CS1010S — Programming Methodology
School of Computing, National University of Singapore

Solutions for Mid-Term Quiz

2 October 2013 **Time allowed:** 1 hour 45 minutes

Matriculation No: | | | | | | | | | |

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 **FOUR (4) 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		
Total		

Question 1: Python Expressions [25 marks]

There are several parts to this problem. Answer each part **independently and separately**. In each part, one or more Python expressions are entered into the interpreter (Python shell). Determine the response printed by the interpreter for the expression entered. If the interpreter produces an error message, or enters an infinite loop, explain why.

A.
```
x = 0
y = 5
z = 4
if x < y:
    print(z)
else:
    print(y/x*z)
```

[5 marks]

```
4
```

This question tests if the student understands the `if` statement and assignment.

B.
```
x = 1
y = 2
def f(x):
    if x <= y:
        return x
    else:
        return good-grief
print(f(2))
```

[4 marks]

```
2
```

This question tests if the student understands how to evaluate a simple function call.

C.
```
count = 1
for i in range(10):
    if i == 0:
        continue
    elif i == 5:
        break
    else:
        count = count + 1
print(count)
```

[4 marks]

```
5
```

This question tests if the student understands the `for` loop, as well as `continue` and `break`

D.
```
a = 0
b = 10
while a < b:
    a = a + 1
    b = b - 1
print(a)
```

[4 marks]

```
5
```

This question tests if the student understands the `while` loop.

E.
```
x = 4
def f(y):
    return g(x,y+x)
def g(y,x):
    return x
print(f(x))
```

[4 marks]

```
8
```

This question tests if the student understands how to evaluate functions with two parameters.

F.
```
def f(x,y):
    return y(x(y(x(y))))
def g(x):
    return 5
def h(x):
    return x
print(f(g,h))
```

[4 marks]

```
5
```

This question tests if the student understands how to evaluate complicated nested functions.

Question 2: Fibonacci Revisited [25 marks]

The Fibonacci sequence is named after Leonardo Fibonacci. His 1202 book Liber Abaci introduced the sequence to Western European mathematics, although the sequence had been described earlier in Indian mathematics.

In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the following recurrence relation:

$$F_n = F_{n-1} + F_{n-2}$$

with seed values:

$$F_0 = 0$$

$$F_1 = 1$$

A. **[Warm Up]** Write an **iterative** function `fib(n)` that returns the nth Fibonacci number. [6 marks]

```
def fib(n):
    a,b = 1,0
    for i in range(n):
        a,b = a+b,a
    return b
```

-2 points if `fib(n)` does not take care of `fib(0)` of `fib(1)` correctly.

B. What is the order of growth in terms of time and space for the function you wrote in Part (a) in terms of n. [4 marks]

Time: $O(n)$

Space: $O(1)$

C. Consider the the sequence G_n defined by the following recurrence relation:

$$G_n = G_{n-1}^2 + G_{n-2}^2$$

with seed values:

$$G_0 = 0$$

$$G_1 = 1$$

Write a function `fib_square(n)` that returns G_n. [6 marks]

```
def fib_square(n):
    a,b = 1,0
    for i in range(n):
        a,b = a*a+b*b,a
    return b
```

This question checks that the student is able to make a minor modification to the solution to a standard problem for a slightly different problem. Quite a few students decided to do this part by recursion, which is okay, but then the order of growth in Part D must match.

D. What is the order of growth in terms of time and space for the function you wrote in Part (c) in terms of n. [2 marks]

Time: $O(n)$

Space: $O(1)$

E. Consider the the sequence H_n defined by the following recurrence relation:

$$H_n = H_{n-1}^k + H_{n-2}^k$$

with seed values:

$$H_0 = a$$

$$H_1 = b$$

Write a function `create_fib(k,a,b)` that returns H_n for a specified value of k. In order words, suppose:

```
fib = create_fib(1,0,1)
fib_square = create_fib(2,0,1)
```

then, `fib` and `fib_square` would be equivalent to the functions that you wrote in Parts (a) and (c) respectively above. [7 marks]

```
def create_fib(k,a,b):
    def helper(n):
        x,y = b,a
        for i in range(n):
            x,y = x**k+y**k,x
        return y
    return helper
```

This question tests that the student understands higher-order functions well enough to define and return a function in `create_fib`. -3 points for returning `helper(n)` instead of `helper` since this suggests that the student does not completely understand the concept of returning a function.

