Lecture 15 — Compiling MiniJa cont.

- Today we will discuss compilation of some more d constructs (which are not included in MP8).
  - While statements
  - Short-circuit evaluation of boolean expressions
  - Arrays
  - Switch statements
  - V-tables and objects

A note on static vs. dynamic typ

- We switched from dynamic typing in MP7 to static typ MP8.
- What did we gain by switching to static typing?

  Efficiency - avoid run-time tag checking.

- What did we lose?

  Lots of programs that would have run fine but are now illegal, e.g. those that use heterogeneous lists.

Static vs. dynamic typing (cont

- Can we compile a dynamically-typed language?
  - Consider compilation scheme for $e_1 + e_2$

Assume values start with one-word tag (1 for int, etc

e₁ + e₂, loc ⇝ il₁ @ il₂ @ [LOADIMM temploc, 1; EQUAL temploc2, loc₁, te
CJUMP m, m′; ...]

e₁, loc₁ ⇝ il₁
e₂, loc₂ ⇝ il₂

code to check tags in loc1 and
and dispatch correct ope
or flag error.

  - Did we gain efficiency?

Yes, some. We avoid having to traverse and int
the AST.

Compilation schemes

Methods: $M \leadsto il$

Statements: $S,\, m \leadsto il,\, m'$

Expressions: $e,\, loc \leadsto il$

Compiling while statements

While (e) S, $m \rightsquigarrow$

$[\text{JUMP } m'] \text{ @ } ils \text{ @ } ile \text{ @ } [\text{CJUMP loc},m+1,m''],m''$

$S,\ m+1 \rightsquigarrow ils,\ m'$

e, loc $\rightsquigarrow ile$

(where $m'' = m' + |ile| + 1$)

do S while (e), $m \rightsquigarrow$ ils @ ile @ [CJUMP loc, m, m' + |ile| +

S, m $\rightsquigarrow$ ils , m'

e, loc $\rightsquigarrow$ ile

Evaluation of boolean expressio

- **MP7 uses strict evaluation of boolean expressions:**

if (e) S_1 else S_2, $m \leadsto$
$\quad il$ @ [CJUMP loc,$m + |il| + 1$, $m' + 1$] @ il_1 @ [JUMP m''] @ il_2,
$\quad e$, loc $\leadsto il$
$\quad S_1$, $m + |\textbf{il}| + 1 \leadsto il_1$, m'
$\quad S_2$, $m' + 1 \leadsto il_2$, m''

x=e, $m \leadsto il$ @ [MOV(addr x, loc)], $m + |il| + 1$
(x a variable)

OperationT(e_1,bop,e_2), loc $\leadsto il_1$ @ il_2 @ [BOP loc,loc1,loc2]
$\quad e_1$, loc1 $\leadsto il_1$
$\quad e_2$, loc2 $\leadsto il_2$

Evaluation of boolean expressio (cont.)

```
public int main (int m, int n) {
   if (m<n  & (m < 10 | 10 < n))
      n = 0;
   else
      n = 1;
   return n;
}
```

```
0:    LESS        3,1,2                CJUMP      9,8,11
      LOADIMM     4,10          8:     LOADIMM    3,0
      LESS        5,1,4                MOV        2,3
      LOADIMM     6,10                 JUMP       13
      LESS        7,6,2         11:    LOADIMM    3,1
      OR          8,5,7                MOV        2,3
      AND         9,3,8         13:    RETURN     2
```

Short-circuit evaluation of boole expressions

- The best way to compile boolean expressions is to computing the value of the expression.

$$e,\ m,\ t,\ f \rightsquigarrow_2 il,\ m'$$

- Some expressions are compiled very simply:

True, $m,\ t,\ f \rightsquigarrow_{sc}$ [JUMP t], $m+1$

False, $m,\ t,\ f \rightsquigarrow_{sc}$ [JUMP f], $m+1$

!$e,\ m,\ t,\ f \rightsquigarrow_{sc} il,\ m'$
$\qquad e,\ m,\ f,\ t \rightsquigarrow_{sc} il,\ m'$

Short-circuit evaluation of boole expressions (cont.)

