# Scope and Memory Management (part 2)

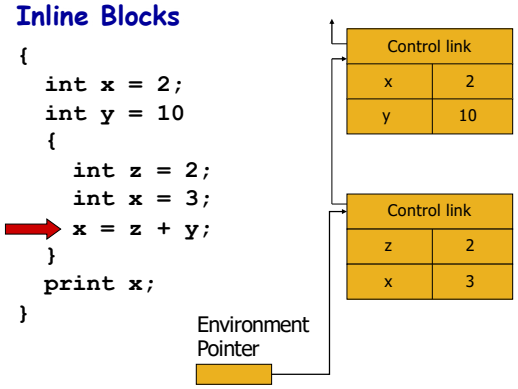
CSCI 334

Stephen Freund



## Inline Blocks

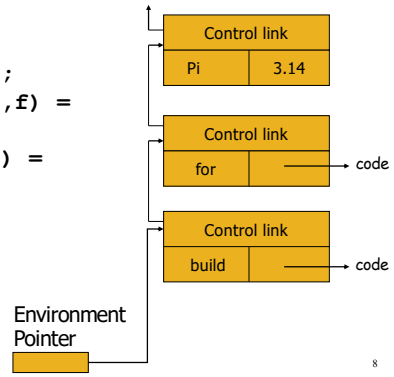
```
{
  int x = 2;
  int y = 10
  {
    int z = 2;
    int x = 3;
    x = z + y;
  }
  print x;
}
```






## Declarations

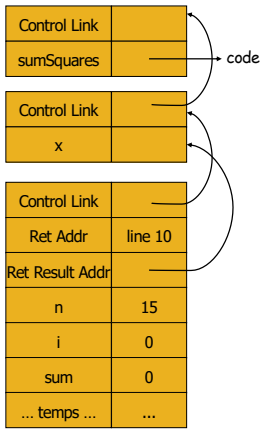
```
val Pi = 3.14;
fun for(lo,hi,f) =
  ...
fun build(...) =
  ...
```



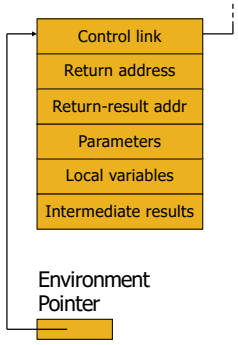



## Function Calls

```
1   int sumSquares(int n) {
2     int i, sum = 0;
3     for (i = 0; i < n; i++)
4       sum = sum + i * i;
5     return sum;
6   }
7   ...
8   {
9     int x = sumSquares(15);
10    print x;
11  }
```

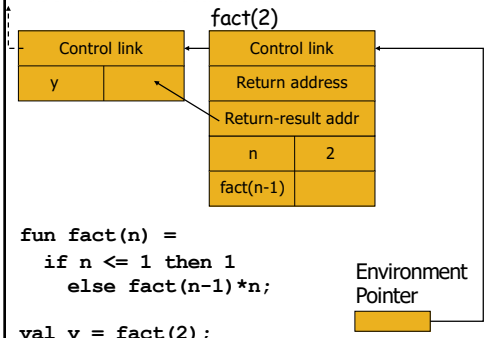



## Activation Record



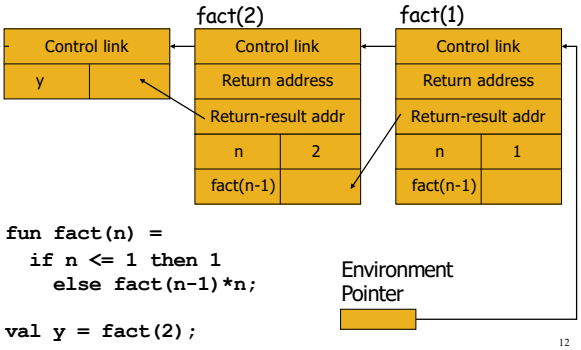



## Activation Record




```
fun fact(n) =
  if n <= 1 then 1
    else fact(n-1)*n;

val y = fact(2);
```

## Activation Record

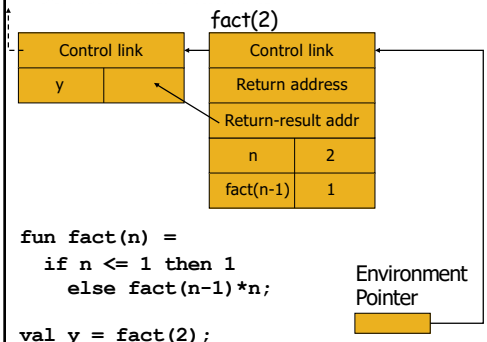



```
fun fact(n) =
  if n <= 1 then 1
    else fact(n-1)*n;

val y = fact(2);
```

