cs1120: Exam 2

Due: Monday, 29 November at 3:30pm (in class)

UVA ID (e.g., wrw6y) : ANSWER KEY

Directions

Work alone. You may not discuss these problems or anything related to the material covered by this exam with anyone except for the course staff between receiving this exam and turning it in.

Open resources. You may use any books you want, lecture notes, slides, your notes, and problem sets. You may ***not*** use DrScheme or IDLE (or PLT Scheme or Racket or Python), but it is not necessary to do so. You may also use external non-human sources including books and web sites. If you use anything other than the course books, slides, and notes, cite what you used. You may not obtain any help from other humans other than the course staff.

Answer well. Answer all questions 1-9 (question 0 is your UVA ID in two places, which hopefully everyone will receive full credit for), and optionally answer questions 10-11.

You may either: (1) print out this exam and write your answers on it or (2) write your answers directly into the provided Word template and print the result out. Whichever one you choose, you must turn in your answers printed **on paper** and they must be clear enough for us to read and understand. You should not need more space than is provided to write good answers, but if you want more space you may attach extra sheets. If you do, make sure they are clearly marked.

The questions are not necessarily in order of increasing difficulty, so if you get stuck on one question you should continue on to the next question. There is no time limit on this exam, but it should not take a well-prepared student more than a few hours to complete. It may take you longer, though, so please do not delay starting the exam. There is no valid excuse (other than a medical or personal emergency) for running out of time on this exam.

No "snow jobs". If you leave a question **blank**, you will receive ~**30%** of the points for it. If you have no idea and waste our time with long-winded guessing, we will be less sanguine and the grading will be more sanguine. :-)

Use any procedure from class. In your answers, you may use any Scheme or Python procedure that appears in the lecture notes or in the book without redefining it (e.g., length, filter, sort, map, etc.). If there are multiple similar names (e.g., map vs. list-map, len vs. length), use whichever you like.

Full credit depends on the clarity and elegance of your answer, not just correctness. Your answers should be as short and simple as possible, but not simpler. Your programs will be judged for correctness, clarity and elegance, but you will not lose points for trivial errors (such as missing a closing parenthesis).

UVA ID *again* (e.g., wrw6y) : ANSWER KEY

Your Scores

0	1	2	3	4	5	6	7	8	9	EC	Total

(Your scores are recorded on the second page so that they are not visible to other students when tests are distributed or passed back. *We always use cover sheets on our TPS reports. Didn't you get that memo?*)

Question 0: Write your UVA ID on this page and the previous one and ensure that your exam is stapled cleanly. Non-staple attachments will lose points.

Question 1. Latency and Bandwidth.

Define a procedure **make-latency** that takes two arguments: a latency in seconds and a number of packets. Your procedure should return a sequence of packet arrival times that have the given latency. The arrival times must be in non-decreasing order. Also, indicate the expected output of (make-latency 10 3) for your code. Hint: this is more of a thought puzzle than a tricky programming puzzle. Many answers are possible! For example:

```
> (make-latency 10 3)
(10 12 15)            # other answers are possible
> (make-latency 5 6)
(5 6 7 8 9 10)        # other answers are possible
> (make-latency 2 3)
(2 2.5 2.5)           # other answers are possible
```

```
(define (make-latency latency numpackets)
     (if (= numpackets 0) null
          (cons latency
               (make-latency latency (- numpackets 1)))))
```

For my code, (make-latency 10 3) will output: 10 10 10
For latency, only the first packet arrival time matters. Another very common approach was to increment the latency by 1 each time (i.e., 10 11 12).