-2 points for getting `a` and `b` wrong. Some students just return 0 and 1 instead of complying with the requirements. Zero points for the parts that are pretty much repeats from previous Parts.

Question 3: Higher Order Functions [22 marks]

Consider the following higher-order function that we call `flutter`:

```
def flutter(f, op, n):
    result = f(0)
    for i in range(n):
        result = op(f(i),result)
    return result
```

A. Write down the expression corresponding to the expression that `flutter` computes. You can use $\oplus$ to represent `op` in your expression. [4 marks]

$(f(n-1) \oplus (f(n-2) \cdots (f(2) \oplus (f(1) \oplus (f(0) \oplus f(0))) \cdots))$

This is a test of whether the students can interpret code accurately. Note that $f(0)$ is actually repeated in the expression that is computed! Note that if the student fails to get this part right, the answers in subsequent sections will be marked correct if they comply with the answer given here. Zero points for the following:

$f(0) \oplus f(1) \oplus f(2) \cdots f(n-1)$

Missing $f(0)$, no parenthesis (brackets) and wrong order.

B. Suppose the function `sum_integers(n)` computes the sum of integers from 1 to n (inclusive) and `sum_integers(n)` is defined as follows:

```
def sum_integers(n):
    return flutter(<T1>,
                   <T2>,
                   <T3>)
```

Please provide possible implementations for the terms `T1`, `T2`, and `T3`. [6 marks]

`T1:`

```
lambda x:  x
```

`T2:`

```
lambda x,y:  x+y
```

`T3:`

```
n+1
```

C. Suppose the function `product_integers(n)` computes the product of integers from 1 to *n* (inclusive) and `product_integers(n)` is defined as follows:

```
def product_integers(n):
    return flutter(<T4>,
                   <T5>,
                   <T6>)
```

Please provide possible implementations for the terms `T4`, `T5`, and `T6`. [6 marks]

`T4`:

```
lambda x:  x+1
```

-2 points for `lambda x: x` since this would cause the product to be zeroed. Also, if `lambda x: x` then T6 needs to be `n+1`. Note that `lambda x: 1 if x==0 else x` works.

`T5`:

```
lambda x,y:  x*y
```

`T6`:

```
n
```

D. [Pyramid Sum] We define the pyramid sum P_n as follows:

$$
\begin{aligned}
P_1 &= 1 && (1)\\
P_2 &= 1+2+1 && (2)\\
P_3 &= 1+2+3+2+1 && (3)\\
&\vdots = \vdots && (4)\\
P_n &= 1+\cdots+n-1+n+n-1+\cdots+1 && (5)
\end{aligned}
$$

Suppose we define `pyramid_sum(n)` in terms of `flutter` as follows:

```
def pyramid_sum(n):
    def pyramid(x):
        <T7>
    def op(a,b):
        <T8>
    return flutter(pyramid,
                   op,
                   n+1)
```

Please provide possible implementations for the terms `T7` and `T8`. [6 marks]

`T7`:

```
if x != n:
    return 2*x
else:
    return x
```

There are potentially many solutions to this problem. Since the third argument of `flutter` is fixed at `n+1`, life is a little complicated.

`T8`:

```
return a+b
```

An interesting matching pair that also works is:

```
return x
```

```
return a*a
```

Question 4: Rabbits!! [28 marks]

In this question, you will learn how to model rabbits.

