

Compilerconstructie

najaar 2018

<http://www.liacs.leidenuniv.nl/~vlietrvan1/coco/>

Rudy van Vliet

kamer 140 Snellius, tel. 071-527 2876

rvvliet(at)liacs(dot)nl

college 7, vrijdag 2 november 2018

+ werkcollege

Intermediate Code Generation 2

Hoor- en werkcollege van 16 november naar 14 november

Today

- Translation of control flow
 - Top-down passing of labels (inherited attributes)
 - Backpatching (synthesized attributes)
- Translation of switch-statements

6.6 Control Flow

- Boolean expressions used to
 1. **Alter flow of control: `if (E) S`**
 2. Compute logical values, cf. arithmetic expressions

- Generated by

$P \rightarrow S$

$S \rightarrow \mathbf{id = num;} \mid SS$

$\mid \mathbf{if (B) S} \mid \mathbf{if (B) S else S} \mid \mathbf{while (B) S}$

$B \rightarrow B \mid\mid B \mid B \&\& B \mid !B \mid (B) \mid E \mathbf{rel} E \mid \mathbf{true} \mid \mathbf{false}$

- In $B_1 \mid\mid B_2$, if B_1 is true, then expression is true
In $B_1 \&\& B_2$, if ...

6.6.2 Short-Circuit Code

or jumping code

Boolean operators `||`, `&&` and `!` translate into jumps

Example

```
if ( x < 100 || x > 200 && x!=y ) x = 0;
```

Precedence: `||` < `&&` < `!`

```
    if x < 100 goto L2
    ifFalse x > 200 goto L1
    ifFalse x != y goto L1
L2:  x = 0
L1:
```

6.6.3 Flow-of-Control Statements

Translation using

- **synthesized** attributes *B.code* and *S.code*
- **inherited** attributes (labels) *B.true*, *B.false* and *S.next*

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \parallel label(S.next)$
$S \rightarrow \mathbf{id = num;}$	

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \parallel label(S.next)$
$S \rightarrow \mathbf{id = num};$	$S.code = gen(\mathbf{id.addr} \ ' \ = \ ' \ \mathbf{num.val});$
$S \rightarrow S_1S_2$	

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \parallel label(S.next)$
$S \rightarrow \mathbf{id = num};$	$S.code = gen(\mathbf{id.addr} \ ' \ = \ ' \ \mathbf{num.val});$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \parallel label(S_1.next) \parallel S_2.code$
$S \rightarrow \mathbf{if} (B) S_1$	

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \parallel label(S.next)$
$S \rightarrow \mathbf{id = num};$	$S.code = gen(\mathbf{id.addr} \ ' = ' \ \mathbf{num.val});$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \parallel label(S_1.next) \parallel S_2.code$
$S \rightarrow \mathbf{if} (B) S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \parallel label(B.true) \parallel S_1.code$
$S \rightarrow \mathbf{if} (B) S_1 \mathbf{else} S_2$	

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \parallel label(S.next)$
$S \rightarrow id = num;$	$S.code = gen(id.addr \neq num.val);$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \parallel label(S_1.next) \parallel S_2.code$
$S \rightarrow if (B) S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \parallel label(B.true) \parallel S_1.code$
$S \rightarrow if (B) S_1 else S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code \parallel label(B.true) \parallel S_1.code$ $\parallel gen('goto' S.next) \parallel label(B.false) \parallel S_2.code$
$S \rightarrow while (B) S_1$	