Now define a procedure **make-bandwidth** that takes two arguments: a bandwidth in bits per second and a number of packets. It must produce a sequence of non-decreasing packet arrival times that has the given overall average bandwidth. Also, indicate the expected output of (make-bandwidth 2 2) for your code. Examples:

```
> (make-bandwidth 2.0 4)
(0.5 1.0 1.5 2.0)     # other answers are possible
> (make-bandwidth 2.0 4)
(1 1 2 2)             # other answers are possible
> (make-bandwidth 1 1)
(1)
```

```
(define (make-bandwidth bandwidth numpackets)
  (if (= numpackets 0) null
    (list-append
      (make-bandwidth bandwidth (- numpackets 1))
      (list (/ numpackets bandwidth))
)))
```

We use list-append instead of cons to construct the list in ascending order.
For my code, (make-bandwidth 2 2) will output: (0.5 1.0)
    # 2 bits in 1 second = 2 bits/second
Average bandwidth is the total number of packets divided by the last arrival time.
For my code, (make-bandwidth 5 4) will output: (0.2 0.4 0.6 0.8)
    # 4 bits in 0.8 seconds = 4/0.8 = 5 bits/second, as requested!

Question 2. Static Type Checking

Suppose we change the way Lambda works in StaticCharme. In the class notes, statically-typed lambda looks like this:

(lambda (x : Number y : Number) (* x y))

But now we want to add another typing annotation for the return type! Examples of well-typed lambdas in this brave new world:

(lambda (x : Number y : Number) (< x y) : Bool)
(lambda (s : String) (string-length s) : Number)
(lambda (s : String x: Number) (= x (string-length s)) : Bool)

The added return type annotation is underlined. Here is an example of a badly-typed lambda:

(lambda (x : Number y : Number) (+ x y) : Bool) # *Number does not match Bool*

Complete the following code to typecheck this brave new lambda:

```
def typeLambda(expr,env):
     assert isLambda(expr)
     if len(expr) != 5:
          evalError("Bad lambda expression: %s" % str(expr))
     # from slide 18 of class notes 22
     # only the text in bolded purple below is new
     newenv = Environment(env)
     params = expr[1]
     paramnames = []
     paramtypes = []
     assert len(params) % 3 == 0
     for i in range(0, len(params) / 3):
        name = params[i*3]       # not i*5
        assert params[(i*3)+1] == ':'
        typ = CType.fromParsed(params[(i*3)+2])
        paramnames.append(name)
        paramtypes.append(typ)
        newenv.addVariable(name, typ, None)
      resulttype = typecheck(expr[2], newenv)
      declared_resulttype = Ctype.fromParsed(expr[4])
      if not resulttype.matches(declared_resulttype):
           evalError("Mistyped lambda ...")
      return CprocedureType(CProductType(paramtypes), \
            resulttype)
# You must use Ctype.fromParsed on expr[4]
# You must use .matches() instead of =. We defined the
matches procedure in class.
# You must evalError(), not just assert.
```

Question 3. Is It Computable?

Consider the not-so-famous **Program-Contains-Lolcat** problem:

> **Input:** Program source code S.
> **Output:** True if S contains the literal textual string “Lolcat”, False otherwise.

Is **Program-Contains-Local** computable or not? Why or why not?

Answer: Program-Contains-Lolcat is computable.
There exists a Turing machine that can compute it. There also exist Scheme and Python programs that can compute it. Since we do not have to run the program, only inspect its source code, this is just like asking if it is computable to look up a word in the dictionary. Yes, it is computable. In the worst case, just read the entire dictionary looking for the word.

Common Misconception #1: Thinking that the source code could be infinite. Source code is never infinite. In addition, program inputs are never infinite (cf. using something like lazy data structures to *simulate* infinite inputs). While list-length can handle "inputs of any size" or "an infinite number of inputs", any particular list you pass to it will be of finite length. Similarly, source code is always finite -- just like books written by humans are always finite.

Common Misconception #2: Thinking that the problem was asking you to run the program S. If you reread the description carefully, you will see that it does not ask you to run the program S at all.

Question 4. Is It Computable?

