CS1010X — Programming Methodology
School of Computing
National University of Singapore

Midterm Test

30 March 2019 **Time allowed:** 1 hour 45 minutes

Student 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 test**. 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 test is **1 hour 45 minutes**.
4. The maximum score of this test 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 test.
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: Python Expressions [24 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 final expression entered and **write the exact output in the answer box**. If the interpreter produces an error message, or enters an infinite loop, explain why. Partial marks may be awarded for workings if the final answer is wrong.

A.

```
a = 2
b = 1
def f(a, b):
    def g(a, b):
        if a > b:
            print("a > b")
        else:
            print("a !> b")
        b = a
    g(b, a)
f(b, a)
print("b is " + str(b))
```

[4 marks]

B.

```
num = "12345"
j=0
for i in num:
    j += 1
print(j, num in (num))
```

[4 marks]

C.

```
def grade(score):
    if (score < 40):
        print("too bad: fail!")
    elif (score > 40):
        print("okay, just pass - work hard!")
    elif (score > 70):
        print("good, well done!")
    if (score > 80):
        print("Prof would be proud!")
    elif (score > 90):
        print("Excellent!")
grade(80.5)
```

[4 marks]

D.

```
def small_table():
    for i in "0123":
        for j in range(1,int(i)):
            print(i + " x " + str(j) + " = " + i*j)
small_table()
```

[4 marks]

E.

```
def generator(seed):
    def gen(bag, seed):
        if (seed == ""):
            if (bag==""):
                print("That's all")
            else:
                print(bag)
        else:
            gen(bag+seed[0], seed[1:])
            gen(bag, seed[1:])
    gen("", seed)
generator("123")
```

[4 marks]

F.

```
c = 5
def f(a):
    def g(a):
        d = a + c
        print(d)
    return g
f(3)(4)
```

[4 marks]

Question 2: Recurrence Relation [18 marks]

Consider the sequence G generated with the following equation:

$$G(n) = G(n-1) + 2\,G(n-2) - 2\,G(n-3)$$

where $G(0) = 1, G(1) = 2,$ and $G(2) = 3$. The series generated by $G(n)$, starting with $G(0)$ and so on is as follows:

$$1, 2, 3, 5, 7, 11, 15, 23, 31, \cdots$$

A. Provide a recursive implementation of G with an input n. [4 marks]

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

Time:

Space:

These are marked according to what you did for the previous Part. So, marks are still awarded even though your algorithm is incorrect. Likewise, no mark is given for getting the same as our solutions here if they do not correspond to your given algorithm.

C. Provide an iterative implementation of G with an input n. [4 marks]

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

Time:

Space:

These are marked according to what you did for the previous Part. So, marks are still awarded even though your algorithm is incorrect. Likewise, no mark is given for getting the same as our solutions here if they do not correspond to your given algorithm.

Looking harder at the equation, you notice the following pattern...

$$\begin{aligned}
G(n) &= G(n-1) + 2\,G(n-2) - 2\,G(n-3)\\
G(n-1) &= G(n-2) + 2\,G(n-3) - 2\,G(n-4)\\
G(n-2) &= G(n-3) + 2\,G(n-4) - 2\,G(n-5)\\
\vdots &= \vdots\\
G(3) &= G(2) + 2\,G(1) - 2\,G(0)
\end{aligned}$$

E. Provide a recursive implementation of G for input n that is significantly faster than what you wrote in Part A above. [4 marks]

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

Time:

Space:

These are marked according to what you did for the previous Part. So, marks are still awarded even though your algorithm is incorrect. Likewise, no mark is given for getting the same as our solutions here if they do not correspond to your given algorithm.

Question 3: Higher Order Functions [24 marks]

A. In this question, you will define a few versions of a function `sum_cross_fib` with the following higher-order functions:

```
def fib(n):
    # fibonacci sequence is: 1, 1, 2, 3, 5, 8, 11, 19, ...
    if n < 2:
        return 1
    else:
        return fib(n-1) + fib(n-2)

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

def back_sum(term, a, back, b):
    if a > b:
        return 0
    else:
        return term(b) + back_sum(term, a, back, back(b))

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

The function `sum_cross_fib` takes an input tuple τ (of numbers) to compute the sum of the product of each i^{th} number in τ with the corresponding i^{th} fibonacci number:

$$\texttt{sum\_cross\_fib}(\tau) = \sum_{i=0}^{\texttt{len}(\tau)-1} \tau[i] \times \texttt{fib}(i)$$

Example execution:

```
>>> sum_cross_fib((1,2,3))  # 1x1 + 2x1 + 3x2 = 9
9

>>> sum_cross_fib((5, 4, 3, 2, 1)) # 5x1 + 4x1 + 3x2 + 2x3 + 1x5 = 26
26
```

Firstly, we define it with `sum` as follows:

```
def sum_cross_fib(t):
    return sum(<T1>,<T2>,<T3>,<T4>)
```

Please provide possible implementations for T1, T2, T3, and T4. [6 marks]

`<T1>`: [2 marks]

`<T2>`: [1 marks]

`<T3>`: [2 marks]

`<T4>`: [1 marks]

B. Secondly, we define it with `back_sum` as follows:

```
def sum_cross_fib(t):
    return back_sum(<T5>,<T6>,<T7>,<T8>)
```

Please provide possible implementations for T5, T6, T7, and T8. [6 marks]

`<T5>`: [2 marks]

`<T6>`: [1 marks]

`<T7>`: [2 marks]

`<T8>`: [1 marks]

C. Thirdly, we define it with `fold` as follows:

```
def sum_cross_fib(t):
    return fold(<T9>,<T10>,<T11>)
```

Please provide possible implementations for T9, T10, and T11. [5 marks]

`<T9>`: [2 marks]

`<T10>`: [2 marks]

`<T11>`: [1 marks]

D. We next consider another function called `cross_tuples` that takes as input two equal length tuples τ_1 and τ_2 (of numbers) to compute the product of their corresponding numbers:

$$\texttt{cross\_tuples}(\tau_1,\tau_2) = (\tau_1[0]\times\tau_2[0], \tau_1[1]\times\tau_2[1], ..., \tau_1[\texttt{len}(\tau_1)-1]\times\tau_2[\texttt{len}(\tau_2)-1])$$

Example execution:

```
>>> t1 = (5, 4, 3, 2, 1, 4, 6)
>>> t2 = (1, 2, 3, 4, 5, 6, 7)
>>> cross_tuples (t1, t2)
(5, 8, 9, 8, 5, 24, 42)
```

Define `cross_tuples` in terms of one of `sum`, `back_sum` or `fold` (pick a one will do). For example, perhaps

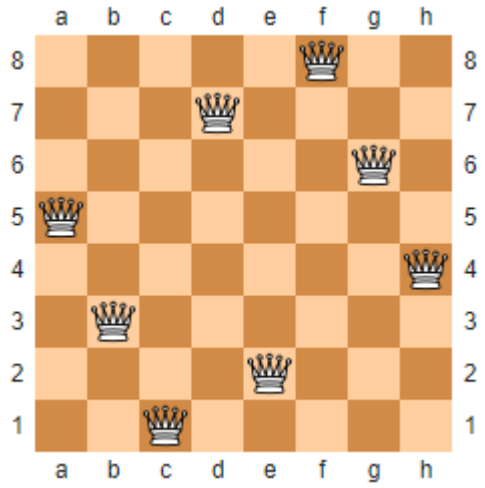
```
def cross_tuples(t1, t2):
    return sum(<T12>,<T13>,<T14>,<T15>)
```

where T12, T13, T14 and T15 are appropriate expressions. Or, perhaps not? In other words, the answer should be just one line of code. [7 marks]

```
def cross_tuples(t1, t2): # assume t1 and t2 are of the same length
```

Question 4: N Queens Problem [31 marks]

Chess composer Max Bezzel published the eight queens puzzle in 1848. Franz Nauck published the first solutions in 1850. The eight queens puzzle is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other; thus, a solution requires that no two queens share the same row, column, or diagonal. One solution to the eight queens problem is shown below. (Source: Wikipedia)



In this problem, you will design the chessboard required to solve a more general n-queens problem. In particular, you will implement the following functions:

1. `make_board` takes in an integer $n > 0$ and creates an empty $n \times n$ chessboard.
2. `add_queen` takes a board, a row and a column and returns a new board with a new queen at the specified position. The indices start at 0. If there is already a queen in the specified position, this function will return `False`.
3. `num_queens` takes a board and returns the number of queens on the board.
4. `size` takes a board and returns the width (or length) of the board.

You are advised to read through all the requirements for this question carefully before deciding on the implementation for your board object.

A. Decide on an implementation for the board object and implement `make_board`. Describe how the state is stored in your implementation and explain how you will track changes to the board. [5 marks]

Note: You are limited to using **tuples** for this question, i.e. you cannot use lists and other Python data structures.

```
def make_board(n):
```

Consider the following sample execution:

```
>>> b1 = make_board(4)
>>> b2 = add_queen(b1,0,1)
>>> b3 = add_queen(b2,1,3)
```

B. Draw the box-and-pointer diagram for your implementation corresponding to the board `b3`. [4 marks]

C. Implement the function `add_queen` that takes a board, a row and a column and returns a new board with a new queen at the specified position. The indices start at 0. If there is already a queen in the specified position, `add_queen` will return `False`. [5 marks]

```
def add_queen(b,r,c)
```

D. Implement the function `num_queens` that takes a board and returns the number of queens on the board. [3 marks]

```
def num_queens(b):
```

E. Implement the function `size` that takes a board and returns the width (or length) of the board. [2 marks]

```
def size(b):
```

F. Implement the function `simple_solved` that takes a board and returns `True` if there are n queens and none of the queens threaten each other either vertically or horizontally. Or `False`, otherwise. Note that this checks if the board partially solves the N Queens problem. [6 marks]

Example execution:

```
>>> b1 = make_board(4)
>>> b2 = add_queen(b1,0,1)
>>> b3 = add_queen(b2,1,3)
>>> b4 = add_queen(b3,2,2)
>>> b5 = add_queen(b4,3,0)
>>> simple_solved(b5)
True
```

```
def simple_solved(b):
```

G. Implement the function `is_solved` that takes a board and returns `True` if the board solves the N Queens problem, i.e. no two queens share the same row, column, or diagonal. Or `False`, otherwise. [6 marks]

Example execution:

```
>>> b1 = make_board(4)
>>> b2 = add_queen(b1,0,1)
>>> b3 = add_queen(b2,1,3)
>>> b4 = add_queen(b3,2,2)
>>> b5 = add_queen(b4,3,0)
>>> is_solved(b5)
False

>>> b6 = add_queen(b3,2,0)
>>> b7 = add_queen(b6,3,2)
>>> is_solved(b7)
True
```

```
def is_solved(b):
```

Question 5: Why Are You Here? [3 marks]

You have successfully made it to college. What do you hope to get out of your NUS education? How do you think what you have learnt in CS1010X will help you achieve that goal? Explain.

Scratch Paper

— E N D O F P A P E R —