Final Exam
CSCI 1101 Computer Science I

Section **04**

KEY

Thursday December 18, 2014
Instructor Muller
Boston College
Fall 2014

Please do not write your name on the top of this test. Before reading further, please arrange to have an empty seat on either side of you. Now that you are seated, please note the number on top of your test and write it together with your name on the sheet that is circulating.

This is a closed-book exam but you may use one 8.5 by 11 sheet of notes. Computers, calculators and books are prohibited. For problems other than 3. involving repetition, feel free to use a solution to one problem in solving another problem. And feel free to use any repetition form that you would like.

Partial credit will be given so be sure to show your work. **Please try to write neatly. And happy holidays!**

Problem	Points	Out Of
1		8
2		6
3		21
4		9
Total		44

1 Snippets (8 Points Total)

For each of the following code snippets, indicate what would happen when attempting to load and then run them. If they have a problem that would be detected when loaded, indicate the problem. If they have a problem that would be detected when run, indicate the problem. If they have no problems, indicate what value they would produce.

1. (2 Points)

   ```
   def h(a, b):
     a * b = c
     return c

   h(4, 5)
   ```

   Answer: Loadtime error cannot assign to a * b

2. (2 Points)

   ```
   def f(x, y):
     z = x / y
     return z

   f(3, 1 / 2)
   ```

   Answer: Runtime error div by 0

3. (2 Points)

   ```
   def what(p, xs):
     if xs == []:
       return False
     else:
       first = xs[0]
       rest = xs[1:]
       return p(first) or what(p, rest)

   def isEven(n): return n % 2 == 0

   what(isEven, [1, 3, 5])
   ```

   Answer: Code is OK, returns `False`.

4. (2 Points)

   ```
   def f(m, n):
     return [[ i for i in range(m) ] for _ in range(n) ]

   f(2, 3)
   ```

   Answer: Code is OK, returns `[[0, 1], [0, 1], [0, 1]]`.

2 Fluency with Repetition Idioms (6 Points)

1. (2 Points) Rewrite the following function without using recursion.

```
def isPrime(n):
  def loop(m):
    if m > math.sqrt(n):
      return True
    else:
      return (n % m != 0) and loop(m + 1)
  return loop(2)
```

Answer:

```
def isPrime(n):
  m = 2
  while m < math.sqrt(n):
    if (n % m == 0) return False
    m = m + 1
  return True
```

```
def isPrime(n):
  for m in range(2, math.sqrt(n) + 1):
    if (n % m == 0) return False
  return True
```

2. (2 Points) Rewrite the following function without using a `for`-loop.

```
def exists(test, xs):
  for x in xs:
    if test(x): return True
  return False
```

Answer:

```
def exists(test, xs):
  if xs == []:
    return False
  else:
    (first, rest) = (xs[0], xs[1:])
    return test(first) or exists(test, rest)
```

```
def exists(test, xs):
  i = 0
  while(i < len(xs)):
    if test(xs[i]): return True
    i = i + 1
  return False
```

3. (2 Points) Let `gameOver : board -> bool` and `makePlay : board -> board`. Rewrite the following function without using a `while`-loop.

```
def play(board):
  while not(gameOver(board)):
    board = makePlay(board)
  print "game over"
```

Answer:

```
def play(board):
  if gameOver(board):
    print "game over"
  else:
    play(makePlay(board))
```

3 Repetition (21 Points Total)

This section has nine problems. There are seven easier 3 point problems and two more challenging 6 point problems. Do any of the nine problems totalling 21 points.

1. (3 Points) Write a function `doubleAll : int list -> int list` such that a call `doubleAll(ns)` returns a list of the same length as `ns` but with each element of `ns` doubled. For example, the call `doubleAll([1, 2, 3])` should return `[2, 4, 6]`.

   Answer:

   ```
   def doubleAll(ns): return [ n * 2 for n in ns ]
   ```

2. (3 Points) Write a function `factors : int -> int list` such that a call `factors(n)` returns the list of all integer factors of `n` that are less than `n`. For example, the call `factors(12)` should return the list `[1, 2, 3, 4, 6]`.

   Answer:

   ```
   def factors(n):
     return [ k for k in range(1, n/2 + 1) if n % k == 0]
   ```

3. (3 Points) A number n is *perfect* if its smaller factors sum to n. For example, 6 is perfect because its factors are 1, 2 and 3 and these numbers add up to 6. Write a function `isPerfect : int -> bool` such that a call `isPerfect(n)` returns `True` if `n` is perfect. Otherwise, `isPerfect` should return `False`.