Consider the even-less-famous **Program-Produces-Lol-Before-Cat** problem:

Input: Program source code S.
Output: True if S produces the string “lol” and then later the string “cat” when run on the input 0. False otherwise.

Example: If (S 0) is “lol” “wes” 55 “cat”, then the output should be True.
Example: If (S 0) is “Seneca” “Falls”, then the output should be False.
Example: If (S 0) is “lol” “falls”, them tje output should be False.

Is **Program-Produces-Lol-Before-Cat** computable or not? Why or why not?

Answer. Program-Produces-Lol-Before-Cat is NOT computable.

It is not computable because no algorithm exists that could compute it. We prove this by a reduction involving the Halting Problem (see Course Book Section 12.4).

The proof is by contradiction. We will assume that PPLBC can be solved. Using that assumption, we'll solve the Halting Problem. Since the Halting Problem cannot be solved, we will have reached a contradiction, and we can conclude that PPLBC cannot be solved.

Assuming an algorithm to compute PPLBC exists. Here's the solution to the Halting Problem:

```
(define (halts? S)
    (if (PPLBC
        (begin
            (remove-all-lols S)
            (display "lol")
            (display "cat")
        ))
        #t      ;; we know it halts!
        #f      ;; we know it loops forever!
)
```

The only way the newly-constructed (begin ...) program can produce lol before cat is if S halts. So we could use PPLBC to solve the Halting Problem. But since the Halting Problem is not computable, PPLBC cannot be computable.

Question 5. Object Oriented Coding: Functional Python Trees

We want to implement trees using Python, similar to the trees mentioned in Class #10 and Class #12. We will have an EmptyTree class, similar to null or the empty list, and a Tree subclass that contains a left child, a number value, and a right child.

Both Tree and EmptyTree must support an insert() method that *returns a new tree object* just like the old one but with the new value inserted. This insert() method is very similar to the insert-one-tree method in Class #12. Example:

```
>>>  a = EmptyTree()
>>>  print a
Empty

>>>  b = a.insert(5)
>>>  print b
Tree(Empty,5,Empty)

>>>  c = b.insert(4)
>>>  print c
Tree(Tree(Empty,4,Empty),5,Empty)

>>>  d = c.insert(6)
>>>  print d
Tree(Tree(Empty,4,Empty),5,Tree(Empty,6,Empty))

>>>  e = d.insert(1)
>>>  print e
Tree(Tree(Tree(Empty,1,Empty),4,Empty),5,Tree(Empty,6,Empty))
```

(continued on next page)

Complete the following code:

```
class EmptyTree:
  def __init__(self):
    return
  def __str__(self):
    return "Empty"
  def insert(self, newValue):
    return Tree(EmptyTree(), newValue, EmptyTree())

class Tree (EmptyTree):
  def __init__(self, leftChild, myValue, rightChild):
    self.leftChild = leftChild
    self.rightChild = rightChild
    self.value = myValue
  def __str__(self):
    return "Tree(%s,%d,%s)" % (self.leftChild , self.value , self.rightChild)
  # write your code here #
  def insert(self, newValue):
          if self.value <= newValue:
                  return Tree(    self.leftChild.insert(newValue),
                                  self.value,
                                  self.rightChild)
          else:
                  return Tree(    self.leftChild,
                                  self.value,
                                  self.rightChild.insert(newValue))
```

# write your code here #

# Misconception #1: checking to see if self is and EmptyTree. That is totally unnecessary here because of how object oriented programming works: if you're inside Tree's insert method, self must be a Tree (or a subclass of Tree). It cannot be an EmptyTree. Review Section 10.2 in the course book.

# Misconception #2: Not returning a new Tree(). The problem asks you to return a new tree (i.e., functional programming). Many of you imperatively and destructively changed the current tree.

# Misconception #3: Calling insert on the leftChild, but forgetting to include the rightChild in your answer. You'll lose half the tree on every insert!

