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 6, vrijdag 26 oktober 2018

Intermediate Code Generation 1

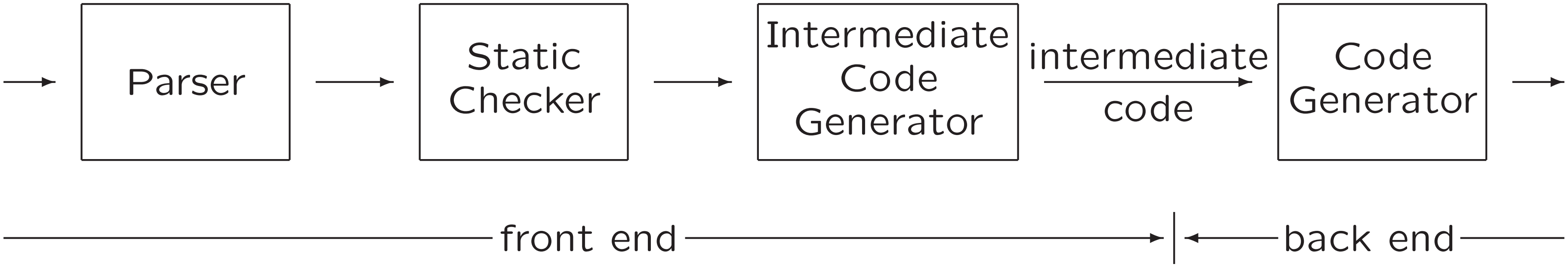
Hoor- en werkcollege van 16 november naar 14 november

Today

- Types of three-address instructions
- Implementations of three-address instructions
- Translation of expressions
- Translation of array references

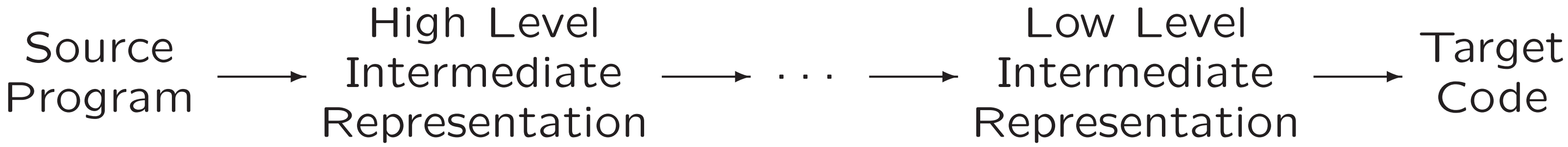
6. Intermediate Code Generation

- Front end: generates intermediate representation
- Back end: generates target code



Intermediate Representation

- Facilitates efficient compiler suites: $m + n$ instead of $m * n$
- Different types, e.g.,
  - syntax trees
  - three-address code: $x = y \; op \; z$
- High-level vs. low-level
- C for C++



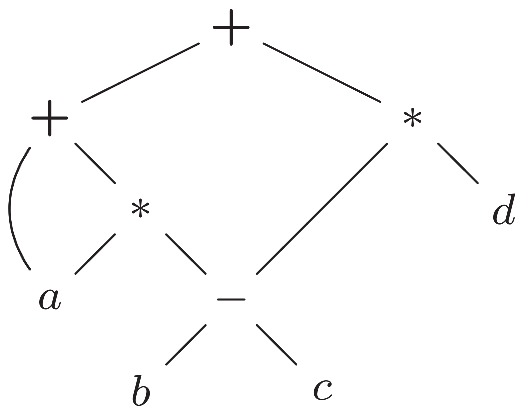
Source Program → High Level Intermediate Representation → ··· → Low Level Intermediate Representation → Target Code

6.2 Three-Address Code

- Linearized representation of syntax tree / syntax DAG
- Sequence of instructions: $x = y$ *op* z

Example: $a + a * (b - c) + (b - c) * d$

Syntax DAG



Three-address code

```
t1 = b - c
t2 = a * t1
t3 = a + t2
t4 = t1 * d
t5 = t3 + t4
```