Syntax-Directed Definition

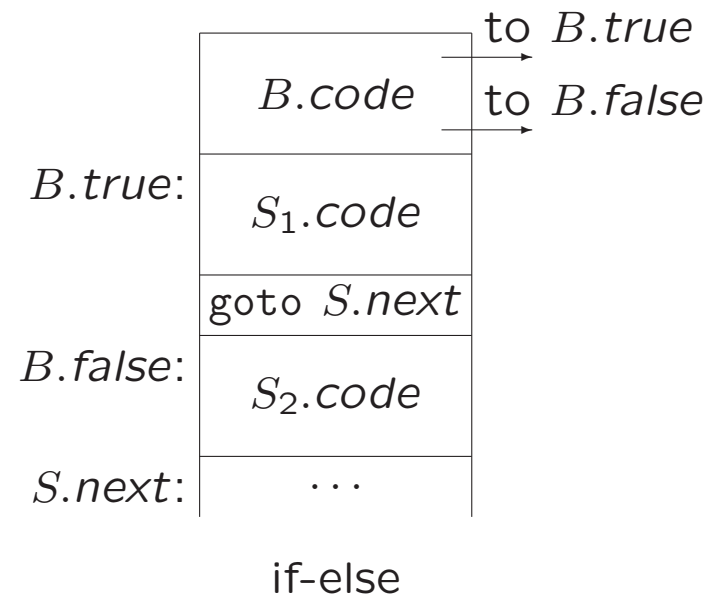
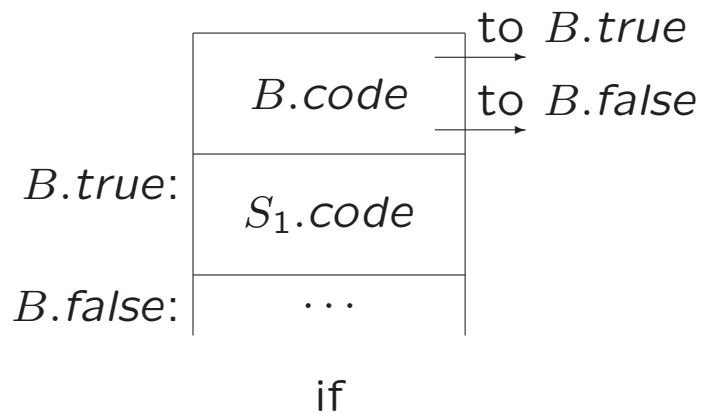
Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$ $P.code = S.code \parallel label(S.next)$
$S \rightarrow \mathbf{id = num};$	$S.code = gen(\mathbf{id.addr} \ '=\ ' \ \mathbf{num.val});$
$S \rightarrow S_1 S_2$	$S_1.next = newlabel()$ $S_2.next = S.next$ $S.code = S_1.code \parallel label(S_1.next) \parallel S_2.code$
$S \rightarrow \mathbf{if} (B) S_1$	$B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \parallel label(B.true) \parallel S_1.code$
$S \rightarrow \mathbf{if} (B) S_1 \mathbf{else} S_2$	$B.true = newlabel()$ $B.false = newlabel()$ $S_1.next = S_2.next = S.next$ $S.code = B.code \parallel label(B.true) \parallel S_1.code$ $\parallel gen('goto' \ S.next) \parallel label(B.false) \parallel S_2.code$
$S \rightarrow \mathbf{while} (B) S_1$	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \parallel B.code \parallel label(B.true)$ $\parallel S_1.code \parallel gen('goto' \ begin)$

6.6.3 Flow-of-Control Statements

$S \rightarrow \mathbf{if} (B) S_1$

$S \rightarrow \mathbf{if} (B) S_1 \mathbf{else} S_2$

$S \rightarrow \mathbf{while} (B) S_1$



Syntax-Directed Definition

Production	Semantic Rules
$B \rightarrow E_1 \mathbf{rel} E_2$	

Syntax-Directed Definition

Production	Semantic Rules
$B \rightarrow E_1 \mathbf{rel} E_2$	$B.code = E_1.code \parallel E_2.code$ $\parallel \mathit{gen}('if' E_1.addr \mathbf{rel.op} E_2.addr 'goto' B.true)$ $\parallel \mathit{gen}('goto' B.false)$
$B \rightarrow B_1 \parallel B_2$	

