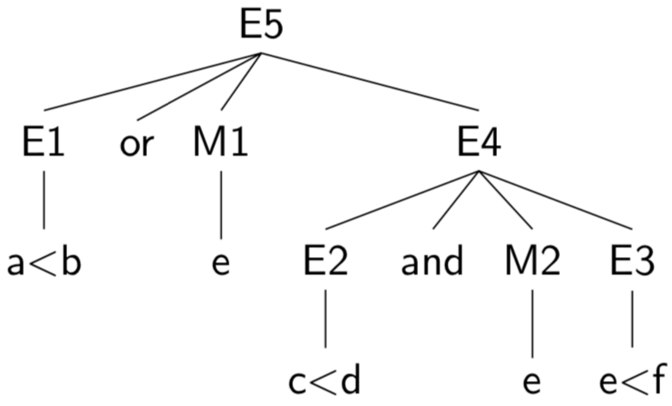
M1 = nextquad = 102
E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

M2 = nextquad = 104

```
100: if a < b goto _
101: goto _
102: if c < d goto _
103: goto _
```

Backpatching for Boolean Expressions

- Example:
  - a < b or c < d and e < f



```
100: if a < b goto _
101: goto _
102: if c < d goto _
103: goto _
104: if e < f goto _
105: goto _
```

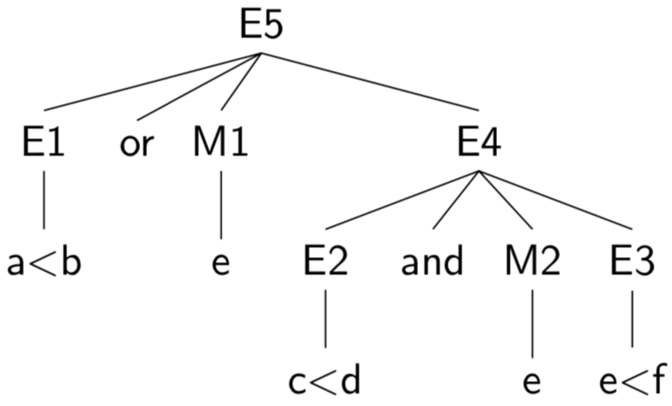
E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

M1 = nextquad = 102
E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

M2 = nextquad = 104
E3.tlist = makelist(next)=104
E3.flist = makelist(next+1)=105

Backpatching for Boolean Expressions

- Example:
  - a < b or c < d and e < f



```
100: if a < b goto _
101: goto _
102: if c < d goto 104
103: goto _
104: if e < f goto _
105: goto _
```

E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

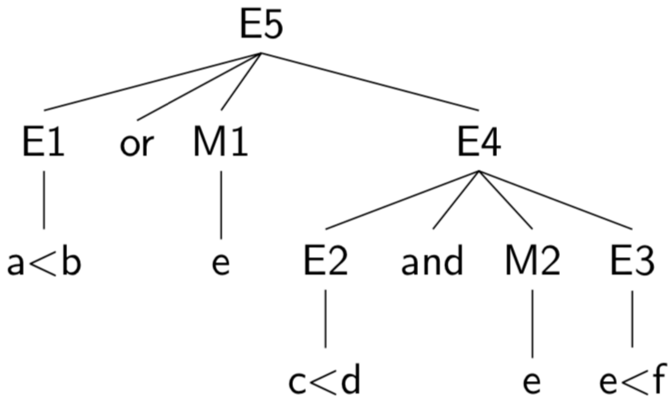
M1 = nextquad = 102
E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

M2 = nextquad = 104
E3.tlist = makelist(next)=104
E3.flist = makelist(next+1)=105

BP(E2.tlist, M2.quad)=({102},104)
E4.tlist = E3.tlist = 104
E4.flist = merge(E2.flist, E3.flist)=103,105

Backpatching for Boolean Expressions

- Example:
  - a < b or c < d and e < f



```
100: if a < b goto _
101: goto 102
102: if c < d goto 104
103: goto _
104: if e < f goto _
105: goto _
```

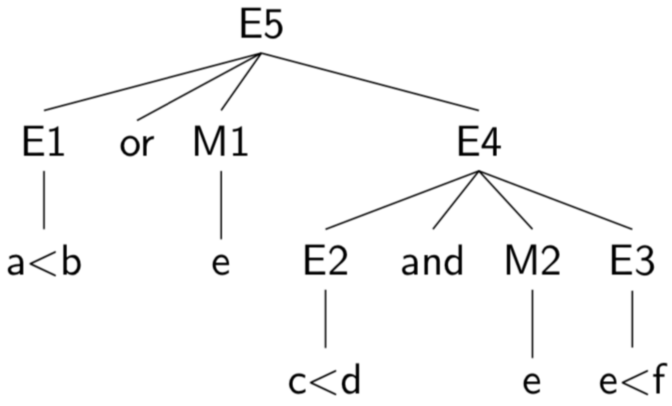
E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

M1 = nextquad = 102
E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

M2 = nextquad = 104
E3.tlist = makelist(next)=104
E3.flist = makelist(next+1)=105

BP(E2.tlist, M2.quad)=({102},104)
E4.tlist = E3.tlist = 104
E4.flist = merge(E2.flist, E3.flist)=103,105
BP(E1.flist, M1.quad)=({101},102)
E5.tlist = merge(E1.tlist, E4.tlist)=100,104
E5.flist = E4.flist=103,105

Backpatching for Boolean Expressions



- Example:
  - a < b or c < d and e < f

```
100: if a < b goto _ +
101: goto 102
102: if c < d goto 104
103: goto _ -
104: if e < f goto _ +
105: goto _ -
```

E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

M1 = nextquad = 102
E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

M2 = nextquad = 104
E3.tlist = makelist(next)=104
E3.flist = makelist(next+1)=105

BP(E2.tlist, M2.quad)=({102},104)
E4.tlist = E3.tlist = 104
E4.flist = merge(E2.flist, E3.flist)=103,105
BP(E1.flist, M1.quad)=({101},102)
E5.tlist = merge(E1.tlist, E4.tlist)=100,104
E5.flist = E4.flist=103,105

Code Generation for Control Flow Statements

PRODUCTION	SEMANTIC RULES

S → if E then S1	E.true := newlabel ;
	E.false := S.next ;
	S1.next := S.next ;
	S.code := E.code || gen(E.true ‘:’) || S1.code

S → if E then S1 else S2	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 S1	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)

Backpatching for Control Flow Statements

PRODUCTION	SEMANTIC RULES

S → if E then M S1	backpatch(E.tlist, M.quad);
	S.nlist = merge(E.flist,S1.nlist);

S → if E then M1 S1 N else M2 S2	backpatch(E.tlist, M1.quad);
	backpatch(E.flist, M2.quad);
	S.nlist = merge(S1.nlist,S2.nlist, N.nlist);

S → while M1 E do M2 S1	backpatch(S1.nlist, M1.quad);
	backpatch(E.tlist, M2.quad);
	S.nlist = E.flist; gen("goto M1.quad");

M —> ε	M.quad = next;

N —> ε	N.nlist = makelist(next);
	gen("goto -");

Code Generation for Control Flow Statements

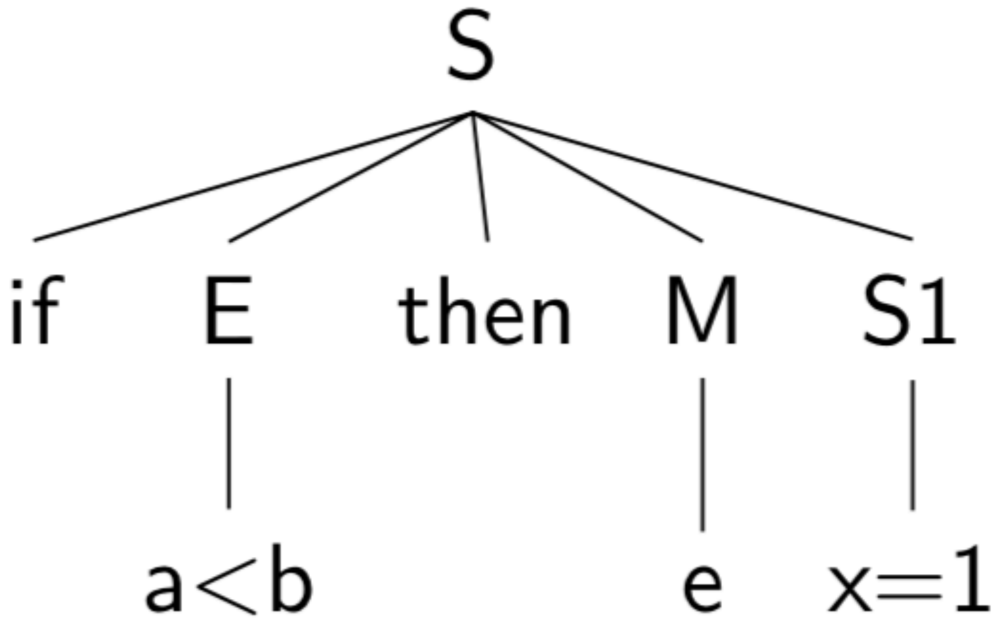
- Example:

```
if ( a < b )
    x = 1;
```

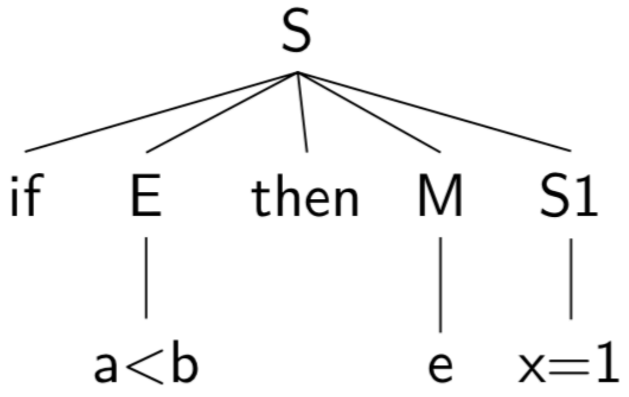
Code Generation for Control Flow Statements

- Example:

```
if ( a < b )
    x = 1;
```



Code Generation for Control Flow Statements



- Example:

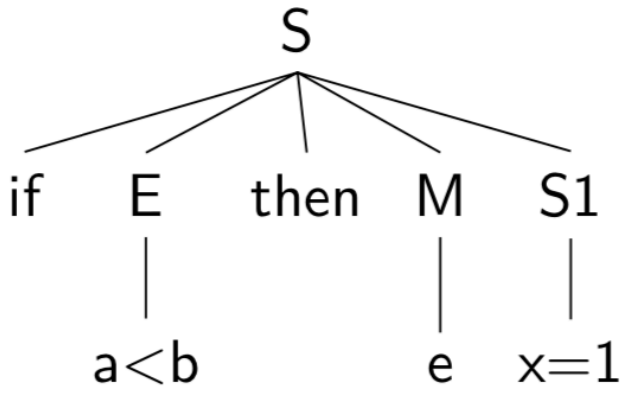
  if (a < b)

  x = 1;

E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101

```
100: if a < b goto _
101: goto _
```

Code Generation for Control Flow Statements



- Example:

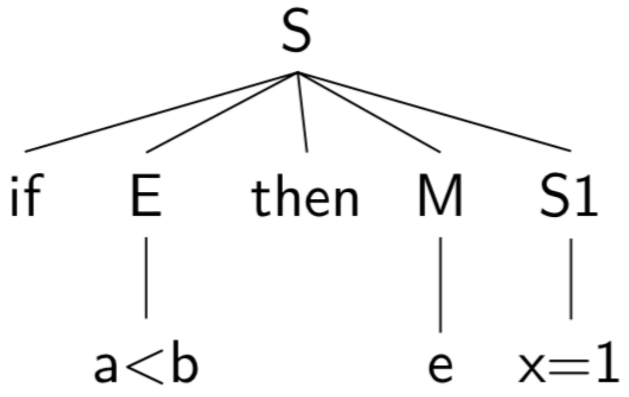
```
if ( a < b )
    x = 1;
```

E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101

M = nextquad = 102

```
100: if a < b goto _
101: goto _
```

Code Generation for Control Flow Statements



- Example:

```
if ( a < b )
    x = 1;
```

E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101

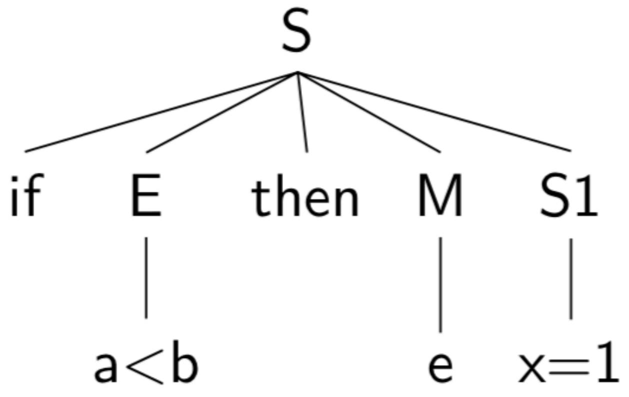
M = nextquad = 102

```
100: if a < b goto _
101: goto _
102: x=1
```