6.2.1 Addresses and Instructions

At most three addresses per instruction

- Name: source program name / symbol-table entry
- Constant
- Compiler-generated temporary: distinct names

Three-Address Instructions

1.	Assignment instructions	$x = y$ *op* z
2.	Assignment instructions	$x =$ *op* y
3.	Copy instructions	$x = y$
4.	Unconditional jumps	`goto` L
5.	Conditional jumps	`if` x `goto` L / `ifFalse` x `goto` L
6.	Conditional jumps	`if` x *relop* y `goto` L / `ifFalse`...
7.	Procedure calls and returns	`param` x_1 `param` x_2 ... `param` x_n `call` p, n `return` y
8.	Indexed copy instructions	$x = y[i]$ / $x[i] = y$
9.	Address and pointer assignments	$x = \&y$, $\quad x = *y$, $\quad *x = y$

Symbolic label L represents index of instruction

Three-Address Instructions (Example)

```
do i = i+1; while (a[i] < v);
```

Syntax tree. . .

Three-Address Instructions (Example)

```
do i = i+1; while (a[i] < v);
```

Syntax tree. . .

Two examples of possible translations:

```
Symbolic labels               Position numbers
  L:  t1 = i+1                  100:  t1 = i+1
      i = t1                    101:  i = t1
      t2 = i * 8                102:  t2 = i * 8
      t3 = a [ t2 ]             103:  t3 = a [ t2 ]
      if t3 < v goto L          104:  if t3 < v goto 100
```

Implementation of Three-Address Instructions

Quadruples: records *op*, *vararg*1, *vararg*2, *result*

Example: `a = b * - c + b * - c`

Syntax tree. . .

Implementation of Three-Address Instructions

Quadruples: records *op*, *vararg*1, *vararg*2, *result*

Example: `a = b * - c + b * - c`

Syntax tree. . .

Three-address code

```
  t1 = minus c
  t2 = b * t1
  t3 = minus c
  t4 = b * t3
  t5 = t2 + t4
   a = t5
```

	op	*vararg*1	*vararg*2	*result*
0	`minus`	c		t_1
1	$*$	b	t_1	t_2
2	`minus`	c		t_3
3	$*$	b	t_3	t_4
4	$+$	t_2	t_4	t_5
5	$=$	t_5		a
	…			

Implementation of Three-Address Instructions

Three-address code

```
t1 = minus c
t2 = b * t1
t3 = minus c
t4 = b * t3
t5 = t2 + t4
 a = t5
```

	op	*vararg*1	*vararg*2	*result*
0	`minus`	c		t_1
1	$*$	b	t_1	t_2
2	`minus`	c		t_3
3	$*$	b	t_3	t_4
4	$+$	t_2	t_4	t_5
5	$=$	t_5		a
	$\ldots$			

Exceptions

1. `minus`, $=$
2. `param`
3. jumps

Field *result* mainly for temporaries. . .

Implementation of Three-Address Instructions

Triples: records *op*, *vararg*1, *vararg*2

Example: `a = b * - c + b * - c`

Syntax tree. . .

Three-address code

```
  t1 = minus c
  t2 = b * t1
  t3 = minus c
  t4 = b * t3
  t5 = t2 + t4
   a = t5
```

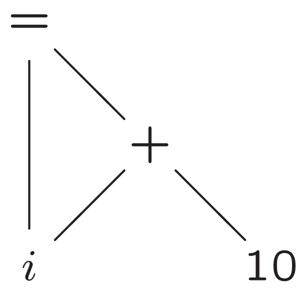
	op	*vararg*1	*vararg*2
0	`minus`	c	
1	$*$	b	(0)
2	`minus`	c	
3	$*$	b	(2)
4	$+$	(1)	(3)
5	$=$	a	(4)
		$\ldots$	