Syntax-Directed Definition

Production	Semantic Rules
$B \rightarrow E_1 \mathbf{rel} E_2$	$B.code = E_1.code \parallel E_2.code$ $\parallel \text{gen('if' } E_1.addr \mathbf{rel.op} E_2.addr \text{'goto' } B.true)$ $\parallel \text{gen('goto' } B.false)$
$B \rightarrow B_1 B_2$	$B_1.true = B.true$ $B_1.false = \text{newlabel}()$ $B_2.true = B.true$ $B_2.false = B.false$
$B \rightarrow B_1 \&\& B_2$	$B.code = B_1.code \parallel \text{label}(B_1.false) \parallel B_2.code$ $B_1.true = \text{newlabel}()$ $B_1.false = B.false$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \parallel \text{label}(B_1.true) \parallel B_2.code$

Syntax-Directed Definition

Production	Semantic Rules
$P \rightarrow S$	$S.next = newlabel()$
$S \rightarrow \text{if } (B) S_1$	$P.code = S.code \parallel label(S.next)$ $B.true = newlabel()$ $B.false = S_1.next = S.next$ $S.code = B.code \parallel label(B.true) \parallel S_1.code$
$B \rightarrow B_1 B_2$	$B_1.true = B.true$ $B_1.false = newlabel()$ $B_2.true = B.true$ $B_2.false = B.false$ $B.code = B_1.code \parallel label(B_1.false) \parallel B_2.code$
$B_1 \rightarrow E_1 \text{ rel } E_2$	$B_1.code = E_1.code \parallel E_2.code$ $\parallel gen('if' E_1.addr \text{ rel.op } E_2.addr 'goto' B_1.true)$ $\parallel gen('goto' B_1.false)$
$B_2 \rightarrow B_3 \&\& B_4$	$B_3.true = newlabel()$ $B_3.false = B_2.false$ $B_4.true = B_2.true$ $B_4.false = B_2.false$ $B_2.code = B_3.code \parallel label(B_3.true) \parallel B_4.code$
$S_1 \rightarrow \text{id} = \text{num};$	$S_1.code = gen(\text{id.addr} ' = ' \text{num.val});$

Example: `if (x < 100 || x > 200 && x != y) x = 0;`

6.6.5 Avoiding Redundant Gotos

```
    if x < 100 goto L2
    goto L3
L3:  if x > 200 goto L4
    goto L1
L4:  if x != y goto L2
    goto L1
L2:  x = 0
L1:
```

Versus

```
    if x < 100 goto L2
    ifFalse x > 200 goto L1
    ifFalse x != y goto L1
L2:  x = 0
L1:
```

6.7 Backpatching

- Code generation problem:
 - Labels (addresses) that control must go to may not be known at the time that jump statements are generated
- One solution:
 - Separate pass to bind labels to addresses

<i>B.code</i>	
	103 goto <i>B.true</i>
	107 goto <i>B.false</i>
	110 goto <i>B.false</i>
	118 goto <i>B.true</i>
<i>B.true:</i>	128 ...
<i>B.false:</i>	135 ...

6.7 Backpatching

- Code generation problem:
 - Labels (addresses) that control must go to may not be known at the time that jump statements are generated
- Other solution: backpatching
 - Generate jump statements with empty target
 - Add such statements to a list
 - Fill in targets when proper address is determined

6.7.1 One-Pass Code Generation Using Backpatching

- **Synthesized** attributes *B.truelist*, *B.falselist*, *S.nextlist* containing lists of jumps
- Three functions
 1. *makelist(i)* creates new list containing index *i*
 2. *merge(p₁, p₂)* concatenates lists pointed to by *p₁* and *p₂*
 3. *backpatch(p, i)* inserts *i* as target label for each instruction on list pointed to by *p*

B.true/B.false vs B.truelist/B.falselist

	<i>B.code</i>
	103 goto <i>B.true</i>
	107 goto <i>B.false</i>
	110 goto <i>B.false</i>
	118 goto <i>B.true</i>
<i>B.true:</i>	128 ···
<i>B.false:</i>	135 ···