Code Generation for Control Flow Statements

- Example:

```
if ( a < b )
    x = 1;
```



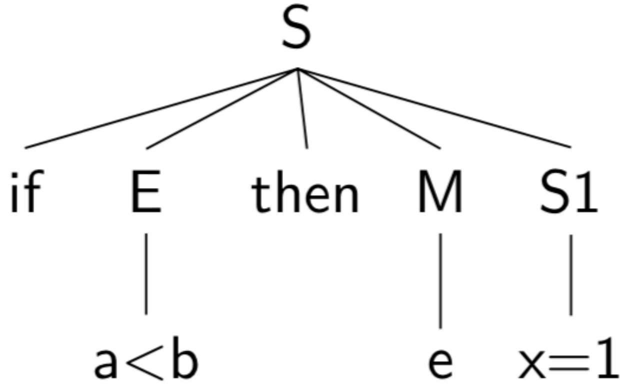
E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101

M = nextquad = 102

```
100: if a < b goto _
101: goto _
102: x=1
```

BP(E.tlist, M.quad)=({100},102)
S.nlist = merge(E.flist,S1.nlist)=101,S1next

Code Generation for Control Flow Statements



- Example:

```
if ( a < b )
    x = 1;
```

E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101

M = nextquad = 102

```
100: if a < b goto 102
101: goto _
102: x=1
```

BP(E.tlist, M.quad)=({100},102)
S.nlist = merge(E.flist,S1.nlist)=101,S1next

Code Generation for Control Flow Statements

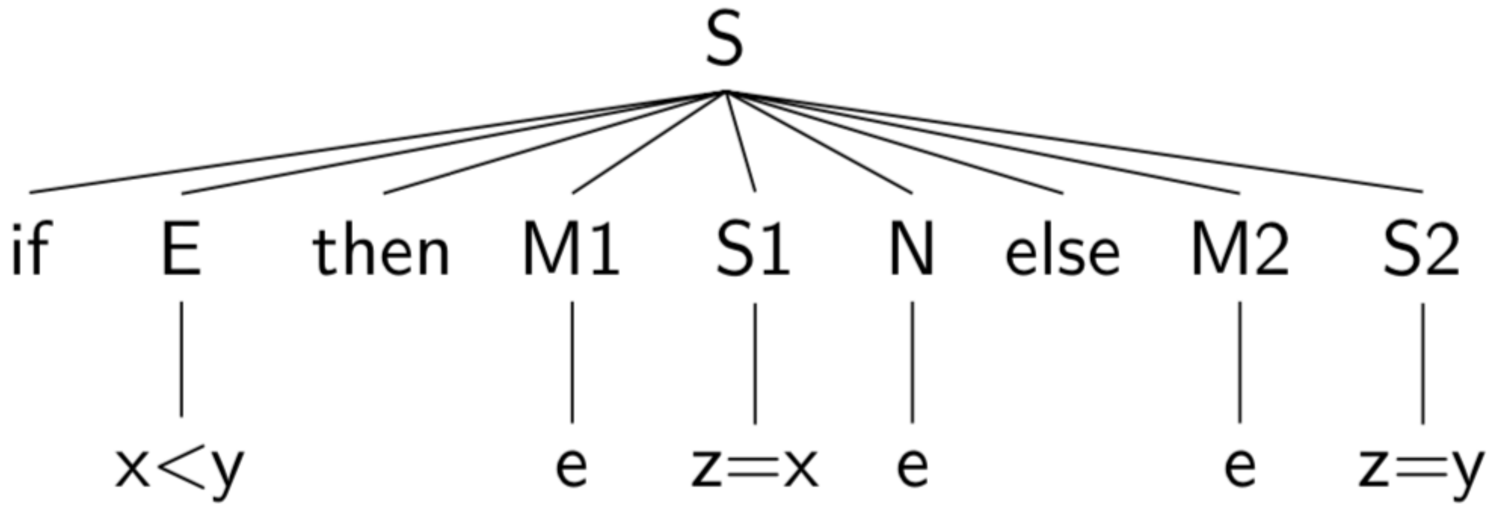
- Example:

```
if ( x < y )
        z = x;
else
        z = y;
```

Code Generation for Control Flow Statements

- Example:

```
if ( x < y )
        z = x;
else
        z = y;
```



Code Generation for Control Flow Statements

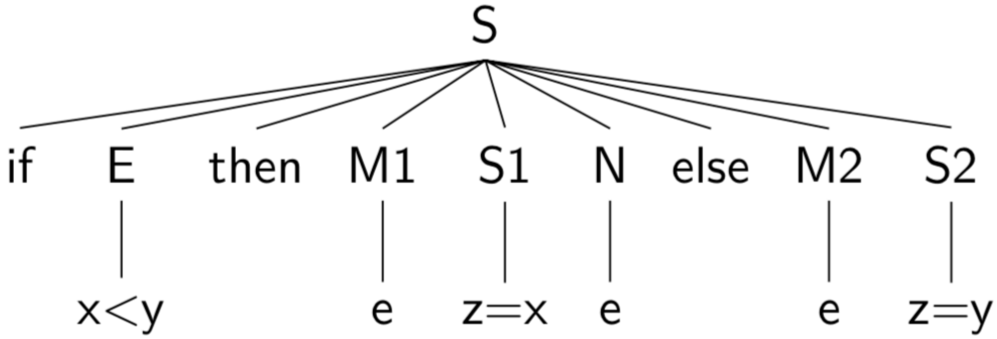
- Example:

  if (x < y)

  z = x;

  else

  z = y;



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101

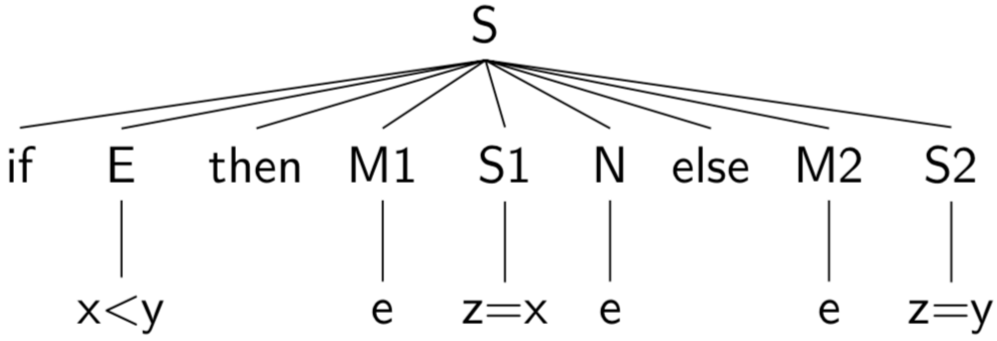
100: if x < y goto _

101: goto _

Code Generation for Control Flow Statements

- Example:

```
if ( x < y )
        z = x;
else
        z = y;
```



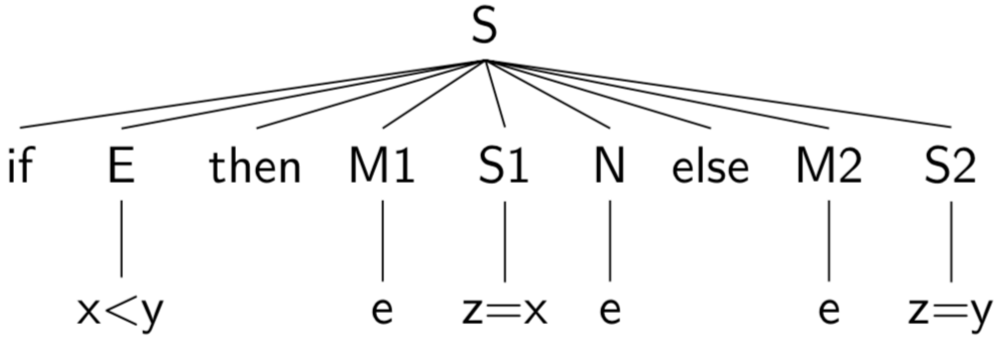
E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

```
100: if x < y goto _
101: goto _
```

Code Generation for Control Flow Statements

- Example:

```
if ( x < y )
        z = x;
else
        z = y;
```



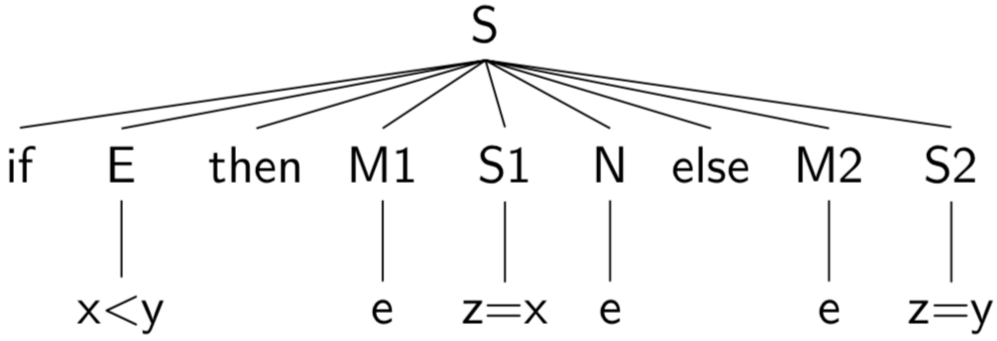
E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

```
100: if x < y goto _
101: goto _
102: z=x
```

Code Generation for Control Flow Statements

- Example:

```
if ( x < y )
        z = x;
else
        z = y;
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

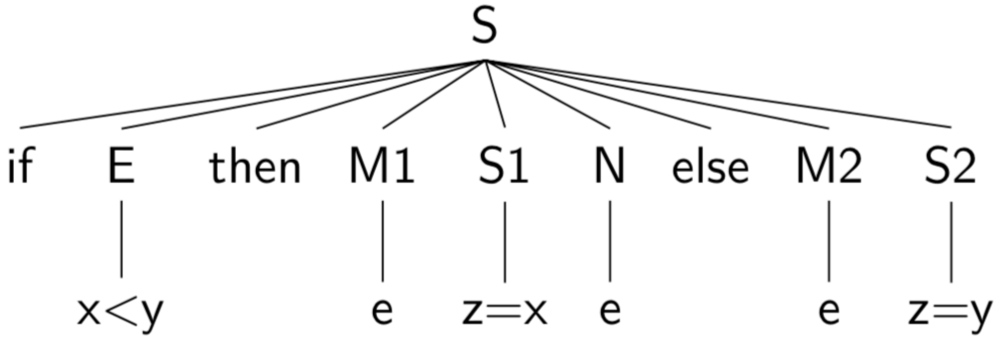
N.Nlist = makelist(next) = 103

```
100: if x < y goto _
101: goto _
102: z=x
103: goto _
```

Code Generation for Control Flow Statements

- Example:

```
if ( x < y )
        z = x;
else
        z = y;
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

N.Nlist = makelist(next) = 103

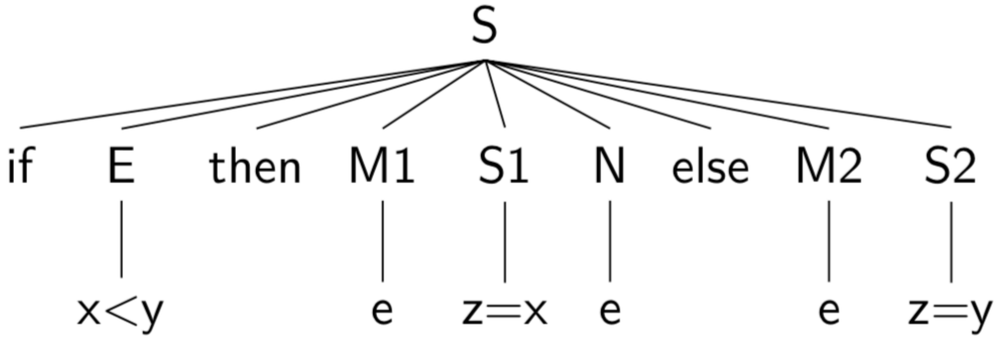
M2 = nextquad = 104

```
100: if x < y goto _
101: goto _
102: z=x
103: goto _
```

