CS1010X — Programming Methodology
School of Computing
National University of Singapore

Midterm Test

31 March 2018 **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 **NINETEEN (19) 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.

```
x = 3
y = 5
def f(x,y):
    x = 7
    def g(x):
        return y
    if x > y:
        print(f(y,x))
    else:
        print(f(y-x,x))
f(y,x)
```

[4 marks]

B.

```
x = 5
y = 7
if y//x>=1:
    print("I")
if (x+y)%5>2:
    print("love")
elif (y**2)%4==2:
    print("CS1010X")
else:
    print("midterms")
```

[4 marks]

C.

```
result = ()
for i in range(6):
    if i%2==1:
        result = result + (i,)
    else:
        result = (result)*2
        i = i+3
print(result)
```

[4 marks]

D.

```
def dozo(x):
    return 2*x if x else "Cool"

print(dozo(1))
print(dozo("False"))
print(dozo(()))
print(()==False)
```

[4 marks]

E.

```
a = 5
b = 3
def foo(a):
    def b(a):
        return a**3
    return b
print(foo(a)(b))
```

[4 marks]

F.

```
a = "holy"
b = "cow"
def potato(x,y):
    result = ""
    for i in x:
        for j in y:
            if i>j:
                result += i
    return result
c = potato(a,b)
print(c[3:25])
```

[4 marks]

Question 2: Fibonacci Sum [18 marks]

By now, you should be familiar with the Fibonacci series:

$$0, 1, 1, 2, 3, 5, 8, 13, \cdots$$

where $fib(0) = 0$, $fib(1) = 1$, and $fib(n) = fib(n-1) + fib(n-2)$ for $n \geq 1$,

A. **[Warm Up]** Provide an iterative implementation of `fib`. [2 marks]

```
def fib(n):
```

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? [2 marks]

Time:

Space:

C. Consider the following sum

$$fib(0) + fib(1) + \cdots + fib(n)$$

which is the sum of the first $n+1$ Fibonacci numbers, which we will call `fib_sum`. Provide a **recursive** implementation of `fib_sum`. [3 marks]

```
def fib_sum(n):
```

D. Assuming iterative `fib`, what is the order of growth in terms of time and space for the function you wrote in Part (C) in terms of n. Explain your answer. [2 marks]

Time:

Space:

E. Provide an **iterative** implementation of `fib_sum`. [3 marks]

```
def fib_sum(n):
```

F. Assuming iterative `fib`, what is the order of growth in terms of time and space for the function you wrote in Part (E) in terms of n. Explain your answer. [2 marks]

Time:

Space:

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

$$\begin{aligned} fib(n) &= fib(n+2) - fib(n+1) \\ fib(n-1) &= fib(n+1) - fib(n) \\ fib(n-2) &= fib(n) - fib(n-1) \\ \vdots &= \vdots \\ fib(1) &= fib(3) - fib(2) \\ fib(0) &= fib(2) - fib(1) \end{aligned}$$

G. Provide an implementation of `fib_sum` that is significantly faster than what you wrote in Parts (C) and (E) above. [2 marks]

```
def fib_sum(n):
```

H. Assuming iterative `fib`, what is the order of growth in terms of time and space for the function you wrote in Part (G) in terms of n. Explain your answer. [2 marks]

Time:

Space:

Question 3: Higher Order Functions [25 marks]

A. We note the we can define `fib_sum(n)` from Question 2 in terms of `sum` (see Appendix) as follows:

```
def fib_sum(n):
    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. We note the we can define `fib_sum(n)` from Question 2 in terms of `fold` (see Appendix) as follows:

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

Please provide possible implementations for T5, T6, and T7. [5 marks]

`<T5>`: [2 marks]

`<T6>`: [2 marks]

`<T7>`: [1 marks]

Consider the following Python code:

```
def twice(f):
    return lambda x: f(f(x))

def thrice(f):
    return lambda x: f(f(f(x)))

def double(x):
    return 2*x

twice_double = twice(double)
thrice_double = thrice(double)
```

C. Provide an implementation of the function `repeated(f,n)` so that we would be able to define `twice_double` and `thrice_double` as follows:

```
twice_double = repeated(double,2)
thrice_double = repeated(double,3)
```

[6 marks]

