CS1010FC — Programming Methodology
School of Computing
National University of Singapore

Solutions for Mid-Term Quiz

29 March 2014 **Time allowed:** 1 hour 45 minutes

Matriculation No:

Instructions (please read carefully):

1. Write down your matriculation number on the **question paper**. DO NOT WRITE YOUR NAME ON THE QUESTION SET!
2. This is **an open-sheet quiz**. You are allowed to bring one A4 sheet of notes (written on both sides).
3. This paper comprises **FIVE (5) questions** and **SEVENTEEN (17) pages**. The time allowed for solving this quiz is **1 hour 45 minutes**.
4. The maximum score of this quiz is **100 marks**. The weight of each question is given in square brackets beside the question number.
5. All questions must be answered correctly for the maximum score to be attained.
6. All questions must be answered in the space provided in the answer sheet; no extra sheets will be accepted as answers.
7. The back-sides of the sheets and the pages marked "scratch paper" in the question set may be used as scratch paper.
8. You are allowed to un-staple the sheets while you solve the questions. Please make sure you staple them back in the right order at the end of the quiz.
9. You are allowed to use pencils, ball-pens or fountain pens, as you like (no red color, please).

GOOD LUCK!

Question	Marks	Remark
Q1		
Q2		
Q3		
Q4		
Q5		
Total		

Question 1: Unwrapping the Mystery Number [25 marks]

Ben Bitdiddle has decided to play a little puzzle game with you. The game is very simple. Ben has hidden some non-zero mystery numbers in some Python expressions. Your job is to write the Python code that is required to extract the mystery numbers. We represent the mystery numbers below with `<?mystery number?>`. The following are two examples on how this game is played.

Example 1: For the function `foo`:

```
def foo():
    return <?mystery number?>
```

we can extract the mystery number with:

```
foo()
```

Example 2: For the function `bar`:

```
def bar (x):
    return x(<?mystery number?>)
```

we can extract the mystery number with:

```
bar(lambda x: x)
```

For each of the following expressions, write a Python expression that will print the mystery number in terms of `T1` to `T5`.

A. `T1 = lambda x: "failed" if x else <?mystery number?>`

[5 marks]

`T1(False)` or `T1(0)` or `T1('')` or `T1(())` or `T1([])` or `T1(None)`

This question tests if the student understands the ternary `if` statement.

B.
```
T2 = ("many", ("layers", "deep", "to", ("go",
        ("to", "extract", (<?mystery number?>)))))
```

[5 marks]

`T2[1][3][1][2]`

This question tests if the student understands tuple indexing. -2 if student has an extra `[0]`. Sorry, this question is a little tricky. `(<?mystery number?>)` is actually just `<?mystery number?>`.

C.
```
def T3(x):
    y = 1
    for i in range(5):
        if i > 3 and x(i) == y:
            return <?mystery number?>
        else:
            y = 2*y
```
[5 marks]

`T3(lambda n: 16)` or `T3(lambda n: n**2)` or `T3(lambda n: 4*n)`

This question tests if the student understands the `for` loop and functions as arguments to other functions.

D.
```
def T4(x,y,z):
    good = <?mystery number?>
    if x:
        y(good)
    else:
        return y(z(y(good)))
```
[5 marks]

`T4(False, lambda x:x, lambda x:x)`

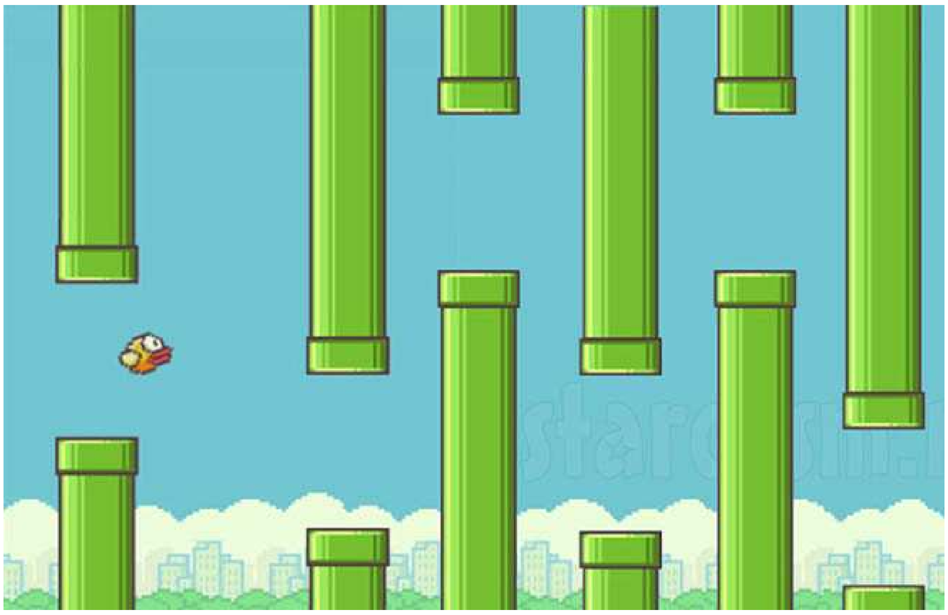
This question tests if the student understands the `return` statement and how to evaluate nested function calls. -2 if first argument is `True` because there is no return in front of `y(good)`!

