Lecture 15 — Compiling MiniJava, cont.

- Today we will discuss compilation of some more difficult 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 typing

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

Static vs. dynamic typing (cont.)

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

Compilation schemes

Methods: $M \rightsquigarrow il$

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

Expressions: e, $loc \rightsquigarrow il$

Compiling while statements

While (e) S, $m \rightsquigarrow$

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

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

e, loc $\rightsquigarrow ile$

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

do S while (e), $m \rightsquigarrow$

S,

e,

Evaluation of boolean expressions

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

if (e**)** S_1 **else** S_2, $m \rightsquigarrow$
il **@ [CJUMP loc,**$m + |il| + 1$, $m' + 1$**] @** il_1 **@ [JUMP** m''**] @** il_2, m''
e, **loc** $\rightsquigarrow il$
S_1, $m + |$**il**$| + 1 \rightsquigarrow il_1$, m'
S_2, $m' + 1 \rightsquigarrow il_2$, m''

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

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

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

Short-circuit evaluation of boolean expressions

- **The best way to compile boolean expressions is to avoid 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'

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

Short-circuit evaluation of boolean expressions (cont.)

$e_1 \&\& e_2$, m, t, f $\rightsquigarrow_{sc}$

$e_1 || e_2$, m, t, f $\rightsquigarrow_{sc}$

If (e) S_1 else S_2, m $\rightsquigarrow$

Short-circuit evaluation of boolean 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     3,2,11
2:   LOADIMM   4,10
     LESS      5,1,4
     CJUMP     5,8,5
5:   LOADIMM   6,10
     LESS      7,6,2
     CJUMP     7,8,11
8:   LOADIMM   3,0
     MOV       2,3
     JUMP      13
11:  LOADIMM   3,1
     MOV       2,3
13:  RETURN    2
```

Arrays in MJ

- **Arrays stored in the heap. Contents are integers — representing integers, bools, or pointers to heap objects (including 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** $\rightsquigarrow$

Multi-dimensional arrays in MJ

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

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

  $e_1[e_2]$**, loc** $\leadsto$

Arrays in C

- **Arrays are addresses:** `a[i]` $\equiv$ `a + i` **(where** `i` **is multiplied by the size of a's elements)**
- **Multi-dimensional arrays always rectangular, and arranged in** ***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 * (size of elements of e_1))

Array assignment in C

- **With arrays, left-hand sides of assignment statements can be complex expressions.**

- **A compilation scheme like this one makes no sense:**

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

- **Consider `a[i] = a[i]`. Can't evaluate both occurrences of `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 sides of assignments. They are** $addresses$**.**
- **Need scheme for calculating l-values.**
- **Compilation of assignment becomes:**

$e_1 = e_2$, $m \leadsto il1$ **@** $il2$ **@ [MOVIND loc1,loc2]**

e_1, **loc1** $\leadsto_{lval}$ $il1$

e_2, **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 for each case; use switch expression as index.**
- **When should you use one or the other?**

Objects

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

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

V-tables

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

V-tables (cont.)

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

```
class B {
   void f() {}
   void g() {}
}

class C1 extends B {
   void h() {}
}

class C2 extends B {
   void g() {}
}

class D extends C1 {
   void i() {}
   void g() {}
}
```

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 programming languages.**
- **What to do now:**
  - **Finish MP8**