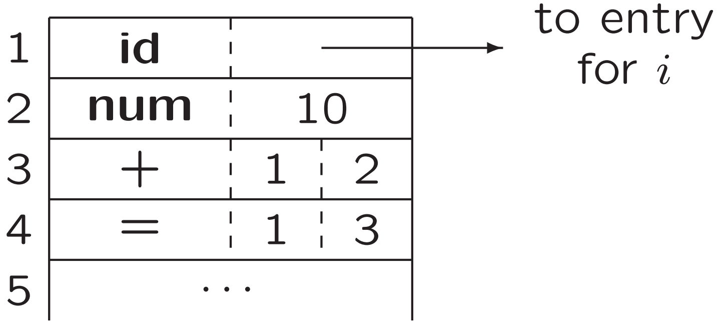
A slide from lecture 5:

6.1.2 The Value-Number Method

An implementation of DAG

DAG for $i = i + 10$





1	**id**		
2	**num**	10	
3	$+$	1	2
4	$=$	1	3
5	$\cdots$		

to entry
for i

- Search array for (existing) node
- Use hash table

Implementation of Three-Address Instructions

Three-address code

```
t1 = minus c
t2 = b * t1
t3 = minus c
t4 = b * t3
t5 = t2 + t4
 a = t5
```

	op	*vararg*1	*vararg*2
0	`minus`	c	
1	$*$	b	(0)
2	`minus`	c	
3	$*$	b	(2)
4	$+$	(1)	(3)
5	$=$	a	(4)
	...		

Equivalent to DAG

Pro: temporaries are implicit
Con: difficult to rearrange code
Indirect triples...

Three-Address Instructions in quadruples, triples. . .

1. Assignment instructions	$x = y\ \mathit{op}\ z$
2. Assignment instructions	$x = \mathit{op}\ y$
3. Copy instructions	$x = y$
4. Unconditional jumps	$\texttt{goto}\ L$
5. Conditional jumps	$\texttt{if}\ x\ \texttt{goto}\ L$ / $\texttt{ifFalse}\ x\ \texttt{goto}\ L$
6. Conditional jumps	$\texttt{if}\ x\ \mathit{relop}\ y\ \texttt{goto}\ L$ / $\texttt{ifFalse}\ldots$
7. Procedure calls and returns	$\texttt{param}\ x_1$ $\texttt{param}\ x_2$ $\ldots$ $\texttt{param}\ x_n$ $\texttt{call}\ p, n$ $\texttt{return}\ y$
8. Indexed copy instructions	$x = y[i]$ / $x[i] = y$
9. Address and pointer assignments	$x = \&y, \quad x = *y, \quad *x = y$

Symbolic label L represents index of instruction

6.4 Translation of Expressions

- Temporary names are created
  $E \rightarrow E_1 + E_2$ yields $t = E_1 + E_2$, e.g.,

```
t5 = t2 + t4
 a = t5
```

- If expression is identifier, then no new temporary
- Nonterminal E has two attributes:
  - E.*addr* – address that will hold value of E
  - E.*code* – three-address code sequence
- Nonterminal S has one attribute:
  - S.*code* – three-address code sequence

6.4.1 Operations Within Expressions

Syntax-directed definition
to produce three-address code for assignments

Production	Semantic Rules
$S \rightarrow \textbf{id} = E;$	$S.code = E.code \Vert$ $gen(top.get(\textbf{id}.lexeme)\ '='\ E.addr)$
$E \rightarrow E_1 + E_2$	$E.addr = \textbf{new}\ Temp()$ $E.code = E_1.code \Vert E_2.code \Vert$ $gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr)$
$\mid\ -E_1$	$E.addr = \textbf{new}\ Temp()$ $E.code = E_1.code \Vert$ $gen(E.addr\ '='\ '\textbf{minus}'\ E_1.addr)$
$\mid\ (E_1)$	$E.addr = E_1.addr$ $E.code = E_1.code$
$\mid\ \textbf{id}$	$E.addr = top.get(\textbf{id}.lexeme)$ $E.code = ''$

Example: $a = b + -c$ . . .