E.
```
def T5(x):
    secret = ("failed",<?mystery number?>)
    while True:
        secret = ("failed",) + secret
        if len(secret) % 2 == 1:
            secret = secret[::2]
        if len(secret) > 5:
            x(secret)
```
[5 marks]

This question tests if the student understands the `while` loop and tuple manipulation. It is not possible to extract the mystery number using `T5`.

Question 2: Flappy Bird!! [24 marks]

Flappy Bird is a mobile game, developed by Vietnam-based developer Dong Nguyen and published by .GEARS Studios, a small, independent game developer also based in Vietnam. The game has a side-scrolling format and the player controls a bird, attempting to fly between rows of green pipes without coming into contact with them.



In this problem, you will solve some problems based on the game.

A. Suppose that the bird is originally at ground level (height = 0) and it needs to fly across a series of pipe obstacles at various heights. In each step, the bird can move either up one level, down one level, or stay at the same level. Given a tuple representing the levels of the obstacles, write the function `can_clear` that takes in the tuple and returns `True` if the bird can successfully clear the obstacles, or `False` otherwise. [10 marks]

Sample Execution:

```
>>> can_clear((1,2,1,1))
True

>>> can_clear((2,1,1)))
False

>>> can_clear((1,0,1,1))
True

>>> can_clear((5,))
False

>>> can_clear((0,1,1,2,3))
True
```

Recursive version:

```
def can_clear(obstacles):
    def helper (height, obstacles):
        if obstacles == ():
            return True
        elif obstacles[0] > height+1 or obstacles[0] < height-1:
            return False
        else:
            return helper(obstacles[0], obstacles[1:])
    return helper(0,obstacles)
```

Order of growth: $O(n^2)$ Time, $O(n)$ Space.

Iterative version:

```
def can_clear(obstacles):
    current_level = 0
    for level in obstacles:
        if level - current_level < -1 or level - current_level > 1:
            return False
        else:
            current_level = level
    return True
```

Order of growth: $O(n)$ Time, $O(1)$ Space.

-2 points if student fails to take care of empty input tuple.

B. What is the order of growth in terms of time and space for the implementation of `can_clear` in Part (A) above in terms of n, the number of obstacles. [4 marks]

Time: Depends on implementation. See above.

Space: Depends on implementation. See above.

C. Next, suppose that the bird can still only fly up or down one level in each step. but it now has a finite amount of energy and it takes one unit of energy to fly up by one step (and it takes no energy to fly down one level or stay at the same level). The bird will fail to clear a set of obstacles if it comes to an obstacle that is more than one step higher that its current height and does not have any energy to fly up. Write the function `enough_energy_to_clear` that takes in an initial amount of energy and a tuple of obstacles, and returns `True` if the bird can successfully clear the obstacles, or `False` otherwise. [10 marks]

Sample Execution:

```
>>> enough_energy_to_clear(10,(1,2,3,2))
True

>>> enough_energy_to_clear(2,(1,2,3,2))
False

>>> enough_energy_to_clear(3,(1,2,3,2))
True

>>> enough_energy_to_clear(3,(1,2,3,1))
False

>>> enough_energy_to_clear(10,(2,1,1))
False

>>> enough_energy_to_clear(10,(1,0,1,1))
True

>>> enough_energy_to_clear(1,(0,1,1,1,1))
True

>>> enough_energy_to_clear(10,(5,))
False

>>> enough_energy_to_clear(10,(0,1,1,2,3))
True
```

Recursive version:

```
def enough_energy_to_clear(energy, obstacles):
    def helper (height, energy, obstacles):
        if obstacles == ():
            return True
        elif energy < 0:
            return False
        elif obstacles[0] > height+1 or obstacles[0] < height-1:
            return False
        elif obstacles[0]>height:
            return helper(obstacles[0], energy-1, obstacles[1:])
        else:
            return helper(obstacles[0], energy,obstacles[1:])
    return helper(0,energy, obstacles)
```

Iterative version:

```
def enough_energy_to_clear(energy, obstacles):
    current_level = 0
    for level in obstacles:
        if level - current_level < -1 or level - current_level > 1:
            return False
        elif level - current_level == 1:
            if energy <=0:
                return False
            else:
                energy = energy - 1
        current_level = level
    return True
```

Some students would re-use `can_clear` from Part (A) and then count the number of steps that the bird needs to climb and then `return steps >= energy`.

