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!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Out Of</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>44</td>
<td></td>
</tr>
</tbody>
</table>
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)

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

h(4, 5)
```

Answer: Loadtime error cannot assign to a * b

2. (2 Points)

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

f(3, 1 / 2)
```

Answer: Runtime error div by 0

3. (2 Points)

```python
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)

```python
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.

```python
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:

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

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

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

Answer:

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

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

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

\textbf{Answer:}

```python
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:**

```python
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:**

```python
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:**

```python
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:**

```python
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:**

```python
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:**

```python
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 \texttt{out} : \texttt{string \to void} and \texttt{outln} : \texttt{string \to void}, where \texttt{out(s)} prints string \texttt{s} without a trailing \texttt{newline} and where \texttt{outln(s)} prints string \texttt{s} with a trailing \texttt{newline}. For example, the successive calls:

\begin{verbatim}
out("A")
out("B")
\end{verbatim}

prints one line:

\texttt{AB}

Write a function \texttt{many : int \times string \to void} such that a call \texttt{many(n, s)} prints \texttt{n} copies of string \texttt{s} on one line. For example, the call \texttt{many(3, "A")} should print \texttt{AAA}.

\textbf{Answer:}

\begin{verbatim}
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
\end{verbatim}

8. (3 Points) Write a function \texttt{wedge : int \to void} such that a call \texttt{wedge(n)} prints a wedge of \texttt{n} lines of stars \texttt{"*"}. For example, the call \texttt{wedge(5)} should print

\begin{verbatim}
*****
****
***
**
*
\end{verbatim}

\textbf{Answer:}

\begin{verbatim}
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
\end{verbatim}
9. (6 Points) Write a function \texttt{tree : int \rightarrow void} such that the calls \texttt{tree(3)} and \texttt{tree(4)} would print as follows:

\[
\begin{array}{c|c}
\text{tree(3)} & \text{tree(4)} \\
\hline
..* & ..* \\
.*.* & ..*.* \\
*...* & .*...* \\
*.....* & *.....*
\end{array}
\]

\textbf{Answer:}

\[
def \texttt{tree}(\texttt{n}): \\
\texttt{for row in range(n):} \\
\quad \texttt{many(n - row - 1, ',')} \\
\quad \texttt{out('*)} \\
\quad \texttt{if row > 0:} \\
\quad \quad \texttt{many((row - 1) * 2 + 1, ',')} \\
\quad \quad \texttt{out('*)} \\
\quad \texttt{outln('')}\n\]
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:

\[
\text{find : key * dictionary } \rightarrow \text{ value}
\]

and

\[
\text{insert : key * value * dictionary } \rightarrow \text{ dictionary}
\]

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,

\[
((\text{None}, 3, 'Z', \text{None}), 4, 'A', \text{None})
\]

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

Code for finding the value of a key in such a dictionary and inserting 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:

```
+-----------+
| 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

```
+-----------+
| 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.

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

**Answer:**

```
+-----+
| f |
+-----+
```

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

```
+-----+
| a |
+-----+
```

```
+-----+-----+
| 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)

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

```python
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)
```

**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)