6.4.2 Incremental Translation

Translation scheme

to produce three-address code for assignments

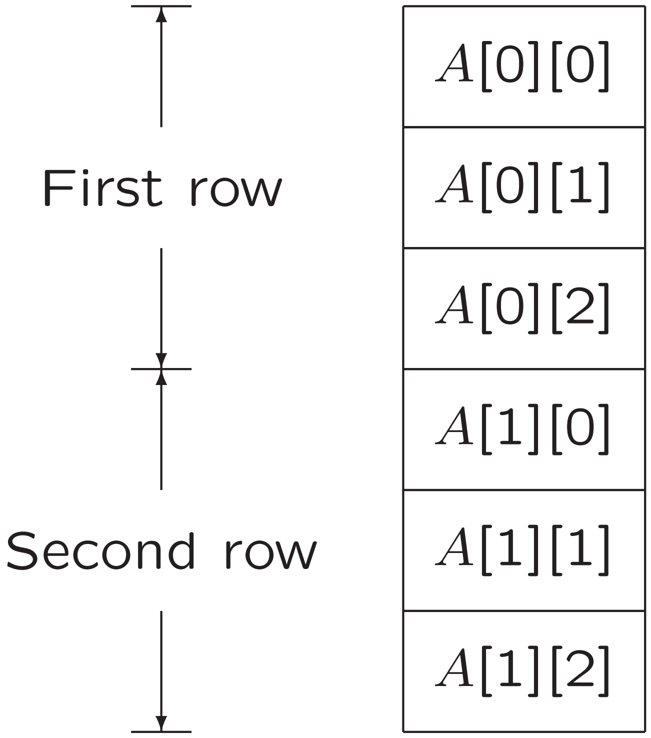
$$\begin{array}{lcll}
S & \rightarrow & \textbf{id} = E; & \{\ \mathit{gen}(\mathit{top.get}(\textbf{id}.\mathit{lexeme})\ '=' \ E.\mathit{addr}); \} \\
E & \rightarrow & E_1 + E_2 & \{\ E.\mathit{addr} = \textbf{new}\ \ \mathit{Temp}(); \\
 & & & \quad \mathit{gen}(E.\mathit{addr}\ '=' \ E_1.\mathit{addr}\ '+' \ E_2.\mathit{addr}); \} \\
 & | & -E_1 & \{\ E.\mathit{addr} = \textbf{new}\ \ \mathit{Temp}(); \\
 & & & \quad \mathit{gen}(E.\mathit{addr}\ '='\ \ '\textbf{minus}' \ E_1.\mathit{addr}); \} \\
 & | & (E_1) & \{\ E.\mathit{addr} = E_1.\mathit{addr}; \} \\
 & | & \textbf{id} & \{\ E.\mathit{addr} = \mathit{top.get}(\textbf{id}.\mathit{lexeme}); \}
\end{array}$$

6.4.3 Addressing Array Elements

- Array $A[n]$ with elements at positions $0, 1, \ldots, n-1$
- Let
  - w be width of array element
  - *base* be relative address of storage allocated for A $(= A[0])$

  Element $A[i]$ begins in location $\quad$ *base* $+ i \times w$
- In two dimensions. . .