Code Generation for Control Flow Statements

- Example:

```
if ( x < y )
        z = x;
else
        z = y;
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

N.Nlist = makelist(next) = 103

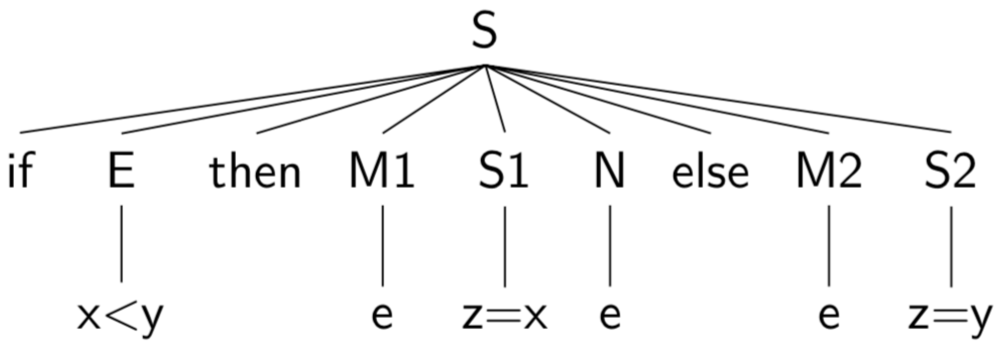
M2 = nextquad = 104

```
100: if x < y goto _
101: goto _
102: z=x
103: goto _
104: z = y
```

Code Generation for Control Flow Statements

- Example:

```
if ( x < y )
      z = x;
else
      z = y;
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

N.Nlist = makelist(next) = 103

M2 = nextquad = 104

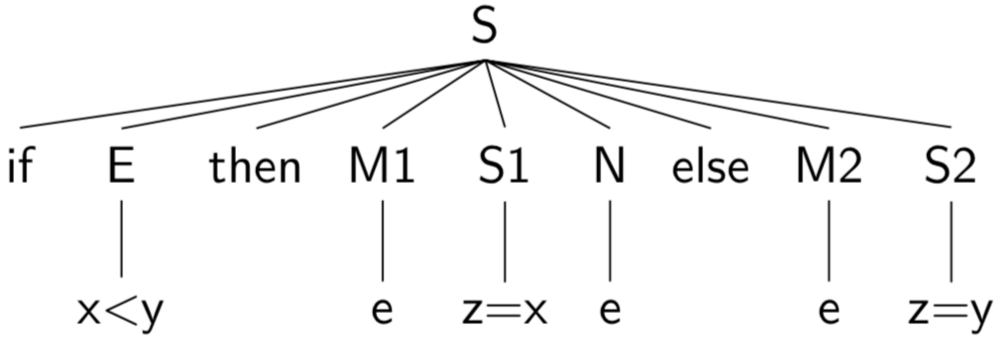
BP(E.tlist, M1.quad) = ({100},102)
BP(E.flist, M2.quad) = ({101},104)
S.nlist = merge(S1.nlist,S2.nlist, N.nlist)=
(S1.nlist+S2.nlist+103)

```
100: if x < y goto _
101: goto _
102: z=x
103: goto _
104: z = y
```

Code Generation for Control Flow Statements

- Example:

```
if ( x < y )
        z = x;
else
        z = y;
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

N.Nlist = makelist(next) = 103

M2 = nextquad = 104

BP(E.tlist, M1.quad) = ({100},102)
BP(E.flist, M2.quad) = ({101},104)
S.nlist = merge(S1.nlist,S2.nlist, N.nlist)=
(S1.nlist+S2.nlist+103)

```
100: if x < y goto 102
101: goto 104
102: z=x
103: goto _
104: z = y
```

Code Generation for Control Flow Statements

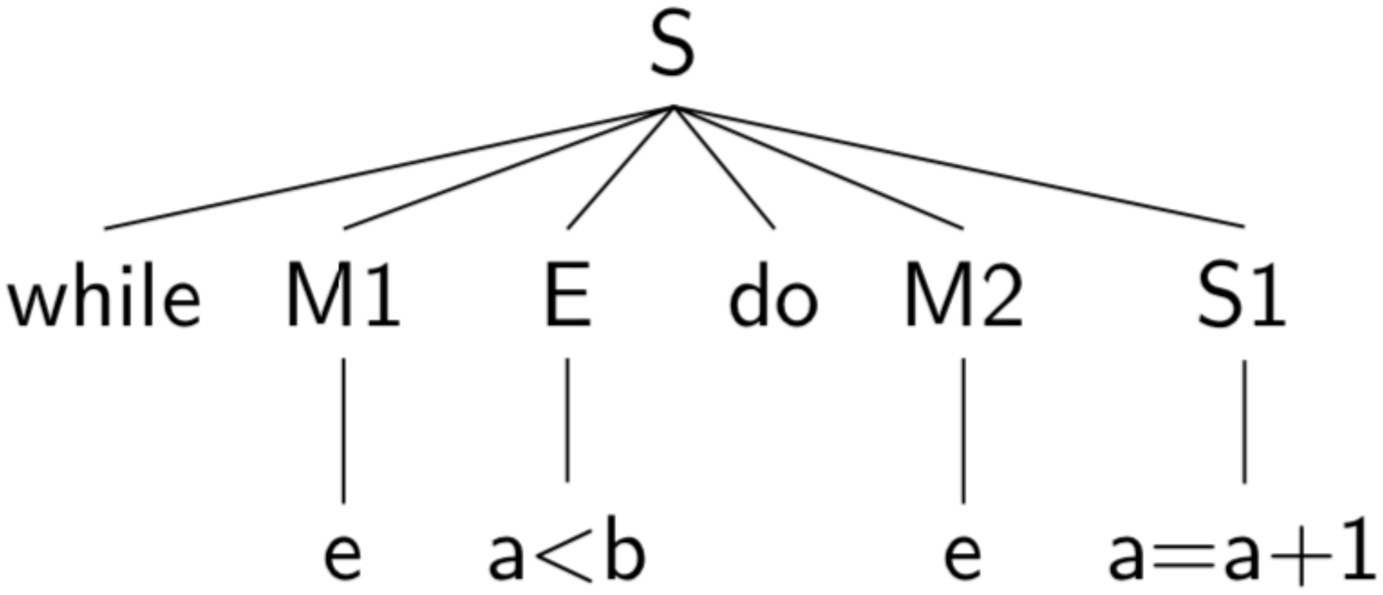
- Example:

```
while ( a < b )
    a = a+1;
```

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
        a = a+1;
```

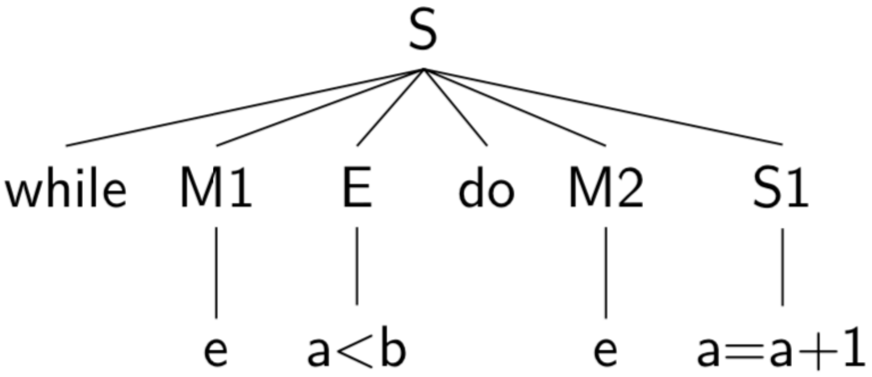


Code Generation for Control Flow Statements

- Example:

  while (a < b)

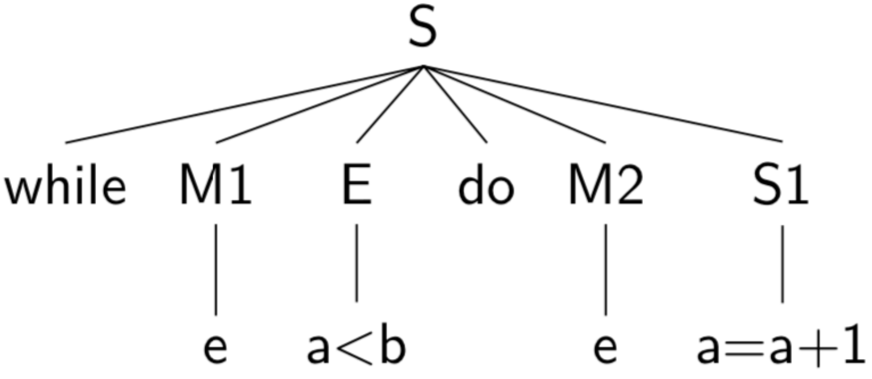
  a = a+1;



Code Generation for Control Flow Statements

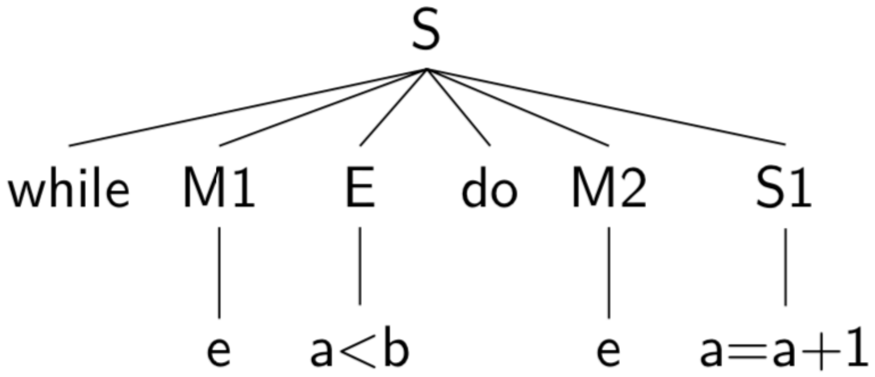
- Example:

```
while ( a < b )
    a = a+1;
```



M1 = nextquad = 100

Code Generation for Control Flow Statements



- Example:

  while (a < b)

  a = a+1;

M1 = nextquad = 100

100: if a < b goto _

101: goto _

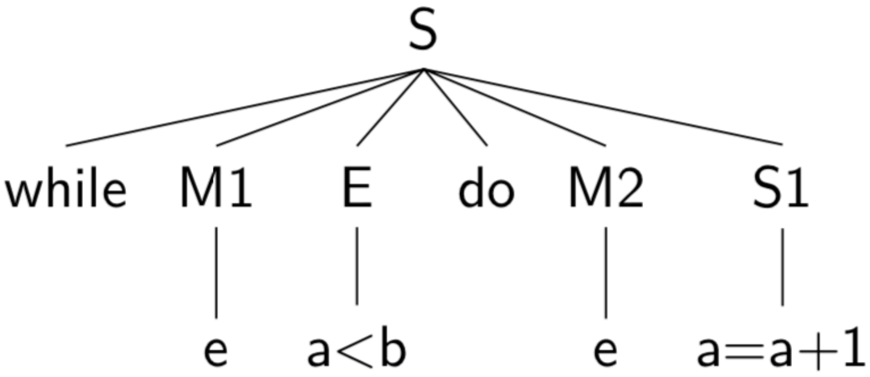
E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101

Code Generation for Control Flow Statements

- Example:

  while (a < b)

  a = a+1;



M1 = nextquad = 100

100: if a < b goto _

101: goto _

E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101

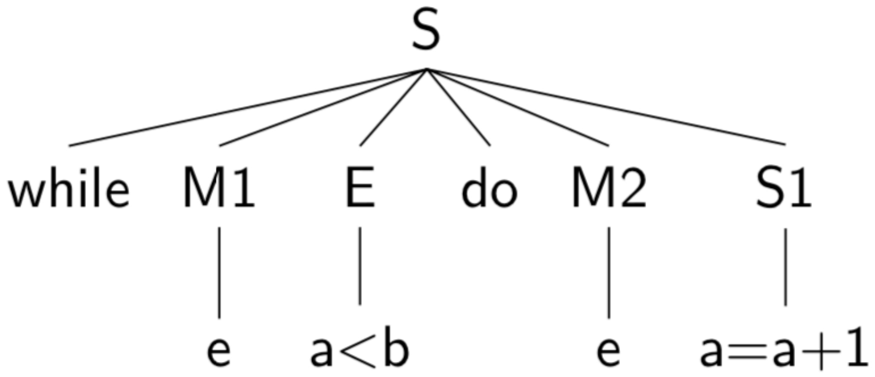
M2 = nextquad = 102

Code Generation for Control Flow Statements

- Example:

  while (a < b)

  a = a+1;