	<i>B.code</i>
	103 goto _
	107 goto _
	110 goto _
	118 goto _
	128 ···
	135 ···

B.truelist = {103, 118}

B.falselist = {107, 110}

Analogous: *S.next* vs *S.nextlist*

Grammars for Backpatching

- Grammar for boolean expressions:

$$\begin{aligned} B &\rightarrow B_1 || MB_2 \mid B_1 \&\& MB_2 \mid !B_1 \mid (B_1) \\ &\quad \mid E_1 \text{ rel } E_2 \mid \text{true} \mid \text{false} \\ M &\rightarrow \epsilon \end{aligned}$$

M is marker nonterminal

- Grammar for flow-of-control statements
(marker nonterminals omitted for readability)

$$\begin{aligned} S &\rightarrow \text{if } (B) S_1 \mid \text{if } (B) S_1 \text{ else } S_2 \\ &\quad \mid \text{while } (B) S_1 \mid \{L\} \mid \text{id} = \text{num}; \\ L &\rightarrow L_1 S \mid S \end{aligned}$$

Translation Scheme for Backpatching

$$B \rightarrow E_1 \text{ rel } E_2$$

Translation Scheme for Backpatching

$$\begin{aligned} B \rightarrow E_1 \text{ rel } E_2 & \{ B.\text{truelist} = \text{makelist}(\text{nextinstr}); \\ & B.\text{falselist} = \text{makelist}(\text{nextinstr} + 1); \\ & \text{gen}(\text{'if' } E_1.\text{addr rel.op } E_2.\text{addr 'goto -'}); \\ & \text{gen}(\text{'goto -'}); \} \\ M \rightarrow \epsilon & \{ M.\text{instr} = \text{nextinstr}; \} \\ B \rightarrow B_1 || MB_2 & \end{aligned}$$

Translation Scheme for Backpatching

$B \rightarrow E_1 \text{ rel } E_2$ { $B.\text{truelist} = \text{makelist}(\text{nextinstr});$
 $B.\text{falselist} = \text{makelist}(\text{nextinstr} + 1);$
 $\text{gen}(\text{'if' } E_1.\text{addr rel.op } E_2.\text{addr 'goto -'});$
 $\text{gen}(\text{'goto -'});$ }

$M \rightarrow \epsilon$ { $M.\text{instr} = \text{nextinstr};$ }

$B \rightarrow B_1 || MB_2$ { $\text{backpatch}(B_1.\text{falselist}, M.\text{instr});$
 $B.\text{truelist} = \text{merge}(B_1.\text{truelist}, B_2.\text{truelist});$
 $B.\text{falselist} = B_2.\text{falselist};$ }

$B \rightarrow B_1 \&\& MB_2$ { $\text{backpatch}(B_1.\text{truelist}, M.\text{instr});$
 $B.\text{truelist} = B_2.\text{truelist};$
 $B.\text{falselist} = \text{merge}(B_1.\text{falselist}, B_2.\text{falselist});$ }

$S \rightarrow \text{id} = \text{num};$

Translation Scheme for Backpatching

$B \rightarrow E_1 \text{ rel } E_2$ { $B.\text{truelist} = \text{makelist}(\text{nextinstr});$
 $B.\text{falselist} = \text{makelist}(\text{nextinstr} + 1);$
 $\text{gen}(\text{'if' } E_1.\text{addr rel.op } E_2.\text{addr 'goto -'});$
 $\text{gen}(\text{'goto -'});$ }

$M \rightarrow \epsilon$ { $M.\text{instr} = \text{nextinstr};$ }

$B \rightarrow B_1 || M B_2$ { $\text{backpatch}(B_1.\text{falselist}, M.\text{instr});$
 $B.\text{truelist} = \text{merge}(B_1.\text{truelist}, B_2.\text{truelist});$
 $B.\text{falselist} = B_2.\text{falselist};$ }

$B \rightarrow B_1 \&\& M B_2$ { $\text{backpatch}(B_1.\text{truelist}, M.\text{instr});$
 $B.\text{truelist} = B_2.\text{truelist};$
 $B.\text{falselist} = \text{merge}(B_1.\text{falselist}, B_2.\text{falselist});$ }