Two-Dimensional Arrays



First row

Second row

$A[0][0]$

$A[0][1]$

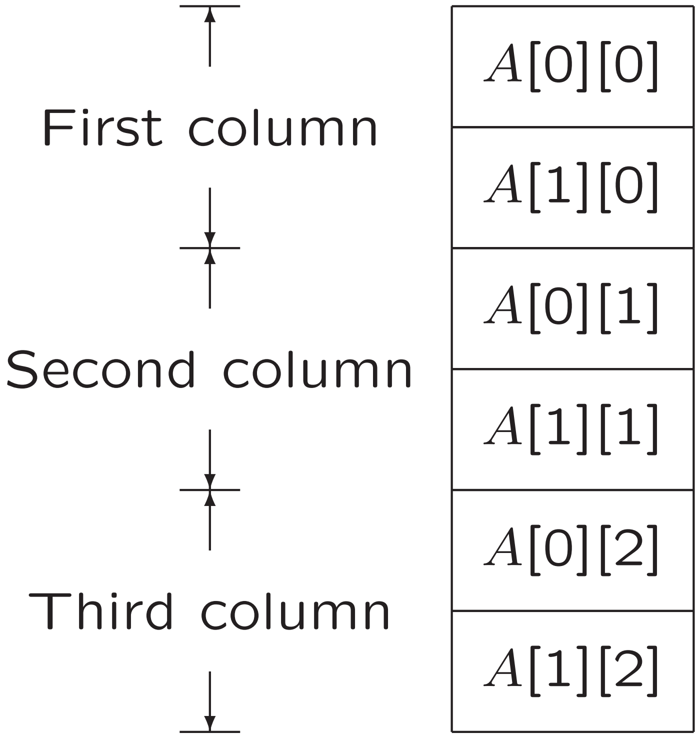
$A[0][2]$

$A[1][0]$

$A[1][1]$

$A[1][2]$

Row Major



First column

Second column

Third column

$A[0][0]$

$A[1][0]$

$A[0][1]$

$A[1][1]$

$A[0][2]$

$A[1][2]$

Column Major

6.4.3 Addressing Array Elements

- In two dimensions, let
  - w_1 be width of row,
  - w_2 be width of element of row

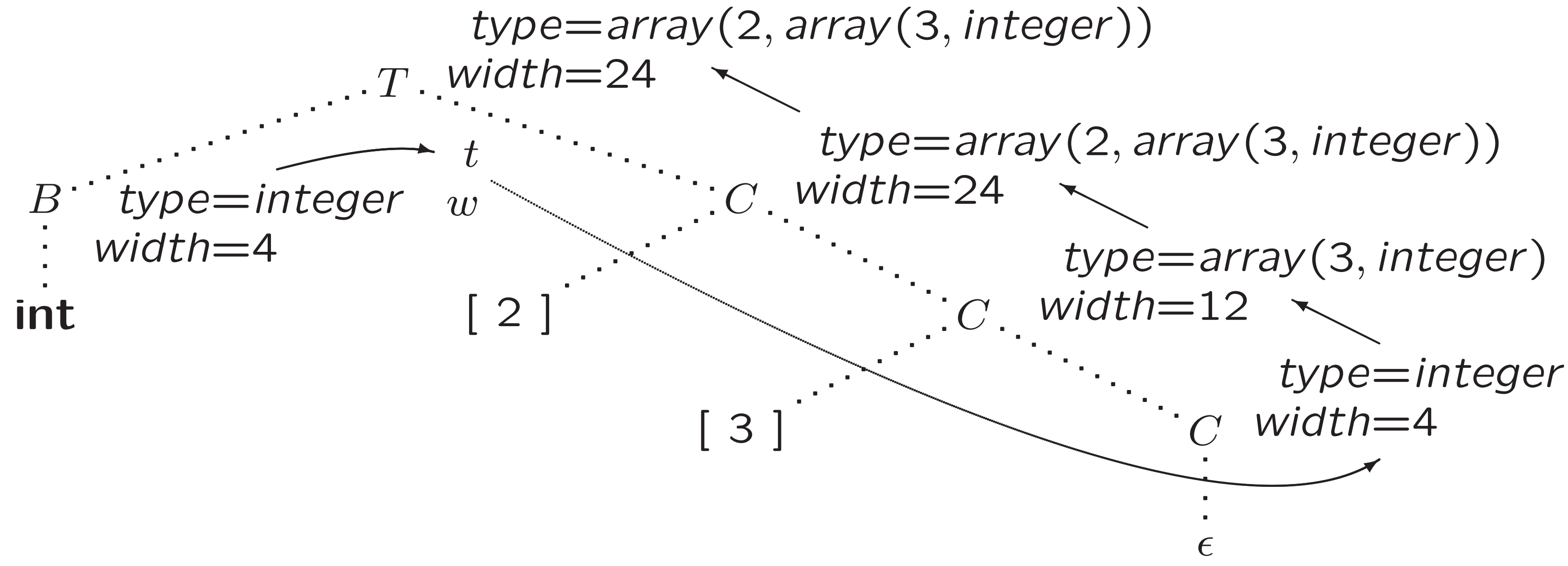
  Element $A[i][j]$ begins in location $\textit{base} + i \times w_1 + j \times w_2$