```
def repeated(f,n):
```

Given 2 functions f and g, we want to construct the following composite function with a Python function `dual_function(f,g,n)`:

$$f(g(f(g(\cdots(x))\cdots)$$

where functions f and g are applied alternately n times in total, not each. The leftmost (outer) function is always f. In other words, `dual_function(f,g,4)` will produce:

```
lambda x: f(g(f(g(x))))
```

while `dual_function(f,g,7)` will produce:

```
lambda x: f(g(f(g(f(g(f(x)))))))
```

D. Provide an implementation of the function `dual_function(f,g,n)`. [6 marks]

```
def dual_function(f,g,n):
```

E. Define `repeated(f,n)` in terms of `dual_function(f,g,n)`. [2 marks]

```
def repeated(f,n):
```

Question 4: Room Booking System [30 marks]

Your company has a number of meeting rooms but everyone wants to use them and the current system involves pre-booking time slots for the meeting rooms on a sheet of paper that is stuck on the wall. Your boss found out that you are currently taking CS1010X and asks you to write a room booking system for your company. You decided that this is the opportunity to impress him with your coding skills. A record of the room bookings is called a schedule.

To fully satisfy the requirements for this question, you are advised to read through all the subquestions before you start. Your answer for Part(A) will affect your answer to subsequent sub-problems. If you do not make a wise decision in Part(A), you might end up having a really hard time with the later subquestions.

You are required to implement the following 6 functions in this problem:

1. `create_schedule` takes in a tuple of rooms (which are strings) and returns a new empty schedule for the rooms.
2. `is_schedule` takes a object and returns `True` if it is a valid schedule, or `False` otherwise.
3. `get_venues` returns a list of the rooms that is managed by the schedule.
4. `reserve` will take a room, a range of time slots and return a new schedule with the specified time slots reserved. Operation fails with `False` if one of the slots is already reserved.
5. `is_reserved` takes a room and time slot returns `True` if it is already reserved, or `False` otherwise.
6. `remove_reservation` returns a new schedule with the specified time slot removed.

Example execution:

```
>>> s1 = create_schedule(("LT15", "LT2", "LT19"))
>>> get_venues(s1)
('LT15', 'LT2', 'LT19')

>>> is_schedule(s1)
True
>>> is_schedule(())
False
>>> is_schedule("Mambo")
False

>>> is_reserved(s1, "LT15", 4)
False

>>> s2 = reserve(s1,"LT15",3,5)
>>> get_venues(s2)
('LT15', 'LT2', 'LT19')
>>> is_schedule(s2)
True
```

```
>>> is_reserved(s2, "LT15", 4)
True
>>> reserve(s1,"LT15",3,5)
False

>>> s3 = remove_reservation(s2, "LT15", 4)
>>> is_schedule(s3)
True
>>> is_reserved(s3, "LT15", 4)
False
```

A. Decide on an implementation for the schedule object and implement `create_schedule`. Describe how the state is stored in your implementation and explain. [5 marks]

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

```
def create_schedule(rooms):
```

B. Implement the function `is_schedule(s)` that returns `True` if `s` is a valid schedule, or `False` otherwise. [4 marks]

```
def is_schedule(obj):
```

C. Implement the function `get_venues(s)` that returns a tuple that contains the list of rooms for a schedule. [3 marks]

```
def get_venues(schedule):
```

D. Implement the function `reserve(schedule, room, start, end)` that will return a new schedule where the specified room is reserved for time slots from *start* to *end* (inclusive). You can assume that both *start* and *end* are non-negative integers and that $start \leq end$. If some of the slots in the specified time slots are already reserved, return `False`. [7 marks]

```
def reserve(schedule, room, start, end):
```

E. Implement the function `is_reserved(schedule, room, slot)` that returns `True` if it is already reserved, or `False` otherwise. [4 marks]

```
def is_reserved(schedule, room, slot):
```

F. Implement the function `remove_reservation(schedule, room, slot)` that will return a new schedule where the specified is removed, or the original schedule if the slot to remove is not occupied. [7 marks]

```
def remove_reservation(schedule, room, slot):
```

Question 5: Draw Something! [3 marks]

You have been taking CS1010X for about 3 months now. Draw(!) something below that best expresses the most important thing(s) that you have learnt in the class thus far.

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 product(term, a, next, b):
  if a > b:
    return 1
  else:
    return term(a) * product(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 —