Scope and Memory Management (part 2)

CSCI 334
Stephen Freund

Inline Blocks

```
{ int x = 2;
 int y = 10
 { int z = 2;
   int x = 3;
   x = z + y;
 } print x;
}
```

Control link

```
<table>
<thead>
<tr>
<th>Environment Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>x 2</td>
</tr>
<tr>
<td>y 10</td>
</tr>
</tbody>
</table>
```

Accessing Globals

```
val m = 5;
fun force(a) = m * a;
fun cow(y) =
let m = y * y in
  force(m)
end;
cow(10);

force(100)
```

Dynamic Scope: force(100)
follow control links

```
<table>
<thead>
<tr>
<th>Static Scope: force(100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>how to find m? # links to follow?</td>
</tr>
</tbody>
</table>
```

Accessing Globals

```
val m = 5;
fun force(a) = m * a;
fun cow(y) =
let m = y * y in
  force(m)
end;
cow(10);

moo(10);
moo(10);
```

Static Scope: force(100)
Now how many???

```
<table>
<thead>
<tr>
<th>Static Scope: force(100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>how to find m? # links to follow?</td>
</tr>
</tbody>
</table>
```

Static Scope:

```
val m = 5;
fun force(a) = m * a;
fun cow(y) =
let m = y * y in
  force(m)
end;
cow(10);

moo(10);
moo(10);
```

Now how many???
**Accessing Globals**

```plaintext
val m = 5;
fun force(a) = m * a;
fun cow(y) =
  let m = y * y in
  force(m)
  end;
cow(10);
```

- **Access link**: link to activation record for enclosing scope

**Activation record for static scope**

- Control link: link to activation record of previous (calling) block
- Access link: link to activation record of closest enclosing block in program text
- **Difference**
  - Control link depends on dynamic behavior of program
  - Access link depends on static form of program text

**Another Example**

```plaintext
val cm = 2.54;
fun toCM(y) = cm * y;
...
toCM(5.0);
```

**Passing Functions to Functions**

```plaintext
val cm = 2.54;
fun toCM(y) = cm * y;
fun map(h,nil) = nil
  | map(h,x::xs) = h(x)::map(h,xs);
map(toCM,[1.0,2.0]);
```

**Closures**

```plaintext
val cm = 2.54;
fun toCM(y) = cm * y;
fun map(h,nil) = nil
  | map(h,x::xs) = h(x)::map(h,xs);
map(toCM,[1.0,2.0]);
toCM(1.0)
```

**makeRand**

```plaintext
fun makeRand(seed1, seed2) =
  let val generator = Random.rand(seed1,seed2);
  fun rand(lo, hi) =
    Random.randRange(lo,hi)(generator)
  in
  rand
  end;
val gen = makeRand(10,12);
val x = gen(0,10);
```
fun make(seed) = let fun rand(lo) = lo + seed in rand end;

val gen = make(0);
gen(5) + gen(4);

(make (not so random...)

seed is free var

Function Results and Closures

Right before executing "lo + seed" in gen(5)....)