- In k dimensions $\textit{base} + i_1 * w_1 + i_2 * w_2 + \cdots + i_k * w_k$

A slide from lecture 5:

Types and Their Widths (Example)

$$
\begin{array}{lcll}
T & \rightarrow & B & \{\ t = B.\mathit{type};\ \ w = B.\mathit{width};\} \\
 & & C & \{\ T.\mathit{type} = C.\mathit{type};\ \ T.\mathit{width} = C.\mathit{width};\} \\
B & \rightarrow & \textbf{int} & \{\ B.\mathit{type} = \mathit{integer};\ \ B.\mathit{width} = 4;\} \\
B & \rightarrow & \textbf{float} & \{\ B.\mathit{type} = \mathit{float};\ \ B.\mathit{width} = 8;\} \\
C & \rightarrow & \epsilon & \{\ C.\mathit{type} = t;\ \ C.\mathit{width} = w;\} \\
C & \rightarrow & [\ \textbf{num}\ ]\ C_1 & \{\ C.\mathit{type} = \mathit{array}(\textbf{num}.\mathit{value},\ C_1.\mathit{type}); \\
 & & & \phantom{\{}\ C.\mathit{width} = \textbf{num}.\mathit{value} \times C_1.\mathit{width};\}
\end{array}
$$



Addressing Array Elements

More general: `int A[low..high];`

- $\mathit{base} + (i - \mathit{low}) \times w = i \times w + \underbrace{\mathit{base} - \mathit{low} \times w}_{c}$

- More dimensions. . .

- Precalculate c

- Dynamic arrays. . .

6.4.4 Translation of Array References

L generates array name followed by sequence of index expressions

$$\begin{array}{rcl} E & \rightarrow & E + E \mid \textbf{id} \mid L \\ L & \rightarrow & L[E] \mid \textbf{id}[E] \end{array}$$