A. **[Warm Up]** Write function `create_rabbit(gender,father,mother)` that will take 3 arguments, the gender of a rabbit as well as the parents of a rabbit respectively and return a new *rabbit* object. [3 marks]

```
def create_rabbit(gender,father,mother):
    return (gender,father,mother)
```

B. Write the following accessors: `get_gender`, `get_father` and `get_mother` for the rabbit object that you defined in Part (a) above. [6 marks]

```
def get_gender(rabbit):
    return rabbit[0]

def get_father(rabbit):
    return rabbit[1]

def get_mother(rabbit):
    return rabbit[2]
```

C. Assume that there are only two valid inputs for gender: ”Male” and ”Female” (strings). Write a function `mate(rabbit1, rabbit2)` that will take as arguments two rabbit objects. `mate` will return a new rabbit object if and only if `rabbit1` and `rabbit2` are of different gender, or `None` otherwise. Also, if `mate` does return a new rabbit, the rabbit has a 50% chance of being either male or female. **Hint:** The `random()` function will return a random value between 0 and 1. [6 marks]

```
from random import random

def mate(rabbit1,rabbit2):
    def make_rabbit(father, mother):
        if random()<0.5:
            return create_rabbit("Male", father, mother)
        else:
            return create_rabbit("Female", father, mother)

    if (get_gender(rabbit1) == "Male") and (get_gender(rabbit2) ==
"Female"):
        return make_rabbit(rabbit1,rabbit2)
    elif (get_gender(rabbit2) == "Male") and (get_gender(rabbit1) ==
"Female"):
        return make_rabbit(rabbit2,rabbit1)
    else:
        return None
```

If the student breaks the abstraction barrier, i.e. does `rabbit1[0]` or returns a tuple directly instead of `create_rabbit`, 4 points will be taken off. And this is true for subsequent questions as well, but -4 will only be taken off once even if abstraction is broken in several sub-questions. -3 points if student does not realize that `random` will return a random float between 0 and 1, and thinks instead that it returns either 0 or 1.

D. Write a function `is_parent(a,b)` that will return `True` if `b` is a parent of `a`, where both `a` and `b` are rabbit objects, or `False` otherwise. [4 marks]

```
def is_parent(a,b):
    return (get_father(a) is b) or (get_mother(a) is b)
```

Tests that the student understands the use of `is` and `or`. If == is used instead of `is`, student gets zero for this question.

E. Write a function `is_sibling(a,b)` that will return `True` if `a` and `b` are siblings, i.e. they share at least one parent, where both `a` and `b` are rabbit objects, or `False` otherwise. [4 marks]

```
def is_sibling(a,b):
    return (get_father(a) is get_father(b)) or (get_mother(a) is
get_mother(b))
```

Tests that the student understands the use of `is` and `or`. If == is used instead of `is`, student gets zero for this question, unless previous question was already zeroed.

F. Write a function `get_grandparents(a)` that will return a tuple containing **all** the grandparents of a rabbit object `a`. Note also that there **should not** be any duplicates in the returned tuple. [5 marks]

```
def get_grandparents(rabbit):
    def get_parents(rabbit):
        return (get_father(rabbit), get_mother(rabbit))

    if rabbit == None:
        return ()
    grandparents = get_parents(get_father(rabbit))
    for grandparent in get_parents(get_mother(rabbit)):
        if (grandparent is grandparents[0]) or (grandparent is
grandparents[1]):
            continue
        grandparents += (grandparent,)
    return grandparents
```

This question tests that the student is able to iterate over a tuple and use `is` to ensure that there are no duplicates. This is pretty much an all-or-nothing question. It is unlikely that partial credit will be given unless the error is *really* minor. Alternative solution:

```
def get_grandparents(rabbit):
    if rabbit == None:
        return ()
    grandfather = get_father(get_father(rabbit))
    grandmother = get_mother(get_father(rabbit))
    grandpa = get_father(get_mother(rabbit))
    grandma = get_mother(get_mother(rabbit))
    if grandfather is grandpa:
        grandfathers = (grandpa,)
    else:
        grandfathers = (grandpa, grandfather)
    if grandma is grandmother:
        grandmothers = (grandma,)
    else:
        grandmothers = (grandma, grandmother)
    return grandfathers + grandmothers
```

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

— END OF PAPER —