Question 3: Higher Order Functions [24 marks]

A. Write a recursive function `sum_power_2(n)` that computes the following sum:

$$1 + 2 + 2^2 + \cdots + 2^n$$

You are not allowed to use a `for` or a `while` loop to solve this problem. [5 marks]

```
def sum_power_2(n):
    if n == 0:
        return 1
    else:
        return 2**n + sum_power_2(n-1)
```

B. Suppose `sum_power_2(n)` is defined as follows:

```
def sum_power_2(n):
    return sum(<T1>,
               1,
               <T2>,
               n+1)
```

Please provide possible implementations for the terms `T1` and `T2`, where `sum` is defined in lecture and reproduced in the Appendix. [4 marks]

`T1:`
[2 marks]

```
lambda n:  2**(n-1)
```

`T2:`
[2 marks]

```
lambda n:  n+1
```

C. Suppose `sum_power_2(n)` is instead defined as follows:

```
def sum_power_2(n):
    return sum(<T3>,
               1,
               <T4>,
               2**n)
```

Please provide possible implementations for the terms `T3` and `T4`. [4 marks]

`T3:` [2 marks]	`lambda n:  n`
`T4:` [2 marks]	`lambda n:  2*n`

D. Suppose `sum_power_2(n)` is defined as follows:

```
def sum_power_2(n):
    return fold(<T5>,
                <T6>,
                <T7>)
```

Please provide possible implementations for the terms `T5`, `T6` and `T7`, where `fold` is defined in lecture and reproduced in the Appendix. [6 marks]

`T5:` [2 marks]	`lambda a,b:a+b`
`T6:` [2 marks]	`lambda n:  2**n`
`T7:` [2 marks]	`n`

E. Suppose we now want to compute the following sum:

$$1+k+k^2+\cdots+k^n$$

Write a function `make_sum_power(k)` that will return a function that will compute this sum for k. [5 marks]

Sample Execution:

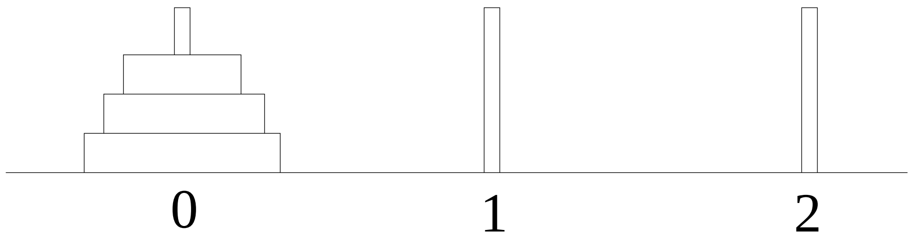
```
>>> sum_power_2 = make_sum_power(2)
>>> sum_power_2(0)
1
>>> sum_power_2(2)  # 1 + 2 + 2^2
7
>>> sum_power_2(3)  # 1 + 2 + 2^2 + 2^3
15

>>> sum_power_3 = make_sum_power(3)
>>> sum_power_3(0)
1
>>> sum_power_3(1) # 1 + 3
4
>>> sum_power_3(2) # 1 + 3 + 3^2
13
>>> sum_power_3(3) # 1 + 3 + 3^2 + 3^3
40
```

```
def make_sum_power(k):
    def helper(n):
        return sum(lambda n:  k**n, 0, lambda n:  n+1, n)
    return helper
```

The student would get zero credit if he returns `helper(n)` instead of `helper`. The point of this question is to test that the student has a firm grasp of returning functions in a function.

Question 4: Return of the Hanoi Nightmare [24 marks]



In the Tower of Hanoi puzzle, we have 3 pegs and a set of n discs. The goal of the problem is to move all n discs from the first peg to the third peg one disc at a time, subject to the condition that we cannot put a larger disc over a smaller one.