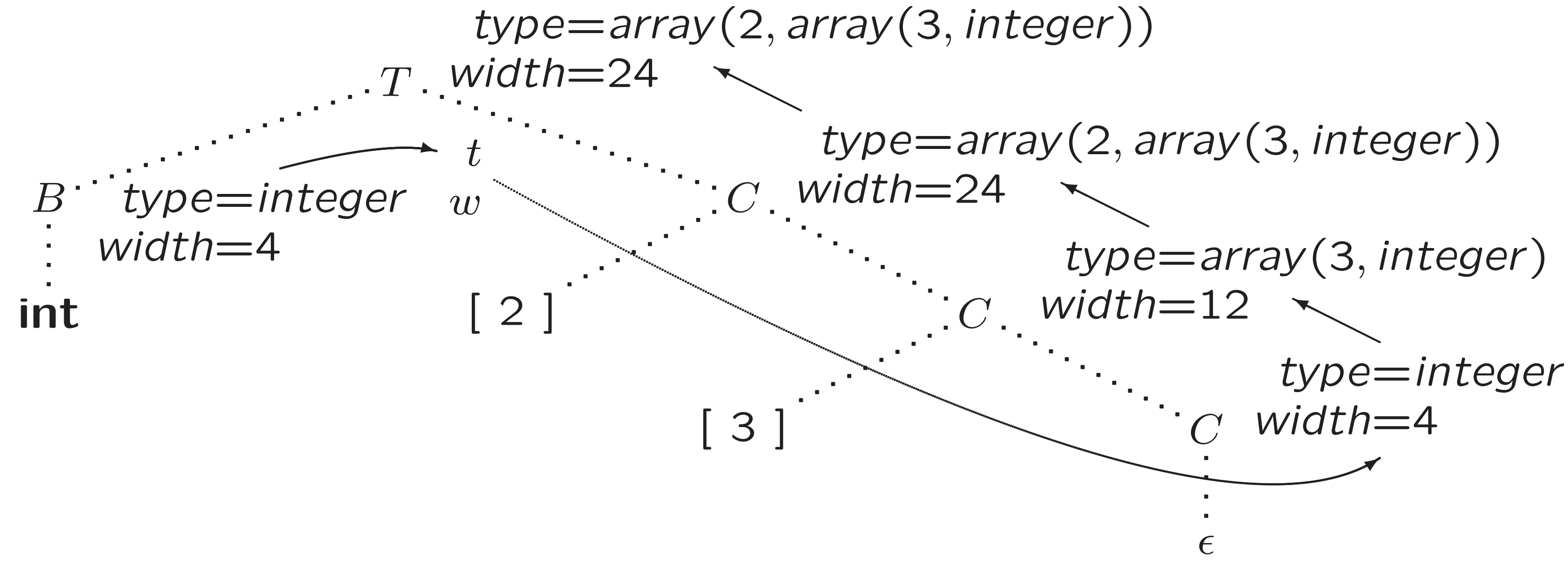
Parse tree for $c + a[i][j] \ldots$

Compare to ‘syntax tree’ for declaration type. . .

A slide from lecture 5:

Types and Their Widths (Example)

T	$\rightarrow$	B	$\{\ t = B.\mathit{type};\ \ w = B.\mathit{width};\ \}$
		C	$\{\ T.\mathit{type} = C.\mathit{type};\ \ T.\mathit{width} = C.\mathit{width};\ \}$
B	$\rightarrow$	**int**	$\{\ B.\mathit{type} = \mathit{integer};\ \ B.\mathit{width} = 4;\ \}$
B	$\rightarrow$	**float**	$\{\ B.\mathit{type} = \mathit{float};\ \ B.\mathit{width} = 8;\ \}$
C	$\rightarrow$	ϵ	$\{\ C.\mathit{type} = t;\ \ C.\mathit{width} = w;\ \}$
C	$\rightarrow$	[**num**] C_1	$\{\ C.\mathit{type} = \mathit{array}(\mathbf{num}.\mathit{value},\ C_1.\mathit{type});$ $C.\mathit{width} = \mathbf{num}.\mathit{value} \times C_1.\mathit{width};\ \}$



Translation of Array References

Three synthesized attributes

- L.*addr*: temporary used to compute location in array
- L.*array*: pointer to symbol-table entry for array name
  - L.*array*.*base*: base address of array
- L.*type*: type of subarray generated by L (‘what must we multiply index by’)
  - For type t: t.*width*
  - For array type t: t.*elem*

Translation of Array References

$$
\begin{array}{lcll}
S & \rightarrow & \textbf{id} = E; & \{\ \textit{gen}(\textit{top.get}(\textbf{id}.\textit{lexeme})\ '=' \ E.\textit{addr}); \} \\
S & \rightarrow & L = E; & \{\ \textit{gen}(L.\textit{array.base}\ '['L.\textit{addr}\ ']'\ '='\ E.\textit{addr}); \} \\
E & \rightarrow & E_1 + E_2 & \{\ E.\textit{addr} = \textbf{new}\ \textit{Temp}(); \\
 & & & \quad \textit{gen}(E.\textit{addr}\ '='\ E_1.\textit{addr}\ '+'\ E_2.\textit{addr}); \} \\
E & \rightarrow & \textbf{id} & \{\ E.\textit{addr} = \textit{top.get}(\textbf{id}.\textit{lexeme}); \} \\
E & \rightarrow & L & \{\ E.\textit{addr} = \textbf{new}\ \textit{Temp}(); \\
 & & & \quad \textit{gen}(E.\textit{addr}\ '='\ L.\textit{array.base}\ '['L.\textit{addr}\ ']'); \} \\
L & \rightarrow & \textbf{id}\ [E] & \{\ L.\textit{array}\ = \textit{top.get}(\textbf{id}.\textit{lexeme}); \\
 & & & \quad L.\textit{type}\ = L.\textit{array.type.elem}; \\
 & & & \quad L.\textit{addr}\ = \textbf{new}\ \textit{Temp}(); \\
 & & & \quad \textit{gen}(L.\textit{addr}\ '='\ E.\textit{addr}\ '*'\ L.\textit{type.width}); \} \\
L & \rightarrow & L_1[E] & \{\ L.\textit{array}\ = L_1.\textit{array}; \\
 & & & \quad L.\textit{type}\ = L_1.\textit{type.elem}; \\
 & & & \quad t = \textbf{new}\ \textit{Temp}(); \\
 & & & \quad L.\textit{addr}\ = \textbf{new}\ \textit{Temp}(); \\
 & & & \quad \textit{gen}(t\ '='\ E.\textit{addr}\ '*'\ L.\textit{type.width}); \\
 & & & \quad \textit{gen}(L.\textit{addr}\ '='\ L_1.\textit{addr}\ '+'\ t); \}
\end{array}
$$