   Answer:

   ```
   def perfect(n): return sum(factors(n)) == n
   ```

4. (6 Points) A piece of text could be represented as a list of words `["four", "score", "and", "seven", ...]` where each word is a lowercase string with no punctuation, spaces, tabs or `newlines`. Write a function `mostFrequent : string list -> string * int` such that a call `mostFrequent(words)` returns the most frequently occurring word together with its frequency. You may assume that there are no ties. Feel free to use helper functions but you'll have to write them.

Answer:

```
def count(word, words):
  c = 0
  for w in words:
    if w == word: c = c + 1
  return c

def maximum(wordCountPairs):
  max = wordCountPairs[0]
  for (word, count) in wordCountPairs:
    if count > max[1]:
      max = (word, count)
  return max

def mostFrequent(words):
  counts = map(lambda word : (word, count(word, words)), words)
  return maximum(counts)
```

5. (3 Points) Write a function `uniques : int list -> int list` such that a call `uniques(ns)` returns a list that is like `ns` but which contains no duplicates. For example, the call `uniques([1, 3, 2, 3, 4])` should return a list like `[1, 2, 3, 4]`.

   Answer:

```
def uniques(ns):
  if ns == []:
    return []
  else:
    first = ns[0]
    rest = ns[1:]
    answer = uniques(rest)
    if first in answer:
      return answer
    else:
      return [first] + answer
```

6. (3 Points) Write a function `copies : (int * a) list -> a list`, such that a call `copies([(n1, v1), ..., (nk, vk)])` returns a list `[v1, ..., v1, ..., vk, ..., vk]` where there are `n1` copies of `v1` and `nk` copies of `vk`. For example, the call `copies([(3, 'A'), (4, 'B')])` should return the list `['A', 'A', 'A', 'B', 'B', 'B', 'B']`.

   Answer:

```
def copies(pairs):
  if pairs == []:
    return []
  else:
    (n, v) = pairs[0]
    rest = pairs[1:]
    return ([v] * n) + copies(rest)
```

7. (3 Points) Assume the existence of printing functions `out : string -> void` and `outln : string -> void`, where `out(s)` prints string `s` without a trailing `newline` and where `outln(s)` prints string `s` with a trailing `newline`. For example, the successive calls:

```
out("A")
out("B")
```

prints one line:

```
AB
```

Write a function `many : int * string -> void` such that a call `many(n, s)` prints `n` copies of string `s` on one line. For example, the call `many(3, "A")` should print `AAA`.

Answer:

```
def many(n, s):
  if n > 0:
    out(s)
    many(n - 1, s)
```

```
def many(n, s):
  for _ in range(n):
    out(s)
```

```
def many(n, s):
  while n > 0:
    out(s)
    n = n - 1
```

8. (3 Points) Write a function `wedge : int -> void` such that a call `wedge(n)` prints a wedge of `n` lines of stars ”*”. For example, the call `wedge(5)` should print

```
*****
****
***
**
*
```

Answer:

```
def wedge(n):
  if n > 0:
    many(n, ’*’)
    outln(’’)
    wedge(n - 1)
```

```
def wedge(n):
  for k in range(n):
    many(n - k, ’*’)
    outln(’’)
```

```
def wedge(n):
 while n > 0:
   many(n, ’*’)
   outln(’’)
   n = n - 1
```

9. (6 Points) Write a function `tree : int -> void` such that the calls `tree(3)` and `tree(4)` would print as follows:

```
   tree(3)            tree(4)

   ..*                ...*
   .*.*               ..*.*
   *...*              .*...*
                      *.....*
```

Answer:

```
def tree(n):
  for row in range(n):
    many(n - row - 1, '.')
    out('*')
    if row > 0:
      many((row - 1) * 2 + 1, '.')
      out('*')
    outln('')
```

4 Storage Diagrams (9 Points Total)

As we discussed in class, *dictionaries* (or *maps*) associating *keys* with *values* are ubiquitous in coding. Python has a very handy built-in dictionary type {`k1:v1, ..., kn:vn`} – we'll set that built-in type aside for this question and review a different representation that we covered in class. The main operations on a dictionary are:

```
find : key * dictionary -> value
```

