

CMSC 330, Spring 2009, Midterm 2 Practice Problems

1. Programming languages

- Describe the difference between OCaml modules and Java classes.
- Describe the difference between strong and weak typing.
- Explain how call-by-name simplifies implementing lazy evaluation.
- Describe the difference between an L-value and an R-value.
- What is an activation record, and why is it usually allocated on a stack?
- Describe short circuiting.

2. Function arguments

For each code, explain whether g is an upward or downward funarg.

- `let f x = let g y = x + y in let app a b = a b in app g 1 ;;`
- `let f x = let g y = x + y in g ;;`

3. Static vs. Dynamic Scoping

Consider the following OCaml code.

```
let a = 1 ;;  
let f = fun () -> a ;;  
let a = 2 ;;  
f ();;
```

- What value is returned by the invocation of $f()$ with static scoping? Explain.
- What value is returned by the invocation of $f()$ with dynamic scoping? Explain.

Consider the following OCaml code.

```
let app f w = let x = 1 in f w ;;  
let add x y = let incr z = z+x in app incr y ;;  
(add 2 3) ;;
```

- What is the order of invocation for the functions `app`, `add`, and `incr` when evaluating the expression `(add 2 3)`?
- What value is returned by `(add 2 3)` with static scoping? Explain.
- What value is returned by `(add 2 3)` with dynamic scoping? Explain.

4. Parameter passing

Consider the following C code.

```
int i = 2;
void foo(int f, int g) {
    f = f-i;
    g = f;
}
int main( ) {
    int a[] = {2, 0, 1};
    foo(i, a[i]);
    printf("%d %d %d %d\n", i, a[0], a[1], a[2]);
}
```

- Give the output if C uses call-by-value
- Give the output if C uses call-by-reference
- Give the output if C uses call-by-name

5. Lazy evaluation

Given the following OCaml code.

```
let doIf p x = if p then x else 0 ;;
let rec loop n = loop n ;;
doIf false (loop 0) ;;
```

- What is the result of evaluating the doIf expression if OCaml uses call-by-value?
- What is the result of evaluating the doIf expression if OCaml uses call-by-name?
- Rewrite the code (using thunks) so that the result of evaluating the doIf expression is the same as if OCaml used call-by-name, even though OCaml uses call-by-value.

6. Tail recursion

For each OCaml function, explain why it is or is not tail recursive

- let rec foo x = 1 + (foo x)
- let rec sum l = match l with
[] -> 0
| (x::xs) -> x + (sum xs)
- let rec last = function
[x] -> x
| (_::xs) -> last xs
- let rec fib x =
if (x = 0) then 0
else if (x = 1) then 1
else (fib (x-1) + fib (x-2))