Translation of Array References

$$\begin{array}{rcll}
S & \to & \mathbf{id} = E; & \{ \; \mathit{gen}(\mathit{top.get}(\mathbf{id}.\mathit{lexeme})\ '='\ E.\mathit{addr}); \} \\
S & \to & L = E; & \{ \; \mathit{gen}(L.\mathit{array.base}\ '['L.\mathit{addr}\ ']'\ '='\ E.\mathit{addr}); \} \\
E & \to & E_1 + E_2 & \{ \; E.\mathit{addr} = \mathbf{new}\ \ \mathit{Temp}(); \\
 & & & \quad \mathit{gen}(E.\mathit{addr}\ '='\ E_1.\mathit{addr}\ '+'\ E_2.\mathit{addr}); \} \\
E & \to & \mathbf{id} & \{ \; E.\mathit{addr} = \mathit{top.get}(\mathbf{id}.\mathit{lexeme}); \} \\
E_2 & \to & L & \{ \; E_2.\mathit{addr} = \mathbf{new}\ \ \mathit{Temp}(); \\
 & & & \quad \mathit{gen}(E_2.\mathit{addr}\ '='\ L.\mathit{array.base}\ '['L.\mathit{addr}\ ']'); \} \\
L_1 & \to & \mathbf{id}\ [E_4] & \{ \; L_1.\mathit{array}\ \ = \mathit{top.get}(\mathbf{id}.\mathit{lexeme}); \\
 & & & \quad L_1.\mathit{type}\ \ = L_1.\mathit{array.type.elem}; \\
 & & & \quad L_1.\mathit{addr}\ \ = \mathbf{new}\ \ \mathit{Temp}(); \\
 & & & \quad \mathit{gen}(L_1.\mathit{addr}\ '='\ E_4.\mathit{addr}\ '*'\ L_1.\mathit{type.width}); \} \\
L & \to & L_1[E_3] & \{ \; L.\mathit{array}\ \ = L_1.\mathit{array}; \\
 & & & \quad L.\mathit{type}\ \ = L_1.\mathit{type.elem}; \\
 & & & \quad t = \mathbf{new}\ \ \mathit{Temp}(); \\
 & & & \quad L.\mathit{addr}\ \ = \mathbf{new}\ \ \mathit{Temp}(); \\
 & & & \quad \mathit{gen}(t\ '='\ E_3.\mathit{addr}\ '*'\ L.\mathit{type.width}); \\
 & & & \quad \mathit{gen}(L.\mathit{addr}\ '='\ L_1.\mathit{addr}\ '+'\ t); \}
\end{array}$$

Translation of Array References (Example)

- Let a be 2×3 array of integers
- Let c, i and j be integers
- Ad hoc three-address code for `c + a[i][j]` ...
- Annotated parse tree for expression `c + a[i][j]`

Exercise 1

Komende week

- Woensdag 31 oktober, 11.00-12.45: practicum

- Donderdag 1 november: inleveren opdracht 2

- Vrijdag 2 november,
  11.00-12.45: hoorcollege + introductie opdracht 3
  13.30-. . . : werkcollege

- Inleveren 22 november

Compilerconstructie

college 6
Intermediate Code Generation 1

Chapters for reading:
6.intro, 6.2–6.2.3 (except indirect triples), 6.4