Scope and Memory Management (part 2)

CSCI 334
Stephen Freund

Inline Blocks

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

Parameter Passing Modes

```c
void swap(int a, int b) {
  int t = a;
  a = b;
  b = t;
}
```

Accessing Globals

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

Why Does it Matter?

- Side Effects
- Aliasing
  ```c
  int add(x, y) {
    x = x + 1;
    return x + y;
  }
  z = 5;
  print add(z, z);
  ```
- Efficiency
  ```c
  add(z, z) by val
  
  add(z, z) by ref
  ```
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: force(100)
follow control links

Examples of Dynamic Scoping

fun formatBuffer(buffer) =
  ... setColor(highlightColor) ...
let highlightColor = Blue in
  formatBuffer(b);

fun playGame() =
  ... if strategy(...) = goLeft then ...
let fun strategy(...) = ...
in playGame();

Stack Inspection

• Permission depends on:
  - permission of calling method
  - permission of all methods above it on stack

void open(String s) {
  SecurityManager.checkRead();
  ...
}

Stack Inspection

• Permission depends on:
  - permission of calling method
  - permission of all methods above it on stack

void open(String s) {
  SecurityManager.checkRead();
  ...
}

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: force(100)
how to find m? # links to follow?

Accessing Globals

val m = 5;
fun force(a) = m * a;
fun cow(y) = let m = y * y in force(m) end;
funkoo(y) = cow(y);
moo(10);
Static Scope: force(100)
Now how many???
Accessing Globals

```ml
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
- Force link: points 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

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

Passing Functions to Functions

```ml
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

```ml
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]);```

Function Values are Closures

```ml
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]);```
Summary of Function Arguments

• Closure maintains pointer to static environment of a function body

• When called, access link set from closure

• All access links point "up" in the stack – can still deallocate activation records in lifo order

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);

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 (not so random...)

fun make(seed) = let fun rand(lo) = lo + seed in rand end;
val gen = make(0); gen(5) + gen(4);

Function Results and Closures

fun make(seed) = let fun rand(lo) = lo + seed in rand end;
val gen = make(0); gen(5) + gen(4);

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

make(seed) =
let fun rand(lo) = lo + seed in
next end;
val gen = make(0); gen(5) + gen(4);