In this problem, you will implement an object that tracks the state of the puzzle. You will be asked to implement four functions:

1. `make_towers(n)` will take an argument n and creates a new tower object with n discs on the first peg. A tower object tracks the state of the 3 towers in the hanoi game.
2. `count_discs(tower,k)` will take two arguments, a tower object and an integer k, s.t. $0 \leq k \leq 2$, and returns the number of discs on peg k. By convention, the starting peg is peg 0 and the final destination peg is peg 2.
3. `move(tower, fr, to)` will take 3 arguments: a tower object, and two integers and try to move the disc from the fr peg to the to peg, s.t. $0 \leq fr, to \leq 2$, and return a **new** tower object after the move. Note that `move` will also check if the proposed move is valid. If a move is invalid, i.e. it attempts to put a bigger disc over a smaller disc, `move` will simply return the tower object in the input argument.
4. `win(tower)` will return `True` if the specified tower object is in the final (winning) configuration, i.e. all the discs are in the correct order on the third peg, or `False` otherwise.

Sample Execution:

```
>>> t = make_towers(2)
>>> count_discs(t,0)
2
>>> count_discs(t,1)
0
>>> count_discs(t,2)
0

>>> t2 = move(t,0,1)
>>> count_discs(t2,0)
1
>>> count_discs(t2,1)
1
```

```
>>> count_discs(t2,2)
0

>>> t3 = move(t2,0,1) # Invalid move!
>>> count_discs(t3,0)
1
>>> count_discs(t3,1)
1
>>> count_discs(t3,2)
0

>>> t4 = move(t3,0,2)
>>> count_discs(t4,0)
0
>>> count_discs(t4,1)
1
>>> count_discs(t4,2)
1
>>> win(t4)
False

>>> t5 = move(t4,1,2) # Win!
>>> count_discs(t5,0)
0
>>> count_discs(t5,1)
0
>>> count_discs(t5,2)
2
>>> win(t5)
True
```

A. Decide on an implementation for the tower object and implement `make_towers`. Describe how the state for the tower object is stored in your implementation. [6 marks]

There are many possible solutions. A straightforward approach is to use a tuple of 3 tuples as follows:

```
def make_towers(n):
    return (tuple(range(n,0,-1)),(), ())
```

Each tuple represents one peg and the order of the discs in each tuple is from bottom to top, i.e. element 0 is the bottom disc. The model **wrong** answer, which get **zero** points, is `(n,0,0)`. Basically, the key in designing ADTs is to figure out what we need to keep track of. It is not enough to keep track of the number of discs on each peg.

B. Provide a possible implementation of `count_discs` for the tower object defined in Part(A). [4 marks]

The solution here obviously depends on Part(A).

```
def count_discs(tower,k):
    return len(tower[k])
```

C. Provide a possible implementation of `move` for the tower object defined in Part(A). [10 marks]

```
def move(tower, fr, to):
    new_tower = [tower[0], tower[1], tower[2]]
    can_move = False
    if tower[fr] != ():  # Need at least one disc in the from tower
       if tower[to] == ():  # Sure can move
          can_move = True
       elif tower[to][-1] > tower[fr][-1]:
          can_move = True
    if can_move:
       new_tower[fr] = tower[fr][:-1]
       new_tower[to] = tower[to]+(tower[fr][-1],)
       return tuple(new_tower)
    else:
       return tower
```

Note that we have used a list in this solution for convenience. It is possible to solve this problem using only tuples. It will just be a lot more code. This question also highlights the convenience of having a mutable data structure.

Some students use list as their underlying data structure, but instead of returning a new tower object, they mutate the underlying object. This will cause them to lose 5 points.

D. Provide a possible implementation of `win` for the tower object defined in Part(A). Explain your implementation. [4 marks]

```
def win(tower):
    return tower[0]==() and tower[1] == ()
```

This solution is so simple because we have already checked that a move is valid in `move`. -2 if solution is not explained.

Question 5: Bonus Question! [3 marks]

What is the most important concept that you have learnt thus far? Explain.

Any reasonable answer will get some credit as long is it not patently false/wrong.

Appendix

The following are some functions that were introduced in class. For your reference, they are reproduced here.

```
def sum(term, a, next, b):
  if (a>b):
    return 0
  else:
    return term(a) + sum(term, next(a), next, b)

def fold(op, f, n):
  if n==0:
    return f(0)
  else:
    return op(f(n), fold(op, f, n-1))

def enumerate_interval(low, high):
    return tuple(range(low,high+1))

def filter(pred, seq):
    if seq == ():
        return ()
    elif pred(seq[0]):
        return (seq[0],) + filter(pred, seq[1:])
    else:
        return filter(pred, seq[1:])

def accumulate(fn, initial, seq):
    if seq == ():
        return initial
    else:
        return fn(seq[0], accumulate(fn, initial, seq[1:]))
```

Scratch Paper

— E N D   O F   P A P E R —