Question 6. Lambda and the Environment Model

Draw the relevant portions of the environment model diagram when evaluating this code:

```
Scheme> (define x 5)
Scheme> ((lambda (x y) (+ (x 2) y))  (lambda (w) (+ x 4))  6)
15
```

In particular, be sure to write each lambda body next to the environment in which it is being evaluated.

Hint 1: There is only one define here, so most of the work will be done by “anonymous” or “nameless” functions. They will still show up in various ways in your environment model diagram.

Hint 2: Pay careful attention to the evaluation rules for lambda expressions and application expressions.

Hint 3: Draw the global environment, but leave room outside of it to draw other environments.

Question 7. Running Times & Evaluation

Consider the following Charme code. The parameters **a** and **b** are lists of numbers.

```
(define (fox x y)
        (if (null? x) 0
                (+ (car x) (fox (cdr x) y))))

(define (hound a b)
        (fox a (list-length b)))

(define (tortoise a b)
        (if (null? a)
                #f
                (if (list-contains-element? b (car a)) # is the element (car a) in the list b?
                        #t
                        (tortoise (cdr a) b))))

(define (hare a b)
        (if (null? a)
                0
                (+      (fox b 0)
                        (hare (cdr a) b))))
```

For each of the following questions, give the running time in Big Theta notation in terms of the length of **a** and/or **b**. Do not give an answer like Θ(N) – you must give answers like Θ(ab^2) instead.

7.a. What is the running time of **hound** in Charme? Θ(a+b)

To evaluate (function arg1 arg2), you first evaluate function, then evaluate arg1, then evaluate arg2, and *then finally* call the function on the actual argument values. So we compute the list-length of b first, *once only*, and then go on to call fox. So it's Θ(b) time to compute the list length once, followed by Θ(a) time for fox. You do not multiply them. See Chapter 7.

7.b. What is the running time of **hound** in MemoCharme? Θ(a+b) -- there is nothing to memoize.

7.c. What is the running time of **hound** in LazyCharme? Θ(a) -- if you look closely, fox never uses its y argument, so we don't need to spend Θ(b) time computing the list-length of b.

7.d. What is the running time of **tortoise** in Charme? Θ(ab) -- each call to tortoise takes Θ(b) time for the call to list-contains-element. There are Θ(a) calls to tortoise, one for each element of a. So the total is Θ(ab)

7.e. What is the running time of **tortoise** in MemoCharme? Θ(ab) -- nothing to memoize

7.f. What is the running time of **tortoise** in LazyCharme? Θ(ab) -- nothing to skip

7.g. What is the running time of **hare** in Charme? Θ(ab) -- each call to hare takes Θ(b) time to execute the call to (fox b 0). There are Θ(a) recursive calls to hare -- one for each element of a. So the total time is Θ(ab).

7.hi. What is the running time of **hare** in MemoCharme? -- the call to (fox b 0) is the same each time, so after the first time we compute it we can *memoize* the result. So it takes Θ(b) to call (fox b 0) once, but after that each call takes Θ(1) time. There are Θ(a) recursive calls to hare, only one of which takes Θ(b) time. So Θ(a+b).

A common misconception was to assume that we did not even have to run (fox b 0) the first time, and thus the total running time is Θ(a). That's tempting, but in order to look up the memoized value, we must have computed it at least once -- just like in PS7!

7.j. What is the running time of **hare** in LazyCharme? Θ(ab) -- all values are needed, laziness saves us nothing.