$e_1 \&\& e_2,\ m,\ t,\ f \rightsquigarrow_{sc}$ il_1 @ il_2 , m''

$e_1, m, m', f \rightsquigarrow_{sc} il_1, m'$

$e_2, m', t, f \rightsquigarrow_{sc} il_2, m''$

$e_1 || e_2,\ m,\ t,\ f \rightsquigarrow_{sc}$ il_1 @ il_2 , m''

$e_1, m, t, m' \rightsquigarrow_{sc} il_1, m'$

$e_2, m', t, f \rightsquigarrow_{sc} il_2, m''$

If (e) S_1 else S_2, $m \rightsquigarrow$ il_e @ il_1 @ [JUMP m'''] @ il_2 , m'''

$e, m, m', m'' \rightsquigarrow_{sc} il_e, m'$

$S_1, m' \rightsquigarrow il_1, m''$

$S_2, m''+1 \rightsquigarrow il_2, m'''$

Short-circuit evaluation of boole expressions (cont.)

```
public int main (int m, int n) {
   if (m<n  & (m < 10 | 10 < n))
      n = 0;
   else
      n = 1;
   return n;
}
```

```
0:   LESS      3,1,2              CJUMP     7,8,11
     CJUMP     3,2,11       8:    LOADIMM   3,0
2:   LOADIMM   4,10               MOV       2,3
     LESS      5,1,4              JUMP      13
     CJUMP     5,8,5        11:   LOADIMM   3,1
5:   LOADIMM   6,10               MOV       2,3
     LESS      7,6,2        13:   RETURN    2
```

Arrays in MJ

- Arrays stored in the heap. Contents are integers — senting integers, bools, or pointers to heap objects (inc arrays).
- Have instruction (not used in MP8):

```
ARRAYREF tgt,src,indx: (p, c, s, h, t, r)
                             -> (p+1, c, s[i/tgt], h, t, r)
```

- Array indexing:

$a[e]$, **loc** $\leadsto$ il @ ARRAYREF loc, addr(a), loc,

e, loc, $\leadsto$ il

Multi-dimensional arrays in M

- **A multi-dimensional array is an array that contains po to other arrays.**

- **Array indexing for multi-dimensional arrays:**

$e_1[e_2]$, **loc** $\rightsquigarrow$ il_1 @ il_2 @ ARRAYREF loc, loc_1, loc_2

$e_1, loc_1 \rightsquigarrow il_1$

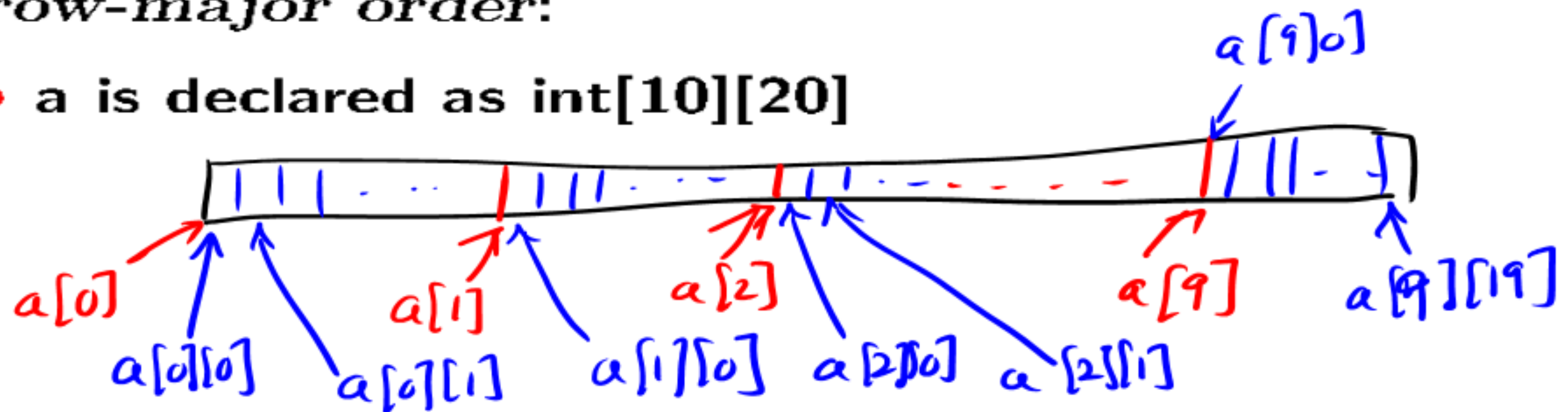
$e_2, loc_2 \rightsquigarrow il_2$

Arrays in C

- **Arrays are addresses:** `a[i]` ≡ `a + i` **(where** `i` **is mul**
  by the size of a's elements)

- **Multi-dimensional arrays always rectangular, and arran**
  ***row-major order*:**