`and`

```
insert : key * value * dictionary -> dictionary
```

Finding the value of a key in a dictionary can be efficient when there is an ordering of the keys that allows us to ask if one key is less than or greater than another. If this is the case, the dictionary can be represented as a *binary search tree*. In Python it's natural to represent an empty bst with `None` and a non-empty bst as a 4-tuple `(left, key, value, right)` where `left` and `right` are dictionaries and the keys are organized in such a way that every key in `left` is smaller than `key` and every key in `right` is greater than `key`. For example,

```
((None, 3, 'Z', None), 4, 'A', None)
```

represents a dictionary relating key `3` to value `'Z'` and key `4` to value `'A'`.

Code for `find`ing the value of a key in such a dictionary and `insert`ing a binding in a dictionary has been covered in class and is reproduced on the attached sheet. (Feel free to detach it.)

Thinking about the underlying storage, it would be reasonable to represent the empty bst `None` as the integer `0` and the non-empty bst `(left, key, value, right)` as an arrow pointing to a block of 4 consecutive words in the heap:

```
      +---+---+---+---+
 o--->| L | K | V | R |
      +---+---+---+---+
```

where `L` is the representation of dictionary `left`, `K` is the representation of `key`, `V` is the representation of `value` and `R` is the representation of dictionary `right`. For example, the dictionary `(None, 2, 'E', None)` would be represented as

```
      +---+---+-----+---+
 o--->| 0 | 2 | 'E' | 0 |
      +---+---+-----+---+
```

1. (3 Points) Given the scheme described above and the code on the attached sheet, show the state of the stack and the heap after (1) has been executed but before (2) has been executed.

```
def f():
  a = insert(4, 'A', None)           (1)
  return a                           (2)


        STACK                                  HEAP
```

Answer:

```
  +-----+
  |  f  |
  +-----+-----+                        +---+---+-----+---+
  |  a  |  o--+----------------------->| 0 | 4 | 'A' | 0 |
  +-----+-----+                        +---+---+-----+---+
```

2. (3 Points) Show the state of the stack and the heap after (1) has been executed but before (2) has been executed.

```
def f():
  a = insert(4, ’A’, None)
  b = insert(3, ’Z’, a)
  c = insert(5, ’B’, b)                (1)
  return c                             (2)

        STACK                              HEAP
```

Answer:

```
+-----+
|  f  |                              +---+---+-----+---+
+-----+-----+                   +--> | 0 | 4 | ’A’ | 0 |
|  a  |  o--+--------------------+   +---+---+-----+---+
+-----+-----+                        +---+---+-----+---+
|  b  |  o--+-----------------------># o | 4 | ’A’ | 0 |
+-----+-----+                        +-+-+---+-----+---+
|  c  |  o--+--+                       |
+-----+-----+   \                      v
                 \                   +---+---+-----+---+
                  \                  | 0 | 3 | ’Z’ | 0 |    SHARED!
                   \                 +---+---+-----+---+
                    \                  ^
                     \                 |
                      \              +-+-+---+-----+---+
                       +-----------> | o | 4 | ’A’ | o |
                                     +---+---+-----+-+-+
                                                     |
                                                     v
                                     +---+---+-----+---+
                                     | 0 | 5 | ’B’ | 0 |
                                     +---+---+-----+---+
```

3. (3 Points) Show the state of the stack and the heap after (1) has been executed but before (2) has been executed.

```
def f():
  a = insert(4, 'A', None)
  b = insert(3, 'Z', a)
  c = insert(5, 'B', b)
  d = insert(3, 'Q', c)
  e = insert(4, 'J', d)                 (1)
  return e                              (2)

        STACK                                  HEAP
```

Answer:

`See the previous answer.`

Dictionary Code for Section 4

```
def find(key, bst):
  if bst == None:
    return None
  else:
    (left, k, v, right) = bst
    if key == k:
      return v
    elif key < k:
      return find(key, left)
    else:
      return find(key, right)


def insert(key, value, bst):
  if bst == None:
    return (None, key, value, None)
  else:
    (left, k, v, right) = bst
    if key == k:
      return (left, key, value, right)
    elif key < k:
      newLeft = insert(key, value, left)
      return (newLeft, k, v, right)
    else:
      newRight = insert(key, value, right)
      return (left, k, value, newRight)
```