Environment Pointer

12

## Activation Record

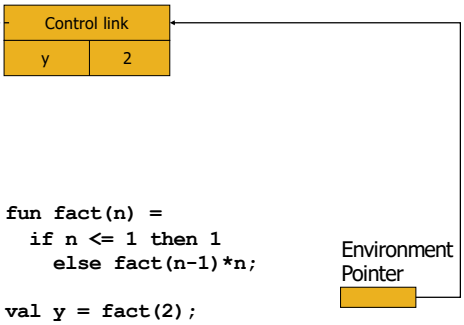



```
fun fact(n) =
  if n <= 1 then 1
    else fact(n-1)*n;

val y = fact(2);
```

Environment Pointer

13

## Activation Record




```
fun fact(n) =
  if n <= 1 then 1
    else fact(n-1)*n;

val y = fact(2);
```

Environment Pointer

14

## Parameter Passing

```
fun swap(int x, int y) {
  int t = x;
  x = y;
  y = t;
}

{
  int a = 10;
  int b = 20;
  swap(a,b);
  print(a);
}
```

15

## Parameter Passing: By Value

```
fun swap(int x, int y) {
  int t = x;
  x = y;
  y = t;
}

{
  int a = 10;
  int b = 20;
  swap(a,b);
  print(a);
}
```

| a | 10 |
|---|---|
| b | 20 |

| x | ~~10~~ 20 |
|---|---|
| y | ~~20~~ 10 |
| t | 10 |

16

## Parameter Passing: By Reference

```
fun swap(int x, int y) {
  int t = x;
  x = y;
  y = t;
}

{
  int a = 10;
  int b = 20;
  swap(a,b);
  print(a);
}
```

| a | ~~10~~ 20 |
|---|---|
| b | ~~20~~ 10 |

| x | |
|---|---|
| y | |
| t | 10 |

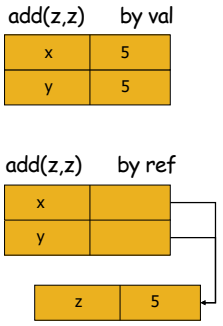
17

## Why Does it Matter?

- Side Effects
- Aliasing

```
int add(x, y) {
  x = x + 1;
  return x + y;
}
z = 5;
print add(z, z);
```

- Efficiency




18

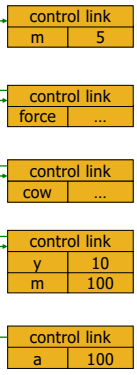
## Accessing Globals

```
val m = 5;

fun force(a) = m * a;

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

cow(10);
```

19

## Accessing Globals

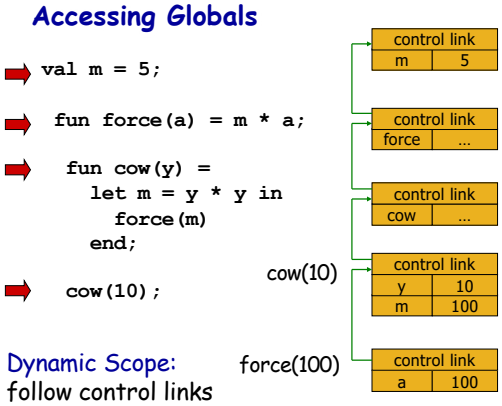
```
val m = 5;

fun force(a) = m * a;

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

cow(10);
```




20

## Accessing Globals

```
val m = 5;

fun force(a) = m * a;

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

cow(10);
```

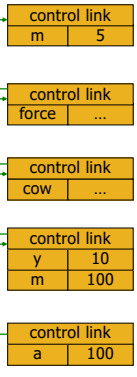



Dynamic Scope:
follow control links

21

## Accessing Globals

```
val m = 5;

fun force(a) = m * a;

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

cow(10);
```

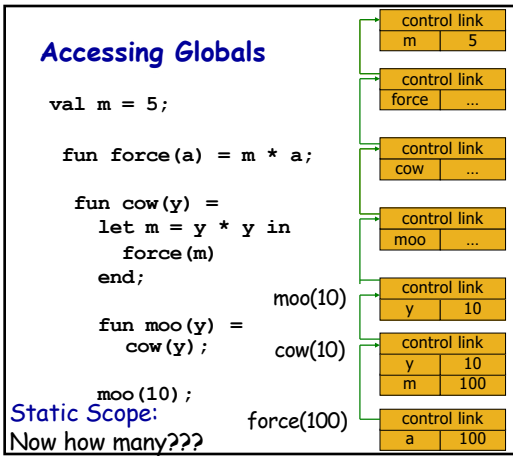



Static Scope:
how to find m? # links to follow?