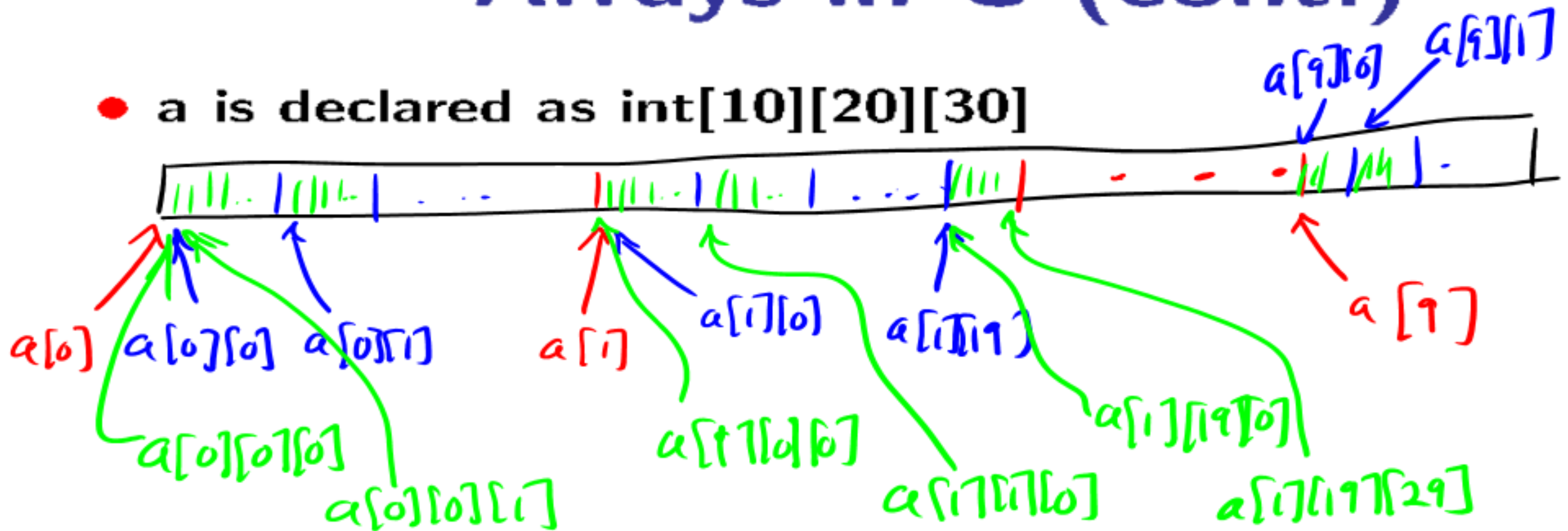
  - **a is declared as int[10][20]**



  - **address of a[i][j] = a + i*80 + j*4.**

Arrays in C (cont.)

- a is declared as int[10][20][30]



- address of a[i][j][k] = a + i*2400 + j*120 + k*4

- **Rule is:** address of $e_1[e_2]$ = address of e_1 + (e_2 * (
  elements of e_1))

Array assignment in C

- With arrays, left-hand sides of assignment statemen
  be complex expressions.

- A compilation scheme like this one makes no sense:

$$e_1[e_2] = e_3,\ m \rightsquigarrow il1 \ \textbf{@}\ il2 \ \textbf{@}\ [\textbf{MOV loc1,loc2}]$$
$$e_1[e_2],\ \textbf{loc1} \rightsquigarrow il1$$
$$e_3,\ \textbf{loc2} \rightsquigarrow il2$$

- Consider `a[i] = a[i]`. Can't evaluate both occurren
  `a[i]` to the same value!

Array assignment in C (cont.)

- "l-values" vs. "r-values"
  - l-values are values of expressions on left-hand si
    assignments. They are *addresses*.
- Need scheme for calculating l-values.
- Compilation of assignment becomes:

$$e_1 = e_2,\ m \leadsto il1\ \textbf{@}\ il2\ \textbf{@}\ [\textbf{MOVIND loc1,loc2}]$$
$$e_1,\ \textbf{loc1} \leadsto_{lval} il1$$
$$e_2,\ \textbf{loc2} \leadsto il2$$

Switch statements

- Switch statements can be compiled two ways:
  - As cascade of if statements.
  - As "jump table" — array of locations of the code fo[r each] case; use switch expression as index.
- When should you use one or the other?

If there are only a small number of cases, cascadin[g] if's may be more efficient.

If cases are very sparse (case 0: ... case 100000 ...) jump table will take up more memory than the time savings are worth.

