

CPTS 355 Sample Final Exam

Here are some questions of the kind I'll be asking on the Final Exam. This is a sample of question types likely found on the exam. It is not exhaustive the of the topics – refer to the class calendar, nor is it representative of the length of the exam. You should expect, topic wise, the 40-50% of the material on the exam will be from the last third of the class, and 50-60% will be review material from the previous two exams.

The exam is open-book, open-notes; you may **not** use a computer, PDA, cell phone, etc. during the test.

1a) [5] Recall that ML uses static scoping. What is the value of the following ML expression? Explain how you determined your answer.

```
let
  val a = 4
  fun b () = a
  fun c () = let
    val a = 6
  in
    b()
  end
in
  c ()
end
```

1b) [5] *What if* ML used dynamic scoping. What is the value of the expression given in 1a) in that case? Explain how you determined your answer.

(2) [10] What is the type of the following ML expression?

```
fun ok t b [] = b
  | ok t b (x::xs) = if (t x) then (ok t b xs) else false
```

4) [10] Consider the following python code fragment:

```
def r (x):
    def q (x):
        try:
            return p()
        except "oops", v:
            return v+3
    def p ():
        if x==0: raise oops,7
        else: return 5
    try:
        return q(x-4)
    except "oops", v:
        return v-3
```

What value is returned by `r (4)` ? Why?

What value is returned by `r (0)` ? Why?

5) [20] (**two parts**) Multiple inheritance poses a number of problems for language designers and implementers. Write separate answers for the following problems.

a) Describe what leads to a *name clash*. Illustrate your answer with example **C++** code. What are some strategies adopted in different languages for dealing with name clashes?

b) Describe *diamond inheritance*. Illustrate your answer with **Python** code exhibiting diamond inheritance. What program design issues are associated with diamond inheritance?

6) [20] **Draw the call stack** at the point marked by ← in the following Python program. Clearly label each activation record with the name of the procedure that it represents and show the static and dynamic links in each activation record. (Recall that in Python assigning to a variable creates an instance of that variable in the scope where the assignment occurs; so in this program there are separate variables named y associated with procedures main, A, and the second B, and a separate variables x associated associated with main and the first B.)

```
def main ():
    y = 5
    x = 12
    def C ():
        def B ():                # this is "the first B"
            x = 14
            B()
            z = y+x ←
            print z
    def A ():
        y = 11
        def B ():                # this is "the second B"
            y = 7
            C()
    A()
```

Using your diagram of the call stack, explain how the uses of variables x and y at the ← are resolved to the correct binding occurrences (either the one in A, the one in B, or the one in Main).

7) Consider the following code in a language which is like C but parameters are passed **by reference**.

```
int x = 4;
void p(int y, int z) {
    y = z*3;
    z = z-2;
}
p(x, x);
```

[5] Draw the activation record stack just before p returns. What is the value of variable x at the end of execution?

10) Race conditions: suppose one thread executes the assignment

$rc = rc + 1$

while another thread concurrently executes

$rc = rc - 1$

10a) On a typical modern processor how would $rc = rc + 1$ be implemented as machine instructions?

10b) If the rc starts out with value 1 and then the two threads each execute their statement, what are the possible final values of rc . Explain your answer.