

CMSC330 Fall 2013 Practice Problems 6

1. Programming languages

- Describe how functional programming may be used to simulate OOP.
- 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 (frame), and why is it usually allocated on a stack?
- Describe the difference between ad-hoc and parametric polymorphism.

2. Function arguments

For each code, explain whether *g* is an upwards 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. Garbage collection

Consider the following Java code.

```
Object a, b, c;
public foo( ) {
    a = new Object( );    // object 1
    b = new Object( );    // object 2
    c = new Object( );    // object 3
    a = b;
    b = c;
    c = a;
}
```

- What object(s) are garbage when foo() returns? Explain why.
- Describe the difference between mark-and-sweep & stop-and-copy.

7. Polymorphism

Consider the following Java classes:

```
class A { public void a() { ... } }  
class B extends A { public void b() { ... } }  
class C extends B { public void c() { ... } }
```

Explain why the following code is or is not legal

- a. `int count(Set<A> s) { ... } ... count(new Set<A>);`
- b. `int count(Set<A> s) { ... } ... count(new Set<B>);`
- c. `int count(Set s) { ... } ... count(new Set<A>);`
- d. `int count(Set<?> s) { ... } ... count(new Set<A>);`
- e. `int count(Set<? extends A> s) { ... } ... count(new Set<B>);`
- f. `int count(Set<? extends B> s) { ... } ... count(new Set<A>);`
- g. `int count(Set<? extends B> s) { for (A x : s) x.a(); }`
- h. `int count(Set<? extends B> s) { for (C x : s) x.c(); }`
- i. `int count(Set<? super B> s) { for (A x : s) x.a(); }`
- j. `int count(Set<? super B> s) { for (C x : s) x.c(); }`