22

## 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);
```

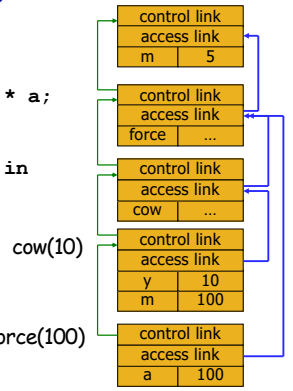



Static Scope:
Now how many???

23

## Accessing Globals

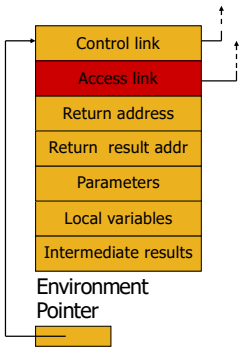
```
val m = 5;

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

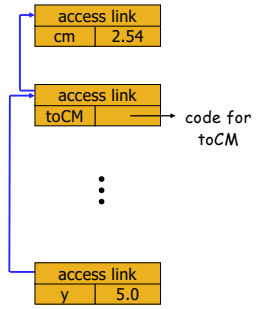



- Control link
  - link to activation record of previous (calling) block
- Access link
  - link to activation record of closest enclosing block in program text
- Difference
  - Control link depends on dynamic behavior of prog
  - Access link depends on static form of program text



## Another Example

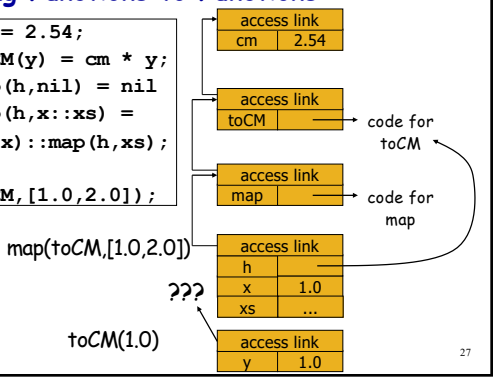
```
val cm = 2.54;
fun toCM(y) = cm * y;

...

toCM(5.0);
```






## Passing Functions to Functions

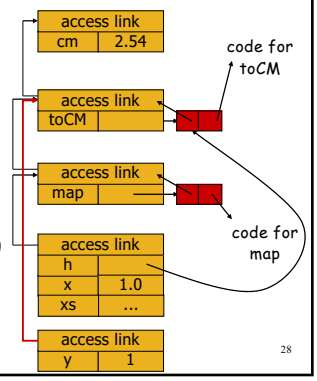
```
val cm = 2.54;
fun toCM(y) = cm * y;
fun map(h,nil) = nil
  | map(h,x::xs) =
      h(x)::map(h,xs);

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...)

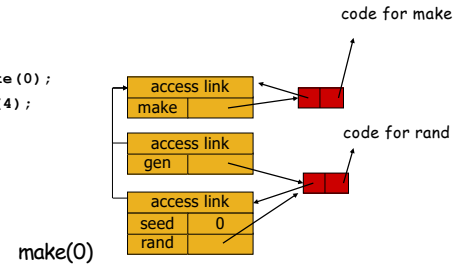
```
fun make(seed) =
  let fun rand(lo) = lo + seed
  in
    rand
  end;

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);
```

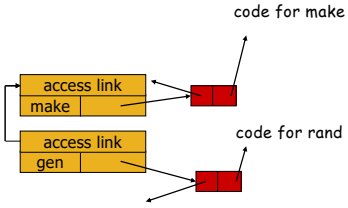



make(0)



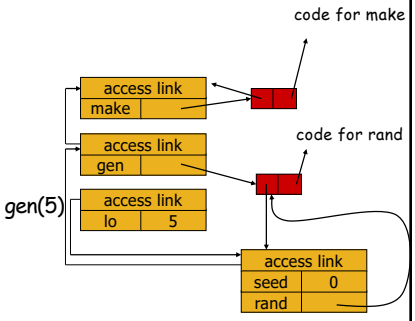
## Function Results and Closures

```
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)....)



## Tail Recursion

```
fun sumSq n =
  if n <= 0
  then 0
  else n*n + sumSq(n-1);
```

```
sumSq(2)
 sumSq(1)
  sumSq(0)
  return 0
 return 1+0
return 4+1
```

```
fun sumSqTail(n, acc) =
  if acc <= 0
  then acc
  else sumSqTail(n-1,acc + n*n);
```

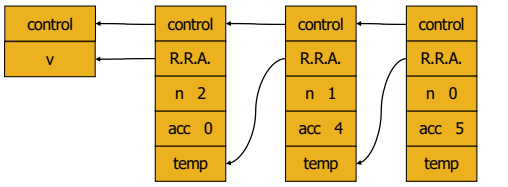
```
sumSqTail(2,0)
 sumSqTail(1,4)
  sumSqTail(0,5)
  return 5
 return 5
return 5
```

```
fun sumSq n = sumSqTail(n, 0)
```



## v = sumSqTail(2,0)

```
fun sumSqTail(n, acc) =
  if acc <= 0
  then acc
  else sumSqTail(n-1,acc + n*n);
```



## v = sumSqTail(2,0)

same test

args to recursive call

base case

```
fun sumSqTail(n, acc) {
while not (n <= 0) {
  acc = acc + n * n;
  n = n - 1;
}
return acc
}
```