$S \rightarrow \text{id} = \text{num};$ { $S.\text{nextlist} = \text{null};$
 $\text{gen}(\text{id.addr ' = ' num.val});$ }

$S \rightarrow \text{if } (B) M S_1$

Translation Scheme for Backpatching

$B \rightarrow E_1 \text{ rel } E_2$	{ $B.truelist = makelist(nextinstr);$ $B.falselist = makelist(nextinstr + 1);$ $gen('if' E_1.addr \text{ rel.op } E_2.addr 'goto _');$ $gen('goto _');$ }
$M \rightarrow \epsilon$	{ $M.instr = nextinstr; \}$
$B \rightarrow B_1 MB_2$	{ $backpatch(B_1.falselist, M.instr);$ $B.truelist = merge(B_1.truelist, B_2.truelist);$ $B.falselist = B_2.falselist; \}$
$B \rightarrow B_1 \&\& MB_2$	{ $backpatch(B_1.truelist, M.instr);$ $B.truelist = B_2.truelist;$ $B.falselist = merge(B_1.falselist, B_2.falselist); \}$
$S \rightarrow \text{id} = \text{num};$	{ $S.nextlist = \text{null};$ $gen(\text{id.addr} ' = ' \text{num.val}); \}$
$S \rightarrow \text{if } (B) MS_1$	{ $backpatch(B.truelist, M.instr);$ $S.nextlist = merge(B.falselist, S_1.nextlist); \}$

Translation Scheme for Backpatching

$S \rightarrow \mathbf{if} (B) MS_1$	{ $backpatch(B.truelist, M.instr);$ $S.nextlist = merge(B.falselist, S_1.nextlist);$ }
$S_1 \rightarrow \mathbf{id = num;}$	{ $S_1.nextlist = \mathbf{null};$ $gen(\mathbf{id.addr} \ '=\ ' \ \mathbf{num.val});$ }
$B \rightarrow B_1 M_1 B_2$	{ $backpatch(B_1.falselist, M_1.instr);$ $B.truelist = merge(B_1.truelist, B_2.truelist);$ $B.falselist = B_2.falselist;$ }
$B_2 \rightarrow B_3 \&\& M_2 B_4$	{ $backpatch(B_3.truelist, M_2.instr);$ $B_2.truelist = B_4.truelist;$ $B_2.falselist = merge(B_3.falselist, B_4.falselist);$ }
$B \rightarrow E_1 \ \mathbf{rel} \ E_2$	{ $B.truelist = makelist(nextinstr);$ $B.falselist = makelist(nextinstr + 1);$ $gen(\mathbf{'if' E_1.addr} \ \mathbf{rel.op} \ \mathbf{E_2.addr} \ \mathbf{'goto -'});$ $gen(\mathbf{'goto -'});$ }
$M \rightarrow \epsilon$	{ $M.instr = nextinstr;$ }

Example: `if (x < 100 || x > 200 && x != y) x = 0;`

Exercises 2 and 3

Translation Scheme for Backpatching

For Exercise 2

(Boolean Expressions)

$$\begin{aligned} B \rightarrow B_1 \&\& M_1 B_2 & \{ \text{backpatch}(B_1.\text{truelist}, M_1.\text{instr}); \\ & B.\text{truelist} = B_2.\text{truelist}; \\ & B.\text{falselist} = \text{merge}(B_1.\text{falselist}, B_2.\text{falselist}); \} \\ B_2 \rightarrow (B_3) & \{ B_2.\text{truelist} = B_3.\text{truelist}; \\ & B_2.\text{falselist} = B_3.\text{falselist}; \} \\ B_3 \rightarrow B_4 \mid \mid M_2 B_5 & \{ \text{backpatch}(B_4.\text{falselist}, M_2.\text{instr}); \\ & B_3.\text{truelist} = \text{merge}(B_4.\text{truelist}, B_5.\text{truelist}); \\ & B_3.\text{falselist} = B_5.\text{falselist}; \} \\ B \rightarrow E_1 \text{ rel } E_2 & \{ B.\text{truelist} = \text{makelist}(\text{nextinstr}); \\ & B.\text{falselist} = \text{makelist}(\text{nextinstr} + 1); \\ & \text{gen}(\text{'if' } E_1.\text{addr rel.op } E_2.\text{addr 'goto -'}); \\ & \text{gen}(\text{'goto -'}); \} \\ M \rightarrow \epsilon & \{ M.\text{instr} = \text{nextinstr}; \} \end{aligned}$$