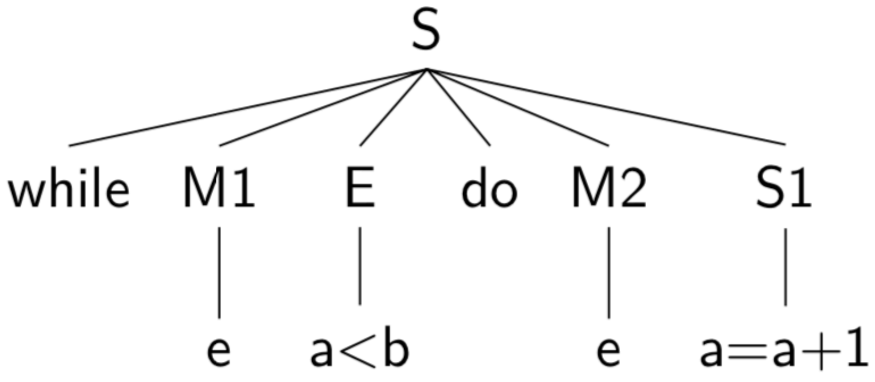
M1 = nextquad = 100

E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101

M2 = nextquad = 102

```
100: if a < b goto _
101: goto _
102: t1 = a+1
103: a = t1
```

Code Generation for Control Flow Statements



- Example:

```
while ( a < b )
    a = a+1;
```

M1 = nextquad = 100

E.tlist = makelist(next)=100

E.flist = makelist(next+1)=101

M2 = nextquad = 102

BP(S1.nlist, M1.quad)=({null},100)

BP(E.tlist, M2.quad)=({100},102)

S.nlist = E.flist = 101

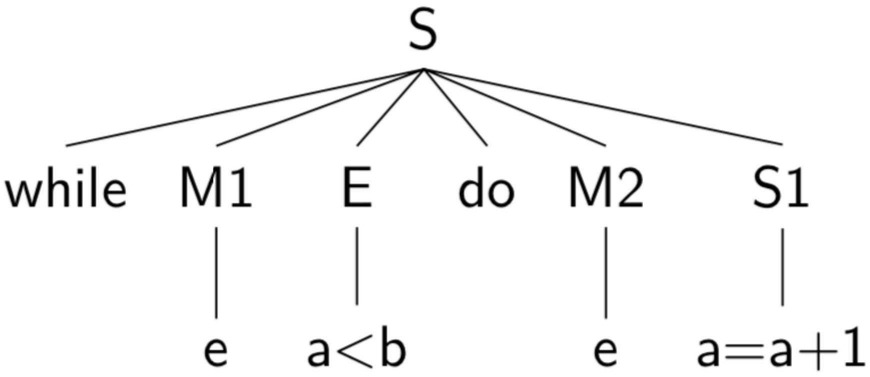
```
100: if a < b goto _
101: goto _
102: t1 = a+1
103: a = t1
104: goto  100
```

Code Generation for Control Flow Statements

- Example:

  while (a < b)

  a = a+1;



M1 = nextquad = 100

100: if a < b goto 102

101: goto _

102: t1 = a+1

103: a = t1

104: goto 100

E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101

M2 = nextquad = 102

BP(S1.nlist, M1.quad)=({null},100)
BP(E.tlist, M2.quad)=({100},102)
S.nlist = E.flist = 101

Code Generation for Control Flow Statements

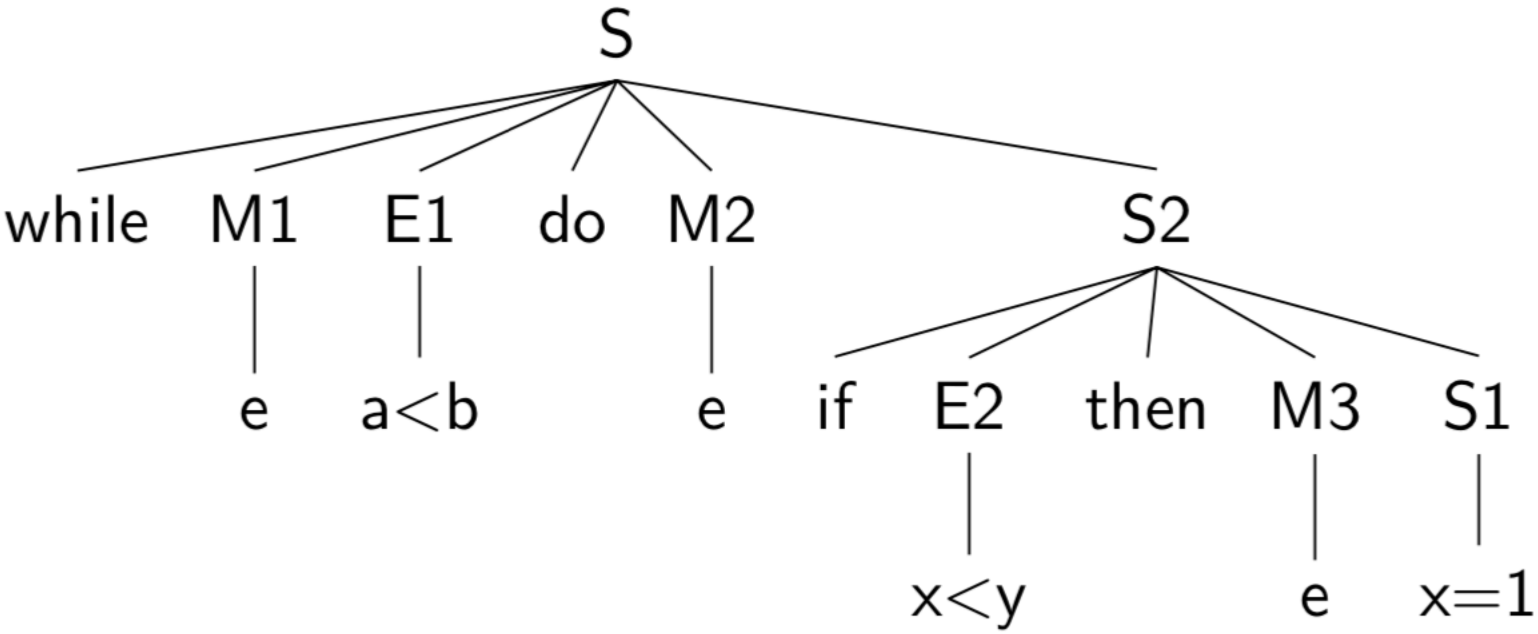
- Example:

```
while ( a < b )
        if (x < y)
            x = 1;
```

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
      if (x < y)
          x = 1;
```



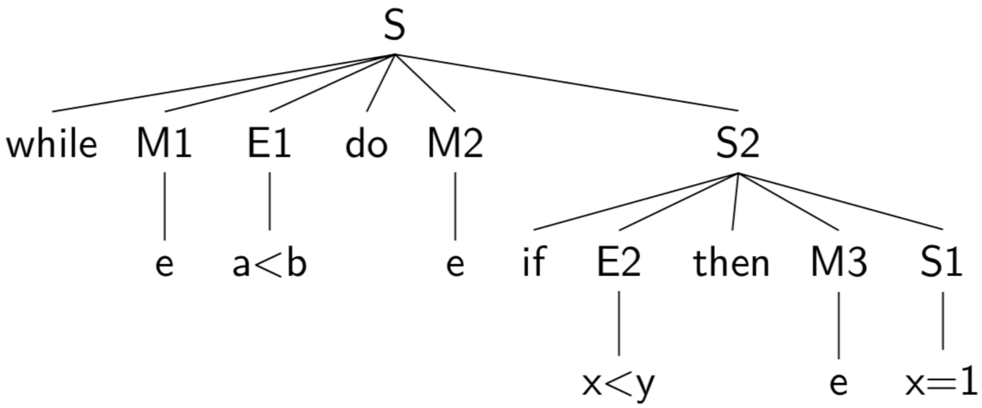
Code Generation for Control Flow Statements

- Example:

  while (a < b)

  if (x < y)

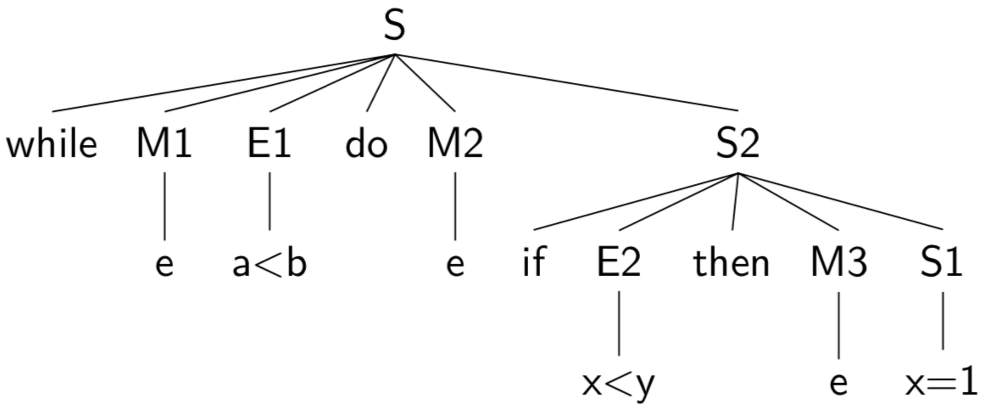
  x = 1;



Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
        if (x < y)
            x = 1;
```



M1 = nextquad = 100

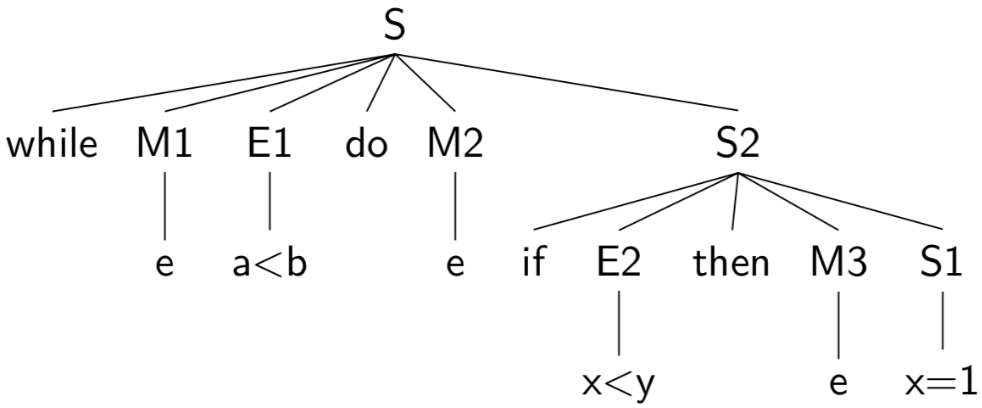
Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
        if (x < y)
            x = 1;
```

100: if a < b goto _

101: goto _

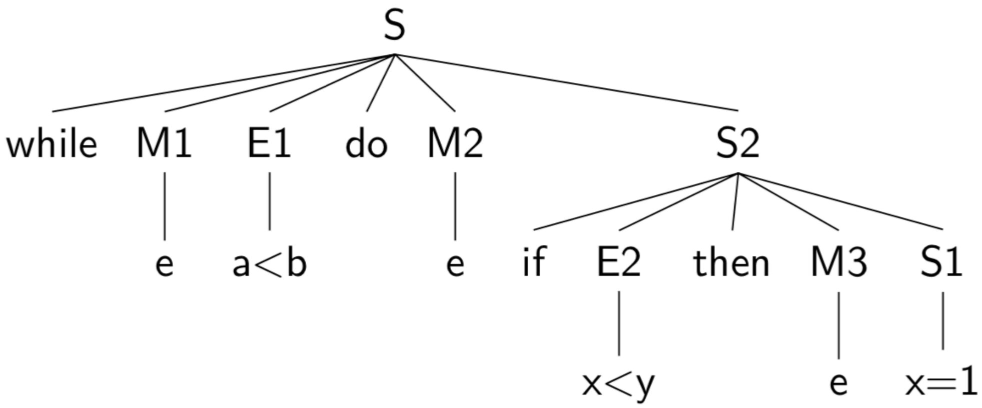


M1 = nextquad = 100

E1.tlist = makelist(next)=100

E1.flist = makelist(next+1)=101

Code Generation for Control Flow Statements



- Example:

```
while ( a < b )
        if (x < y)
            x = 1;
```

$M1 = nextquad = 100$

E1.tlist = makelist(next)=100

E1.flist = makelist(next+1)=101

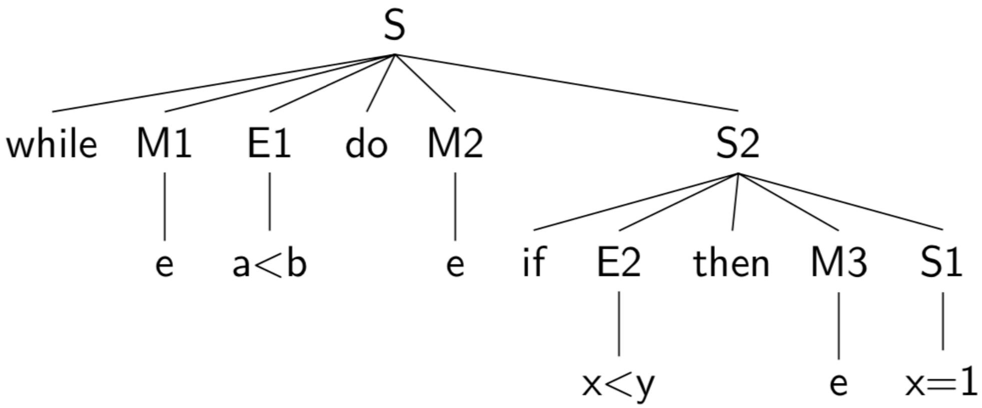
$M2 = nextquad = 102$

```
100: if a < b goto _
101: goto _
```

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
        if (x < y)
            x = 1;
```



M1 = nextquad = 100

E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

M2 = nextquad = 102

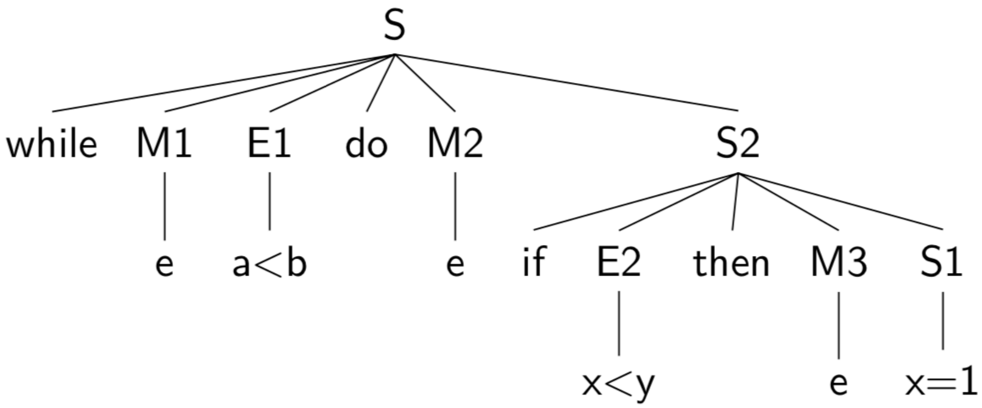
E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

```
100: if a < b goto _
101: goto _
102: if x < y goto _
103: goto _
```

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
        if (x < y)
            x = 1;
```



M1 = nextquad = 100

E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

M2 = nextquad = 102

E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

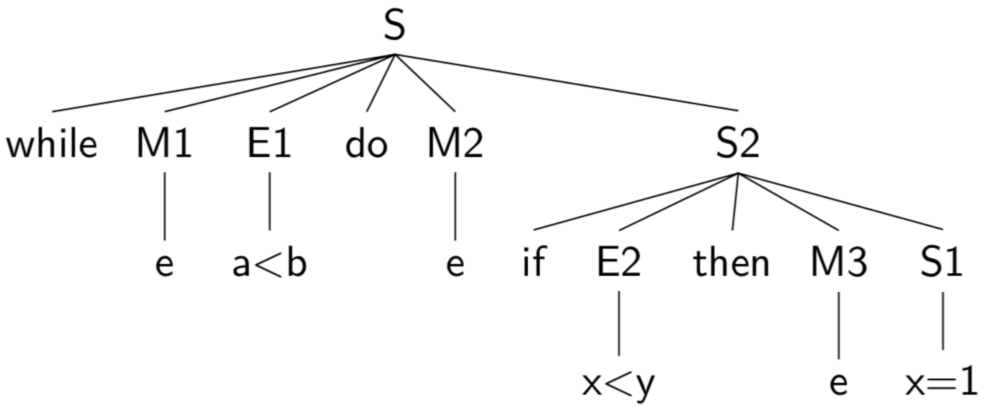
M3 = nextquad = 104

```
100: if a < b goto _
101: goto _
102: if x < y goto _
103: goto _
```

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
        if (x < y)
            x = 1;
```



M1 = nextquad = 100

E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

M2 = nextquad = 102

E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

M3 = nextquad = 104

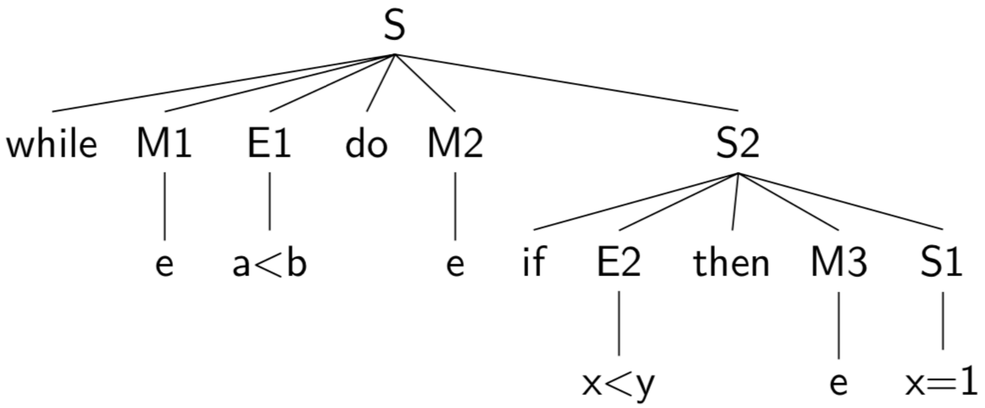
```
100: if a < b goto _
101: goto _
102: if x < y goto _
103: goto _
104: x = 1
```

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
        if (x < y)
            x = 1;
```

```
100: if a < b goto _
101: goto _
102: if x < y goto _
103: goto _
104: x = 1
```



M1 = nextquad = 100

E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

M2 = nextquad = 102

E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

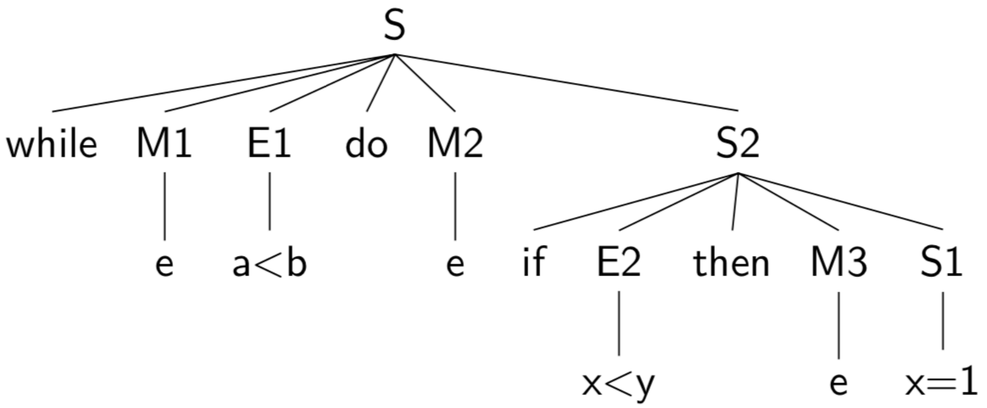
M3 = nextquad = 104

BP(E2.tlist, M3.quad)=({102},104)
S2.nlist = merge(E2.flist,S1.nlist)=103+s1.nlist

Code Generation for Control Flow Statements

- Example:

```
while ( a < b )
      if (x < y)
          x = 1;
```



```
100: if a < b goto _
101: goto _
102: if x < y goto 104
103: goto _
104: x = 1
```

M1 = nextquad = 100

E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

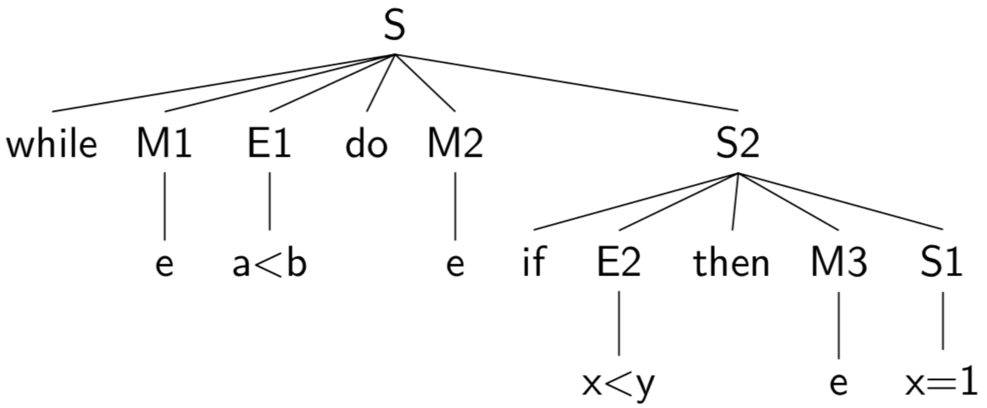
M2 = nextquad = 102

E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

M3 = nextquad = 104

BP(E2.tlist, M3.quad)=({102},104)
S2.nlist = merge(E2.flist,S1.nlist)=103+s1.nlist

Code Generation for Control Flow Statements



- Example:

```
while ( a < b )
          if (x < y)
               x = 1;
```

100: if a < b goto _

101: goto _

102: if x < y goto 104

103: goto _

104: x = 1

105: goto 100

M1 = nextquad = 100

E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

M2 = nextquad = 102

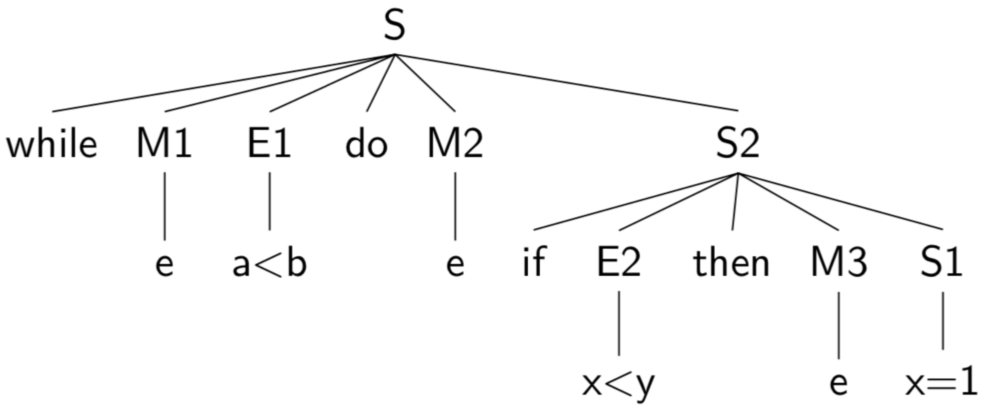
E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

M3 = nextquad = 104

BP(E2.tlist, M3.quad)=({102},104)
S2.nlist = merge(E2.flist,S1.nlist)=103+s1.nlist

BP(S2.nlist, M1.quad)=({103},100)
BP(E1.tlist, M2.quad)=({100},102)
S.nlist = E1.flist = 101

Code Generation for Control Flow Statements



- Example:

```
while ( a < b )
        if (x < y)
            x = 1;
```

```
100: if a < b goto 102
101: goto _
102: if x < y goto 104
103: goto 100
104: x = 1
105: goto 100
```

M1 = nextquad = 100

E1.tlist = makelist(next)=100
E1.flist = makelist(next+1)=101

M2 = nextquad = 102

E2.tlist = makelist(next)=102
E2.flist = makelist(next+1)=103

M3 = nextquad = 104

BP(E2.tlist, M3.quad)=({102},104)
S2.nlist = merge(E2.flist,S1.nlist)=103+s1.nlist

BP(S2.nlist, M1.quad)=({103},100)
BP(E1.tlist, M2.quad)=({100},102)
S.nlist = E1.flist = 101

Backpatching for Control Flow Statements

PRODUCTION	SEMANTIC RULES
S → begin L end	S.nlist = L.nlist;
S → A	S.nlist = NULL;

L → L1 ; M S	backpatch(L1.nlist, M.quad);
	L.nlist = S.nlist;

L → S	L.nlist = S.nlist;

Code Generation for Control Flow Statements

- Example:

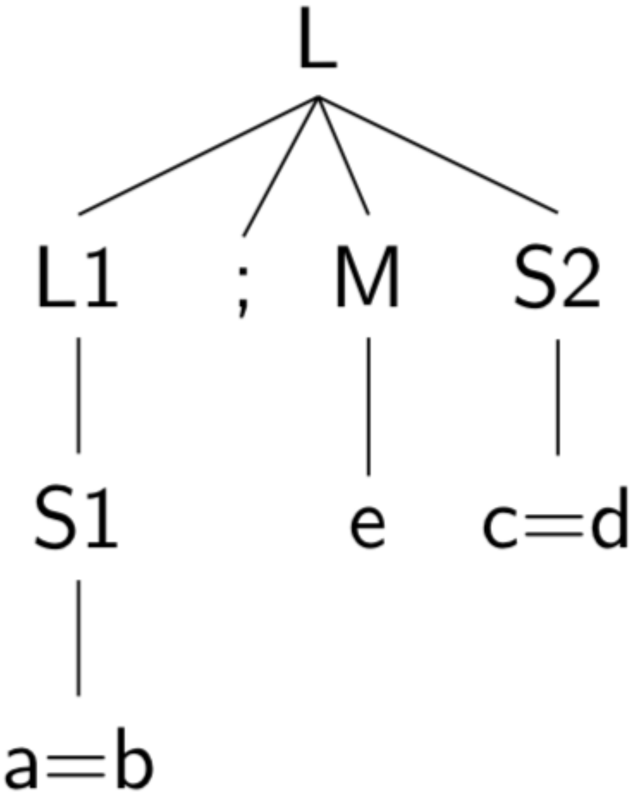
```
a = b;
c = d;
```

Code Generation for Control Flow Statements

- Example:

  a = b;

  c = d;

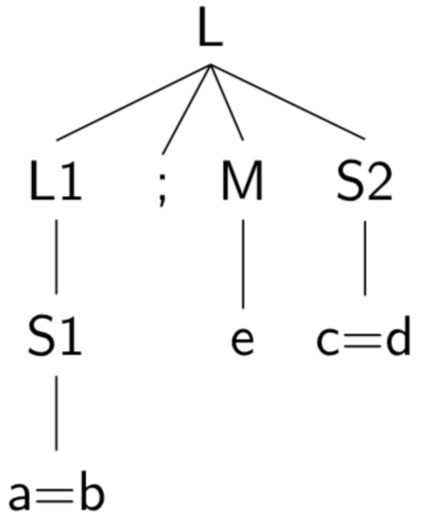


Code Generation for Control Flow Statements

- Example:

  a = b;

  c = d;



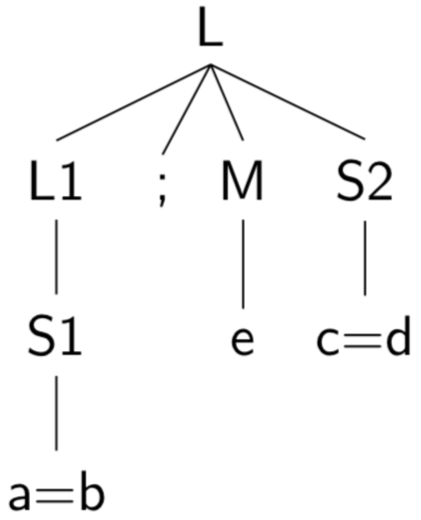
Code Generation for Control Flow Statements

- Example:

  a = b;

  c = d;

  100: a = b



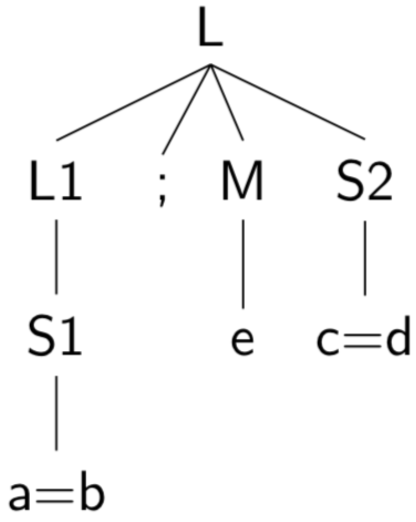
Code Generation for Control Flow Statements

- Example:

  a = b;

  c = d;

  100: a = b



L1.nlist = S1.nlist

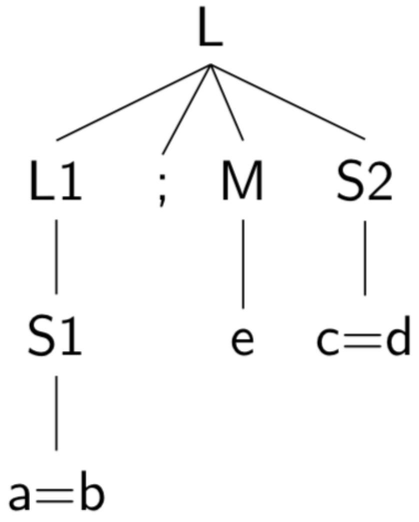
Code Generation for Control Flow Statements

- Example:

  a = b;

  c = d;

  100: a = b



L1.nlist = S1.nlist

M = nextquad = 101

Code Generation for Control Flow Statements

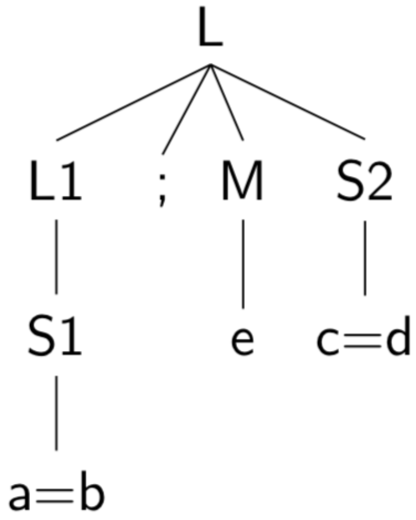
- Example:

  a = b;

  c = d;

  100: a = b

  101: c = d



L1.nlist = S1.nlist

M = nextquad = 101

Code Generation for Control Flow Statements

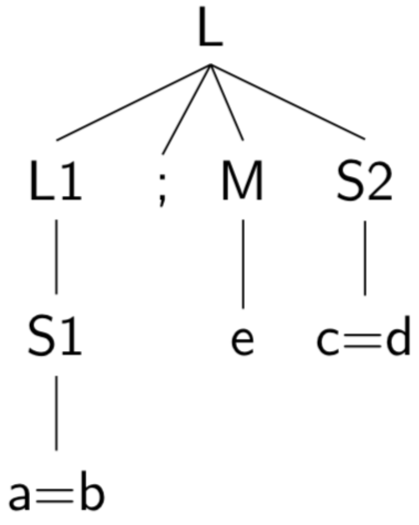
- Example:

  a = b;

  c = d;

  100: a = b

  101: c = d



L1.nlist = S1.nlist

M = nextquad = 101

BP(L1.nlist, M.quad) = ({null}, 101)
L.nlist = S2.nlist

Code Generation for Control Flow Statements

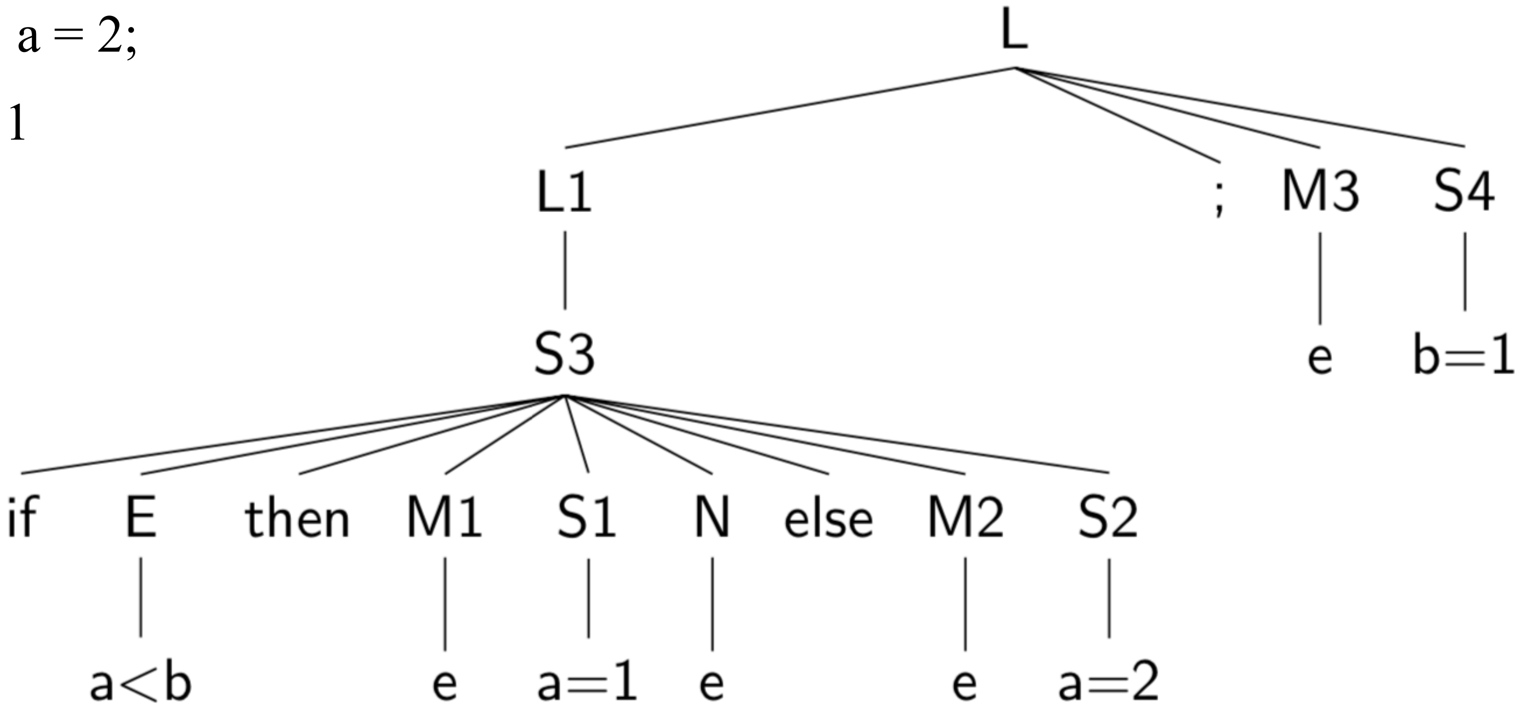
- Example:

```
if ( a < b )
        a = 1;
else
        a = 2;
b = 1
```

Code Generation for Control Flow Statements

- Example:

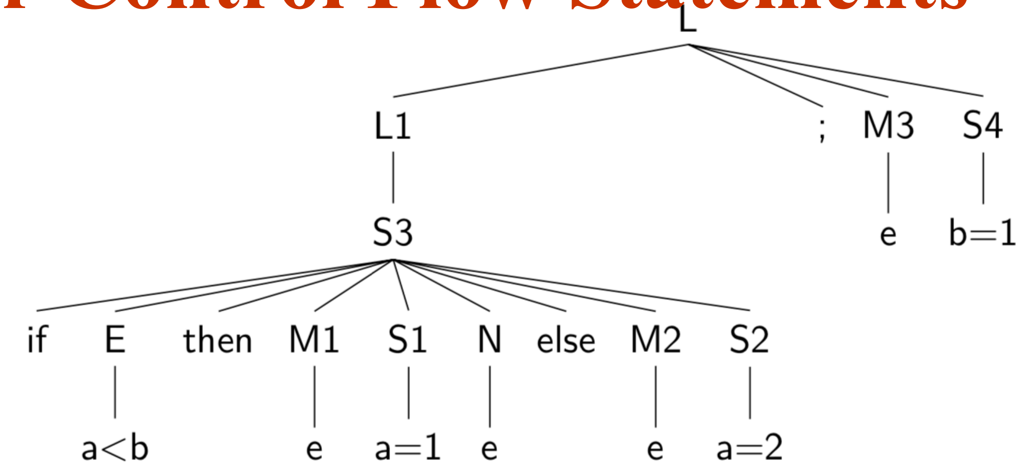
```
if ( a < b )
        a = 1;
else
        a = 2;
b = 1
```



Code Generation for Control Flow Statements

- Example:

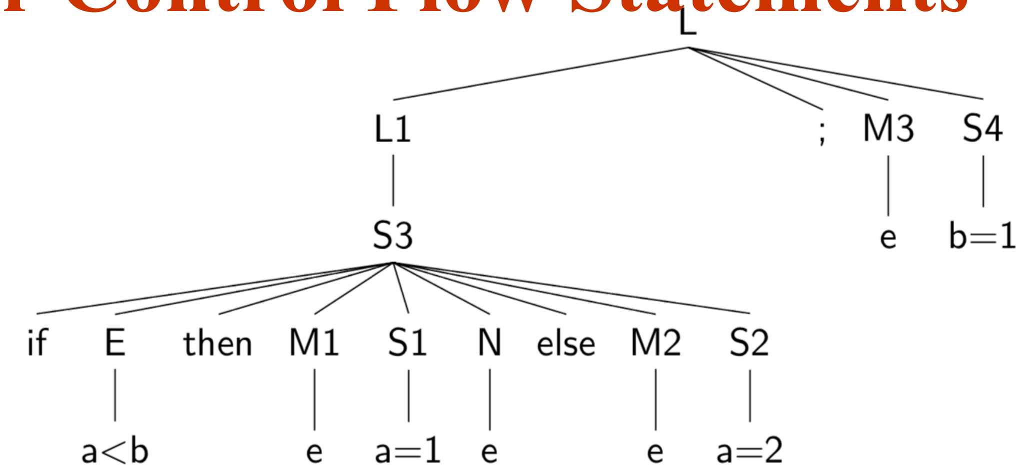
```
if ( a < b )
        a = 1;
else
        a = 2;
b = 1
```



Code Generation for Control Flow Statements

- Example:

```
if ( a < b )
        a = 1;
else
        a = 2;
b = 1
```



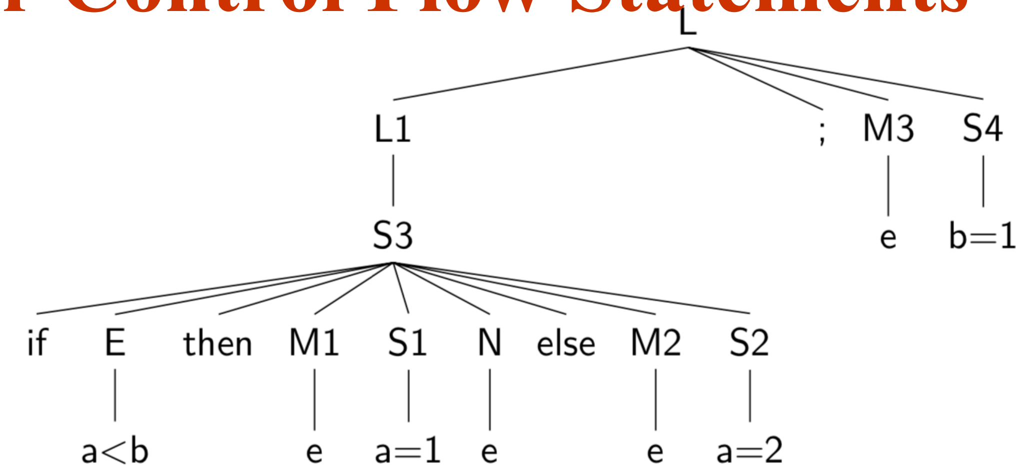
E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101

```
100: if a < b goto _
101: goto _
```

Code Generation for Control Flow Statements

- Example:

```
if ( a < b )
        a = 1;
else
        a = 2;
b = 1
```



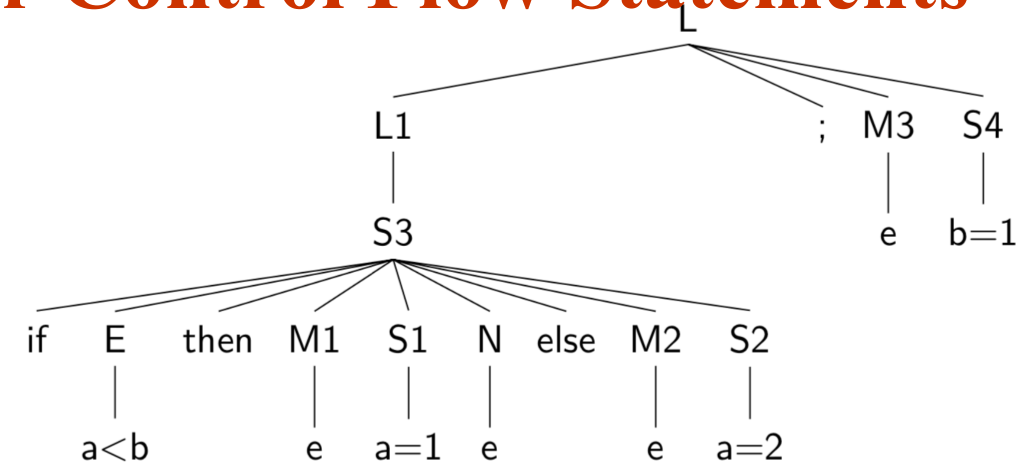
E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

```
100: if a < b goto _
101: goto _
```

Code Generation for Control Flow Statements

- Example:

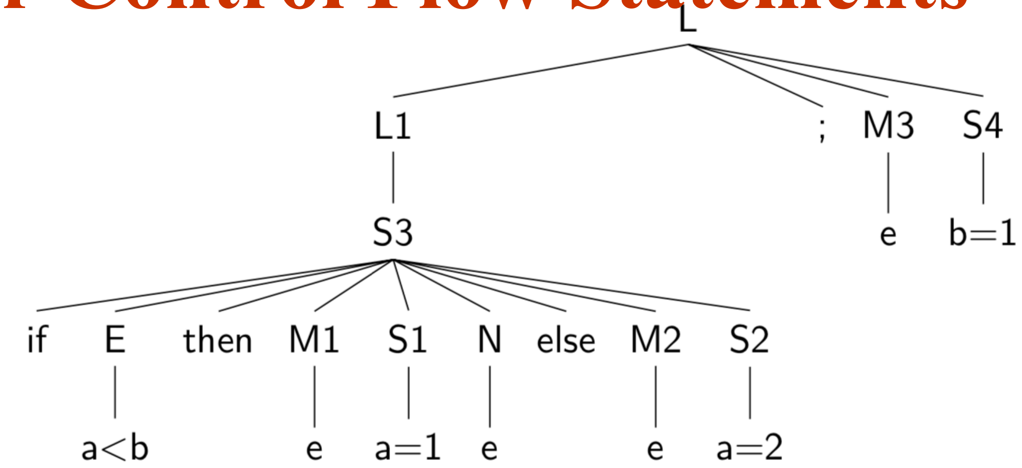
```
if ( a < b )
      a = 1;
else
      a = 2;
b = 1
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

```
100: if a < b goto _
101: goto _
102: a = 1
```

Code Generation for Control Flow Statements



- Example:

```
if ( a < b )
        a = 1;
else
        a = 2;
b = 1
```

E.tlist = makelist(next)=100

E.flist = makelist(next+1)=101

M1 = nextquad = 102

N.Nlist = makelist(next) = 103

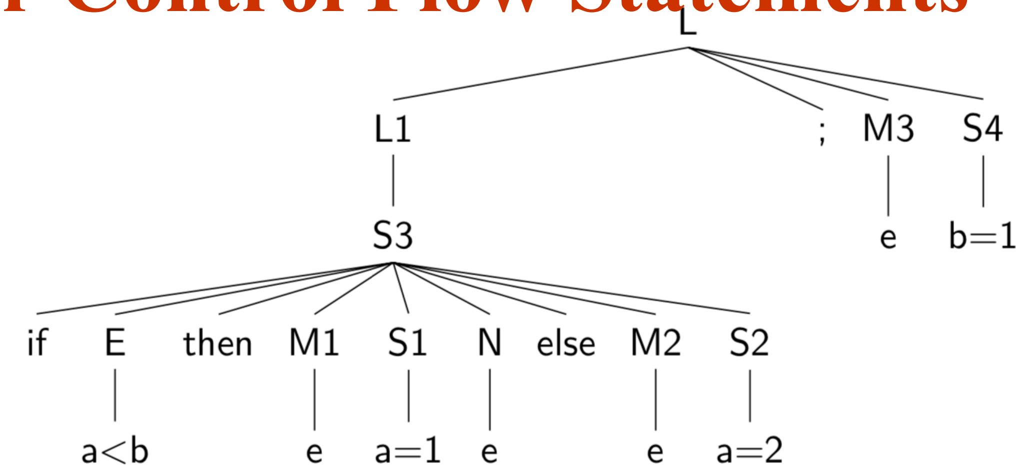
```
100: if a < b goto _
101: goto _
102: a = 1
103: goto _
```

Code Generation for Control Flow Statements

- Example:

```
if ( a < b )
        a = 1;
else
        a = 2;
b = 1
```

```
100: if a < b goto _
101: goto _
102: a = 1
103: goto _
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

N.Nlist = makelist(next) = 103

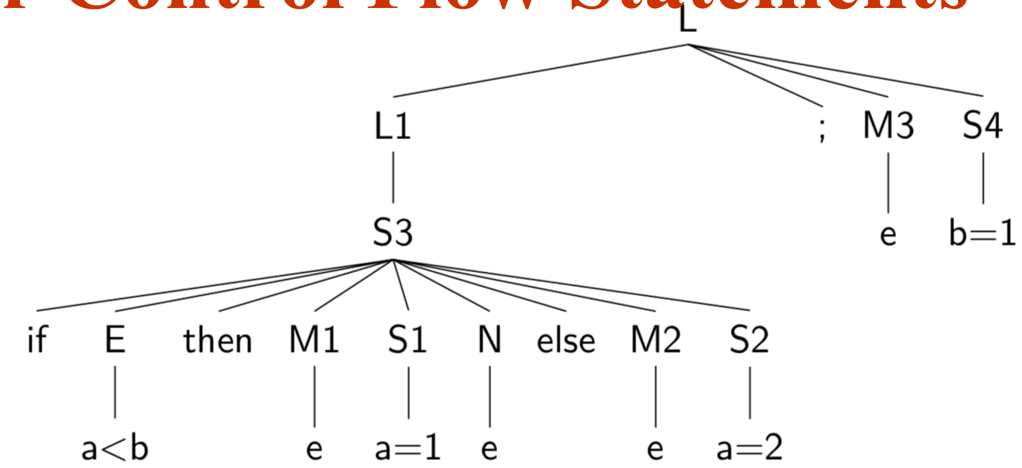
M2 = nextquad = 104

Code Generation for Control Flow Statements

- Example:

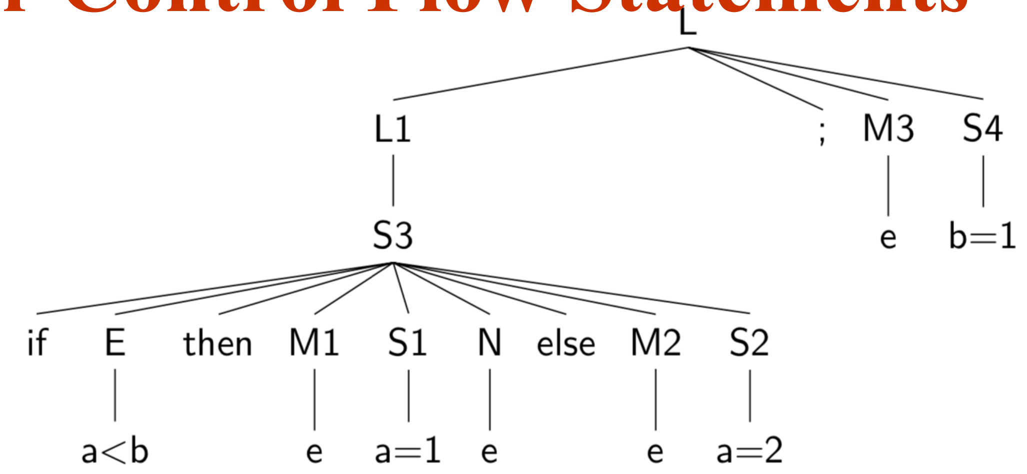
```
if ( a < b )
        a = 1;
else
        a = 2;
b = 1
```

```
100: if a < b goto _
101: goto _
102: a = 1
103: goto _
104: a = 2
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102
N.Nlist = makelist(next) = 103
M2 = nextquad = 104

Code Generation for Control Flow Statements



- Example:

```
if ( a < b )
    a = 1;
else
    a = 2;
b = 1
```

```
100: if a < b goto _
101: goto _
102: a = 1
103: goto _
104: a = 2
```

E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

N.Nlist = makelist(next) = 103

M2 = nextquad = 104

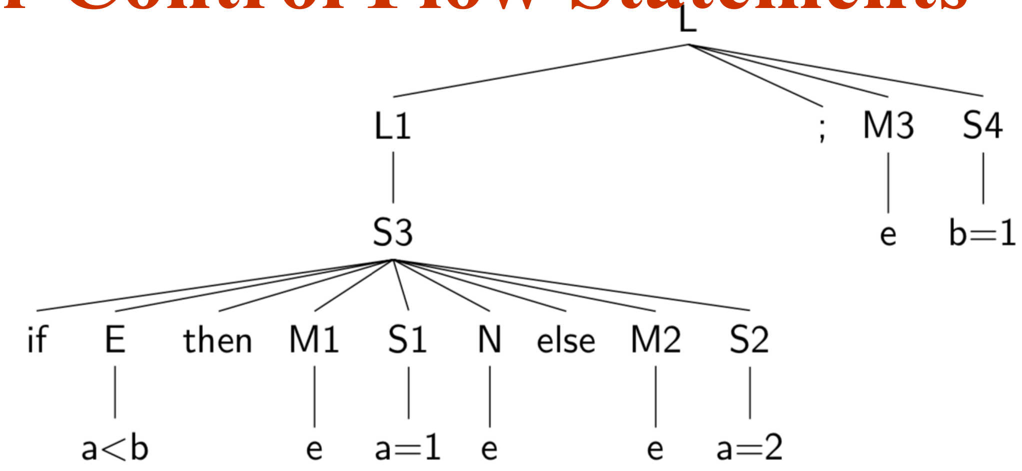
BP(E.tlist, M1.quad) = ({100},102)
BP(E.flist, M2.quad) = ({101},104)
S3.nlist = merge(S1.nlist,S2.nlist, N.nlist)=
(S1.nlist+S2.nlist+103)

