CS558 Programming Languages – Fall 2023 – Study Questions Lecture 4b

These questions are intended for self-study, to help review and deepen your understanding of the lecture. Sample answers are available. There is nothing to hand in.

1. Suppose a C compiler uses a stack to store activation data in the style of slide 4, and consider the following program. (The line numbers on the left are not part of the program.)

```
01 int g(int s, int t) {
02   int p = s * 2;
03   return p+1;
04 }
05 int f(int r) {
06   r = r + 1;
07   int q = r;
08   q = g(q, r + 7);
09   return q+1;
10 }
11 void main() {
12   int a = 2;
13   int b = 20;
14   b = f(a);
15 }
```

For each of the following program points, what local variables and parameters are on the stack and what are their values?

(i) Beginning of line 14.

(ii) Beginning of line 08.

(iii) Beginning of line 03.

(iv) Beginning of line 09.

(v) Beginning of line 15.

2. Consider the following code, written in Scala-like syntax:

```
case class P(var a:Int)
// see note 1 below

def twiddle(var x:P,var y:P) = {
// see note 2 below
 val z = x
 x = y
 y = z
}

def swizzle(x:P,y:P) = {
 val z = x.a
 x.a = y.a
 y.a = z
}

def main () = {
 var p0 = P(0)
 var p1 = P(1)
 twiddle(p0,p1)
 println (p0.a + " " + p1.a)
 swizzle(p0,p1)
 println (p0.a + " " + p1.a)
}
```

Note 1: Scala statically distinguishes between identifiers for immutable values and identifiers for mutable variables. For local declarations, this distinction is marked by using the keyword `val` or `var`, respectively. By default, case class fields are immutable, but the `var` keyword in the definition for `P` makes `a` a mutable field.

Note 2: Function parameters are *always* immutable in real Scala, but we'll pretend that the `var` keywords before the parameters in the definition of `twiddle` (which aren't valid in real Scala) declare those parameters to be mutable variables that *can be* modified in the body of `twiddle`.

Otherwise, assume that the program has Scala-like semantics, except for parameter passing. In particular, assume that objects of class `P` are boxed.

(a) What does the program print under call-by-value semantics? Explain briefly why.

(b) What does the program print under call-by-reference semantics? Explain briefly why.

(c) Now suppose that objects of class `P` are *not boxed*, and the semantics of assignment are adjusted appropriately. Now what does the program print under call-by-value semantics? Explain briefly why.