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  }

Val fact = fact(2)

Activation Record

fun fact(n) =
  if n <= 1 then 1
  else fact(n-1)*n;
val y = fact(2);
fun fact(n) = if n <= 1 then 1 else fact(n-1)*n;
val y = fact(2);
Why Does it Matter?

- Side Effects
- Aliasing
  ```
  int add(x, y) {
    x = x + 1;
    return x + y;
  }
  ```
  add(z, z) by val
  ```
  z = 5;
  print add(z, z);
  ```
  add(z, z) by ref
- Efficiency

Accessing Globals

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

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

```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]);
toCM(1.0)
```

makeRand

```ml
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);
```

makeRand 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);

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)....)

Tail Recursion

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

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

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);
\[ v = \text{sumSqTail}(2, 0) \]

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