Code Generation for Control Flow Statements

- Example:

```
if ( a < b )
        a = 1;
else
        a = 2;
b = 1
```

```
100: if a < b goto 102
101: goto 104
102: a = 1
103: goto _
104: a = 2
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

N.Nlist = makelist(next) = 103

M2 = nextquad = 104

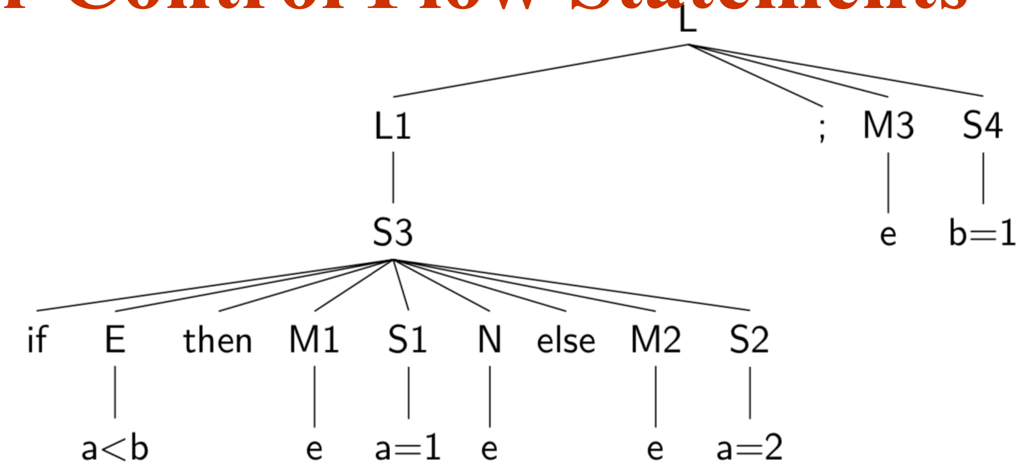
BP(E.tlist, M1.quad) = ({100},102)
BP(E.flist, M2.quad) = ({101},104)
S3.nlist = merge(S1.nlist,S2.nlist, N.nlist)=
(S1.nlist+S2.nlist+103)

Code Generation for Control Flow Statements

- Example:

```
if ( a < b )
      a = 1;
else
      a = 2;
b = 1
```

```
100: if a < b goto 102
101: goto 104
102: a = 1
103: goto _
104: a = 2
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

N.Nlist = makelist(next) = 103

M2 = nextquad = 104

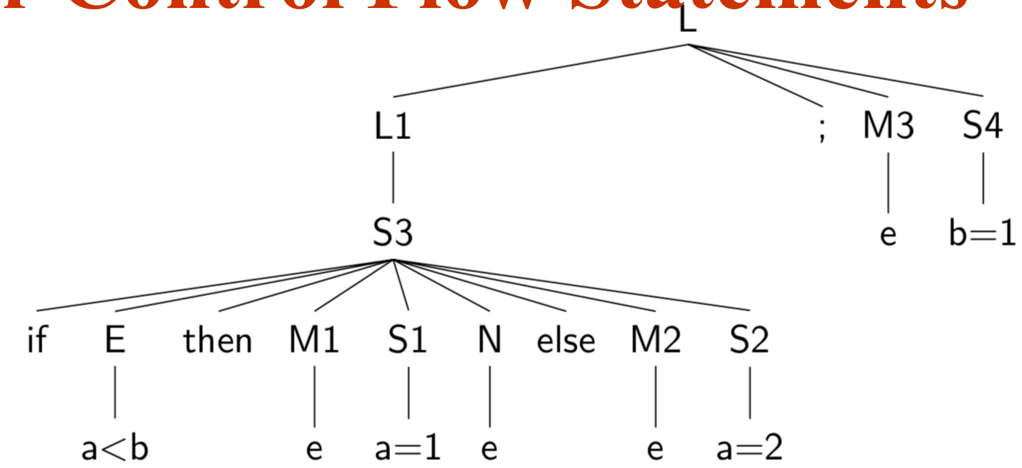
BP(E.tlist, M1.quad) = ({100},102)
BP(E.flist, M2.quad) = ({101},104)
S3.nlist = merge(S1.nlist,S2.nlist, N.nlist)=
(S1.nlist+S2.nlist+103)
L1.nlist = S3.nlist;

Code Generation for Control Flow Statements

- Example:

```
if ( a < b )
      a = 1;
else
      a = 2;
b = 1
```

```
100: if a < b goto 102
101: goto 104
102: a = 1
103: goto _
104: a = 2
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

N.Nlist = makelist(next) = 103

M2 = nextquad = 104

BP(E.tlist, M1.quad) = ({100},102)
BP(E.flist, M2.quad) = ({101},104)
S3.nlist = merge(S1.nlist,S2.nlist, N.nlist)=
(S1.nlist+S2.nlist+103)

L1.nlist = S3.nlist;

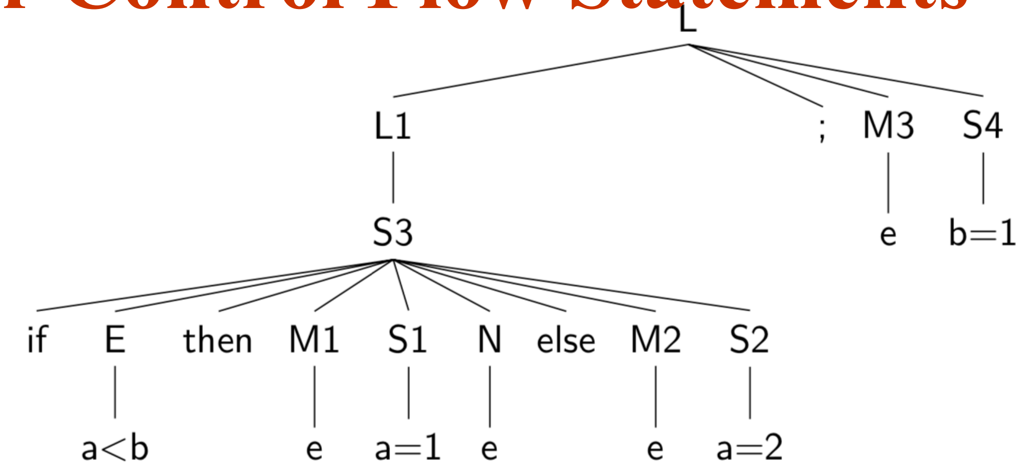
M3 = nextquad = 105

Code Generation for Control Flow Statements

- Example:

```
if ( a < b )
        a = 1;
else
        a = 2;
b = 1
```

```
100: if a < b goto 102
101: goto 104
102: a = 1
103: goto _
104: a = 2
105: b = 1
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

N.Nlist = makelist(next) = 103

M2 = nextquad = 104

BP(E.tlist, M1.quad) = ({100},102)
BP(E.flist, M2.quad) = ({101},104)
S3.nlist = merge(S1.nlist,S2.nlist, N.nlist)=
(S1.nlist+S2.nlist+103)

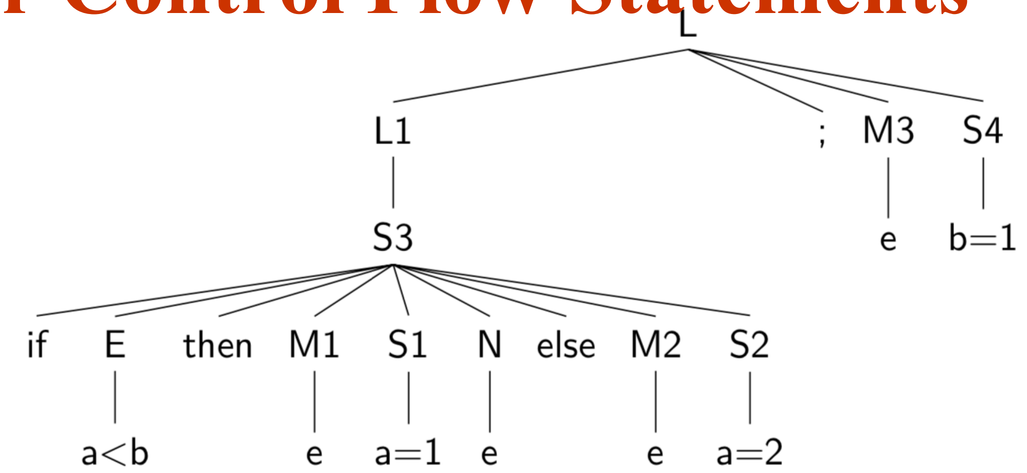
L1.nlist = S3.nlist;

M3 = nextquad = 105

Code Generation for Control Flow Statements

- Example:

```
if ( a < b )
      a = 1;
else
      a = 2;
b = 1
```



```
100: if a < b goto 102
101: goto 104
102: a = 1
103: goto _
104: a = 2
105: b = 1
```

E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

N.Nlist = makelist(next) = 103

M2 = nextquad = 104

BP(E.tlist, M1.quad) = ({100},102)
BP(E.flist, M2.quad) = ({101},104)
S3.nlist = merge(S1.nlist,S2.nlist, N.nlist)=
(S1.nlist+S2.nlist+103)

L1.nlist = S3.nlist;

M3 = nextquad = 105

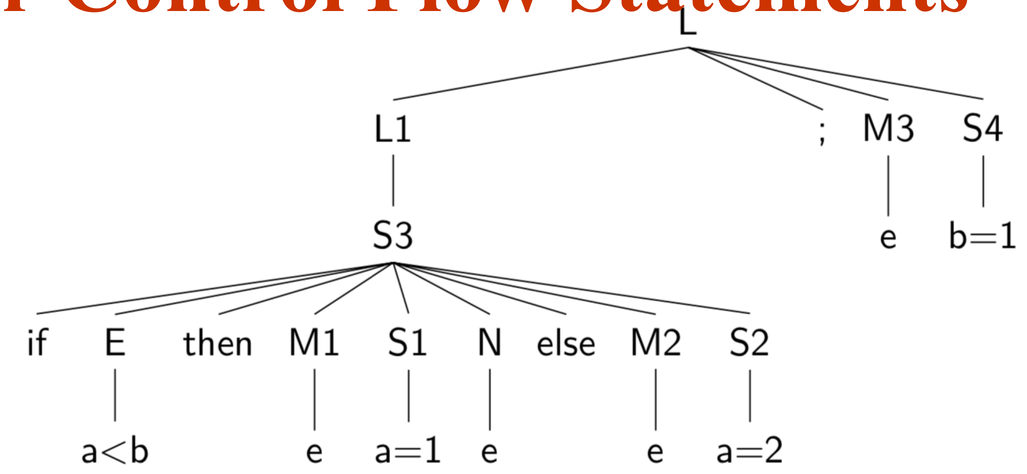
BP(L1.nlist, M.quad) = ({103}, 105)
L.nlist = S3.nlist

Code Generation for Control Flow Statements

- Example:

```
if ( a < b )
        a = 1;
else
        a = 2;
b = 1
```

```
100: if a < b goto 102
101: goto 104
102: a = 1
103: goto 105
104: a = 2
105: b = 1
```



E.tlist = makelist(next)=100
E.flist = makelist(next+1)=101
M1 = nextquad = 102

N.Nlist = makelist(next) = 103

M2 = nextquad = 104

BP(E.tlist, M1.quad) = ({100},102)
BP(E.flist, M2.quad) = ({101},104)
S3.nlist = merge(S1.nlist,S2.nlist, N.nlist)=
(S1.nlist+S2.nlist+103)
L1.nlist = S3.nlist;

M3 = nextquad = 105

BP(L1.nlist, M.quad) = ({103}, 105)
L.nlist = S3.nlist

Summary

- At this stage in compilation, we have
  - an AST,
  - annotated with scope information,
  - and annotated with type information.
- To generate TAC for the program, we do (yet another) recursive tree traversal!
  - Generate TAC for any subexpressions or substatements.
  - Using the result, generate TAC for the overall expression.

Question?