Scope and Memory Management (part 2)

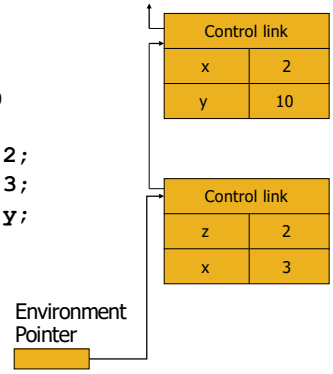
CSCI 334

Stephen Freund

24

Inline Blocks

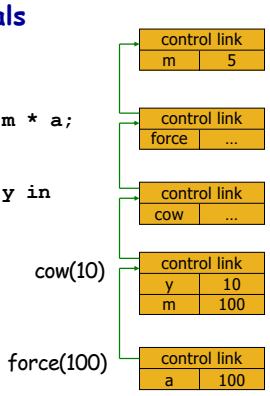
```
{
  int x = 2;
  int y = 10
  {
    int z = 2;
    int x = 3;
    x = z + y;
  }
  print x;
}
```



25

Accessing Globals

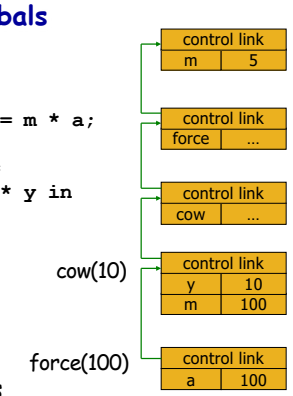
```
val m = 5;

fun force(a) = m * a;

fun cow(y) =
  let m = y * y in
    force(m)
  end;

cow(10);
```



26

Accessing Globals

```
val m = 5;

fun force(a) = m * a;

fun cow(y) =
  let m = y * y in
    force(m)
  end;

cow(10);
```

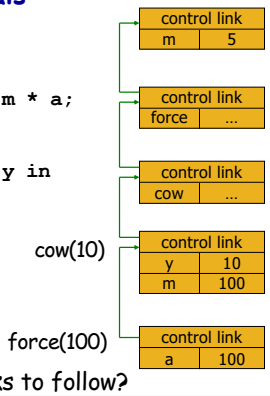


Dynamic Scope:
follow control links

27

Accessing Globals

```
val m = 5;

fun force(a) = m * a;

fun cow(y) =
  let m = y * y in
    force(m)
  end;

cow(10);
```

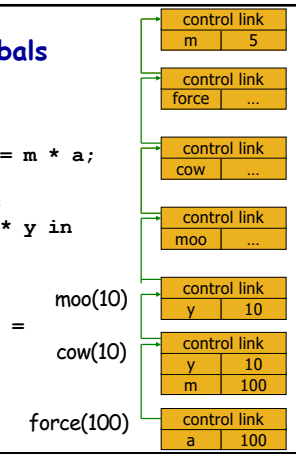


Static Scope:
how to find m? # links to follow?

28

Accessing Globals

```
val m = 5;

fun force(a) = m * a;

fun cow(y) =
  let m = y * y in
    force(m)
  end;

fun moo(y) =
  cow(y);

moo(10);
```

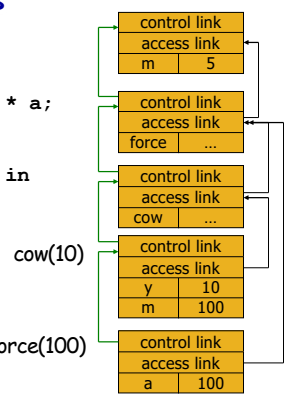


Static Scope:
Now how many???

29

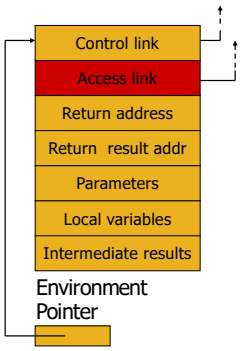
Accessing Globals

```
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

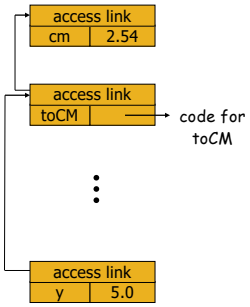
Activation record for static scope



- 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 prog
  - Access link depends on static form of program text

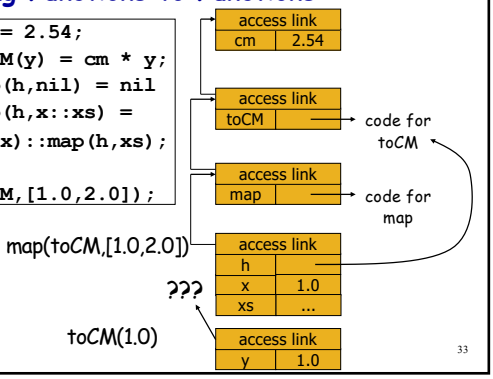
Another Example

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

...

toCM(5.0);
```



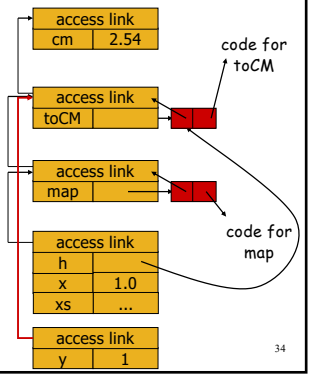
Passing Functions to Functions

```
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

```
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]);
```



makeRand

```
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);
```

generator is free var

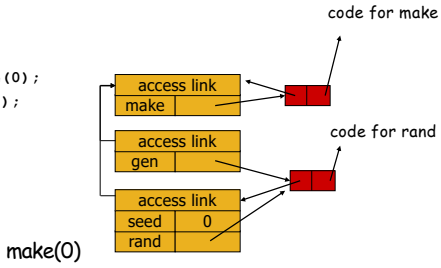
make (not so random...)

```
fun make(seed) =
  let fun rand(lo) = lo + seed
  in
    rand
  end;

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

seed
is free var

Function Results and Closures

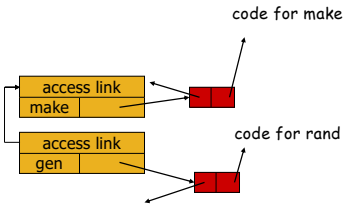
```
fun make(seed) =
  let fun rand(lo) = lo + seed
  in
     rand
  end;

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



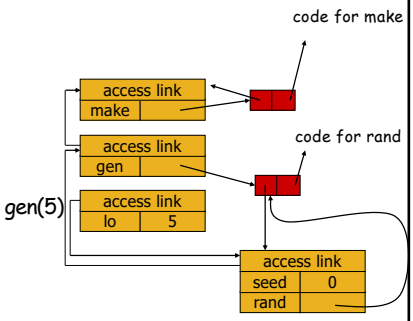
make(0)

Function Results and Closures

```
fun make(seed) =
  let fun rand(lo) = lo + seed
  in
     rand
  end;

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




```
fun make(seed) =
  let fun rand(lo) = lo + seed
  in
     next
  end;

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



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