Objects

- **Fields: How is inheritance of fields handled in our con**

  When object is created, all fields declared in its class o
  its class's ancestors are gathered together, and memo
  large enough for all of them is allocated. For any field f o
  any class C, f's offset is the same, whether it is in a C object
  a descendant class.

- **Methods: How is inheritance, and overriding, of me handled in our compiler?**

  We created a table giving for any method name an
  class name, the appropriate method definition (taking
  inheritance and overriding into account). This table is
  built statically. At run time, we know the class of the
  receiver and the name of the method, and we look them
  up in the table.

V-tables

- **There is one place in our code where a name appears compiled code: when compiling "`new C()`".**
- **Why is it needed? (Hint: not to determine the size object to be allocated.)**

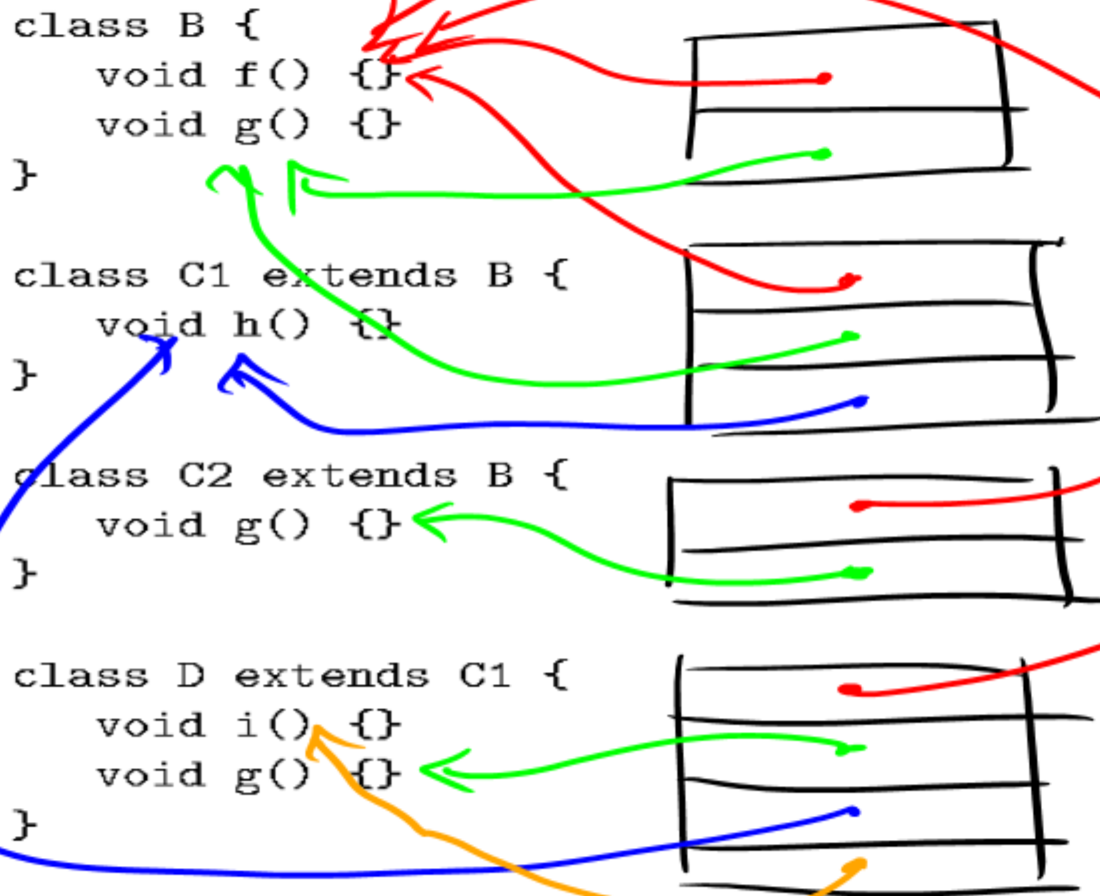
  To index the table just described.

- **How can it be eliminated?**

  Use v-tables.

V-tables (cont.)

- Draw a table for each class, listing all the methods bel
  to that class (including inherited ones). The order sho
  from top of the hierarchy to the bottom.



Wrap-up

- Today we discussed compilation of:
  - While statements
  - Boolean expressions (using short-circuit evaluation)
  - Arrays
  - Objects and inheritance
- We discussed it because:
  - These include most of the constructs you will see in most progra
languages.
- What to do now:
  - Finish MP8