Translation Scheme for Backpatching

For Exercise 3

(Flow-of-Control Statements)

$S \rightarrow \{L\}$	{ $S.nextlist = L.nextlist;$ }
$L \rightarrow L_1 M_3 S_1$	{ $backpatch(L_1.nextlist, M_3.instr);$ $L.nextlist = S_1.nextlist;$ }
$L_1 \rightarrow S_2$	{ $L_1.nextlist = S_2.nextlist;$ }
$S_2 \rightarrow \mathbf{if} (B) M_4 S_3$	{ $backpatch(B.truelist, M_4.instr);$ $S_2.nextlist = merge(B.falselist, S_3.nextlist);$ }
$S_3 \rightarrow \mathbf{id = num;}$	{ $S_3.nextlist = \mathbf{null};$ $gen(\mathbf{id.addr} \neq \mathbf{num.val});$ }
$M \rightarrow \epsilon$	{ $M.instr = nextinstr;$ }

6.8 Switch-Statements

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

Translation:

1. Evaluate expression E
2. Find value V_j in list of cases that matches value of E
3. Execute statement S_j

Implementation Step 2 (Back End)

- Small number of cases: sequence of conditional jumps
- Larger number of cases: hash table
- Values in a small range [min–max]: array[min-max] of labels

Translation of Switch-Statement

Straightforward

```
        code to evaluate E into t
        if t != V1 goto L1
        code for S1
        goto next
L1:     if t != V2 goto L2
        code for S2
        goto next
L2:     ...
        ...
L_{n-2}: if t != V_{n-1} goto L_{n-1}
        code for S_{n-1}
        goto next
L_{n-1}: code for S_n
next:
```

Translation of Switch-Statement

Easier for back end to recognize multiway branch:
combine tests

```
    code to evaluate E into t
    if t = V1 goto L1
    if t = V2 goto L2
    ...
    if t = V_{n-1} goto L_{n-1}
    goto L_{n}
L1:  code for S1
     goto next
L2:  code for S2
     goto next
    ...
L_{n-1}: code for S_{(n-1)}
        goto next
L_{n}:  code for S_n
next:
```

Translation of Switch-Statement

Easier to generate: tests at the end

```
        code to evaluate E into t
        goto test
L1:   code for S1
        goto next
L2:   code for S2
        goto next
        ...
L_{n-1}: code for S_{n-1}
        goto next
L_{n}:   code for S_n
        goto next
test: if t = V1 goto L1
        if t = V2 goto L2
        ...
        if t = V_{n-1} goto L_{n-1}
        goto L_{n}
next:
```

Translation of Switch-Statement

Even easier for back end to recognize: case statements

```
        code to evaluate E into t
        goto test
L1:   code for S1
        goto next
L2:   code for S2
        goto next
        ...
L_{n-1}: code for S_{(n-1)}
        goto next
L_{n}:   code for S_n
        goto next
test: case t V1 L1
        case t V2 L2
        ...
        case t V_{n-1} L_{n-1}
        case t t L_{n}
next:
```

Vervolgens. . .

- Nu: introductie opdracht 3
- Vrijdag 9 november, 13.30-15.15: practicum
- Inleveren 22 november
- **Woensdag 14 november**, hoorcollege + werkcollege

Compilerconstructie

college 7

Intermediate Code Generation 2

Chapters for reading:
6.6–top-of-page-406,
6.7–6.7.3, 6.8