Question 8. Databases

Suppose we are going to implement a simpler version of the auction from Problem Set 5. In this version, we have only a single bids table. Each bid is a three-element list containing the bidder, the bid amount, and the desired object. Example:

```
(define bids    (list    (list “richard-castle”    1000    “library”)
                         (list “kate-beckett”      2000    “rotunda”)
                         (list “alexis-castle”     50      “library”)
                         (list “richard-castle”    789     “rotunda”)
                         (list “alexis-castle”     10      “rotunda”)
                         (list “kate-beckett”      1500    “library”)
                ))
```

You must write a procedure **library-over-rotunda** that returns all bid amounts for bids on the library that were placed by a person who bid on both the library and the rotunda, but bid more on the library. Example:

```
>  (library-over-rotunda bids)
(1000 50)                # 1000 from “richard-castle” is more than 789
                         # 50 from “alexis-castle” is more than 10
                         # the answer (50 1000) is also acceptable
```

Although the examples here use Scheme, *you may write your answer in Python if you prefer.*

Hint for Scheme:

```
> (car (list 1 2 3))

1

> (cadr (list 1 2 3))

2

> (caddr (list 1 2 3))

3
```

General hint: You may use ideas like **table-select**, but the bids table format for this question is not the same as the bids table format for PS5, so **table-select** won't work “out of the box”.

(continued on next page)

Fill in your definition for **library-over-rotunda** here:

We assume that each person can only bid on an item at most once.

```
# Here's on answer that gathers up a list of names, notes all of the bids, and then
# checks each person directly to see if that person bid more on the library.
def library-over-rotunda(bids):
        answer = []
        names = []
        library_bids = {}
        rotunda_bids = {}
        # gather up the names of all bidders
        for bid in bids:
                name = bid[0]
                if not (name in names):
                        names = names + [name]
                if bid[2] = "library":
                        library_bids[name] = bid[1]
                else if bid[2] = "rotunda":
                        rotunda_bids[name] = bid[1]
        for name in names:
                if name in library_bids & name in rotunda_bids & \
                        library_bids[name] > rotunda_bids[name]:
                        answer = answer + [ library_bids[name] ]
        return answer

# Here is a more compact answer. Just consider every possible pair of bids.
def library-over-rotunda(bids):
        answer = []
        for x in bids:
                for y in bids:
                        if x[0] = y[0] & x[2] = "library" & y[2] = "rotunda" & \
                                x[1] > y[1]:
                                answer = answer + [ x[1] ]
        return answer
```

Question 9. Security and Cryptography

For this question, you will be implementing one-time pad encryption for lists of bits. You will write a function called **otp** that takes as input two lists: a list of bits (the text) and another list of bits (the pad). The pad list will be at least as long as the text list. The output will be a list of bits – the text encrypted via the pad. We will represent bits using the numbers 0 and 1. *You may write in Scheme or Python – whichever you prefer.* Modular addition on bits is carried out using the XOR operator. (Feel free to search around for more hints on one-time pads if you need them.) Examples:

```
Python>>> 1 ^ 1          # ^ is XOR in Python
0
Scheme> (xor 1 1)        ; xor is XOR in Scheme
0
```

```
(define (otp text pad)
     (if (null? text) null
          (cons (xor (car text) (car pad))
               (otp (cdr text) (cdr pad)))))
     or
(define (otp text pad)
     (map xor text pad))
     or
def otp(text,pad):
     answer = []
     for i in range(len(text)):
          answer[i] = text[i] ^ pad[i]
     return answer
```

What is the running-time, in Big Theta notation, of the best possible implementation of **otp**?

Theta(Text)

There may be more elements in pad, but we'll ignore them -- we stop at the end of the text. Computing xor takes constant time. So there are |Text| recursive calls, each of which takes constant time, for a total of Theta(Text) time.

Could a Turing machine compute the **otp** function? Why or why not?

Yes. Turing Machines are Universal -- anything that Scheme or Python can do, a Turing Machine can also do. Misconception: Turing machines can only accept a single input. See the "binary add" example in the class notes for a counter-example.

Question 10a. (1 point of extra credit max) Did you complete the API Examples Study on or before midnight Friday November 19th? What was your completion code?

Question 10b. Zero points. Do you think your performance on this Exam will fairly reflect your understanding of the material in the second half of the course? If not, explain why.