CS1010X — Programming Methodology
School of Computing
National University of Singapore

Re-Midterm Test

16 April 2016 **Time allowed:** 1 hour 45 minutes

Student No:

A								

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 [30 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 = "Happy"
b = "Re-midterm!"
flag = 1
while (a or b):
    if flag:
        b += a[0]
        a = a[1:]
        flag = 0
    else:
        a += b[0]
        b = b[1:]
        flag = 1
print(a + b)
```

[5 marks]

B.

```
def h(y):
    return lambda x: y[0](y[1](y[2](x)))
e = lambda x: x + 2
f = lambda y: y**2
g = lambda z: z//2
print(h((f,e,g))(2))
```

[5 marks]

C.

```
x = 6
y = 9
def g(y):
    x = 2*y
    def f(x):
        return x - y
    return f
def f(x):
    y = 4
    return g(x-y)(x)
print(f(x))
```

[5 marks]

D.

```
a = (1,2,3,4)
b = a
c = (1,2,3,4)+()
print(a is b, b is c, c is a, a == b, b == c, c == a)
```

[5 marks]

E.

```
tup = (1,2)
counter = 0
for i in range(3,11):
    if counter == 2:
        tup = tup[2]
        counter = 0
    else:
        tup = tup + (tup,)
        counter += 1
print(tup[3][2][1])
```

[5 marks]

F.

```
once = lambda f: lambda x: f(x)
twice = lambda f: lambda x: f(f(x))
thrice = lambda f: lambda x: f(f(f(x)))
print(thrice(twice)(once)(lambda x: x + 2)(9))
```

[5 marks]

Question 2: Numbers of Love [22 marks]

Amicable numbers are the couples of the number world. To find an amicable number pair, you just have to find one, and the other can be found by the sum of the proper divisors of the first. Proper divisors are all the factors of the number excluding itself but including 1.

For example, (220, 284) is the first pair of amicable numbers. Proper divisors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 and 110, which sums up to 284. Doing the same to 284 must produce its inseparable other, 220.

A. **[Warm up]** Implement the function `find_divisors` that takes as input a positive number *n* and returns a tuple containing all its divisors (including 1). [4 marks]

```
def find_divisors(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*. Explain your answer. [4 marks]

Time:

Space:

C. Implement the function `pair` that takes as input an amiable number *n* and returns its pair. You can assume that *n* is amiable. [4 marks]

```
def pair(n):
```

D. Implement the function `has_amiable` that takes as input two integers *a* and *b*, such that $a < b$ and returns `True` if there is an amiable number in the range of integers between *a* and *b* (inclusive), or `False` otherwise. [4 marks]

```
def has_amiable(a,b):
```

E. Implement the function `find_k_amiable` that takes as input a positive integer *k* and returns the first *k* pairs of amiable numbers as a tuple of tuple pairs. [**Hint:** Make sure you deal with potential repeats.] [6 marks]

Example execution:

```
>>> find_k_amiable(1)
((220, 284),)

>>> find_k_amiable(2)
((220, 284), (1184, 1210))

>>> find_k_amiable(3)
((220, 284), (1184, 1210), (2620, 2924))
```

```
def find_k_amiable(k):
```

Question 3: Reversing Higher-Order Sums [17 marks]

We discussed the following higher order function `sum` in class:

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

For example,

```
sum(lambda x: x, 1, lambda x: x + 1, 5) = 1 + 2 + 3 + 4 + 5

sum(lambda x: x**2, 1, lambda x: x + 1, 5) = 1 + 4 + 9 + 16 + 25
```

A. [Warm up] Implement the function `get_indices` that given *a*, `next` and *b* will return a list of all the indices of the terms in the sum. [4 marks]

Example execution:

```
>>> get_indices(1, lambda x: x+1, 10)
(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

>>> get_indices(1, lambda x: 2*x, 5)
(1, 2, 4)

>>> get_indices(1, lambda x: 2**x, 30)
(1, 2, 4, 16)
```

```
def get_indices(a,next,b):
```

It turns out that we can also compute the sum backwards, if we only knew the corresponding `back` function,

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

B. Express the function `back_sum` in terms of the `sum` function, i.e.

```
def back_sum(term, a, back, b):
    return sum(<T1>, <T2>, <T3>, <T4>)
```

You can assume that the *a* and *b* are specially chosen so that the `back` function will end exactly at *a* when repeated applied on *b*. [5 marks]

```
def back_sum(term, a, back, b):
```

C. Express the function `sum` in terms of the `back_sum` function, i.e.

```
def sum(term, a, next, b):
    return back_sum(<T1>, <T2>, <T3>, <T4>)
```

As before, you can assume that the `next` function stops exactly at *b* before exceeding it when repeatedly applied to *a*. [4 marks]

```
sum(term, a, next, b):
```

D. Suppose we defined the functions `back_sum` and `sum` according to the new definitions in Parts (B) and (C), what is the output of `sum(lambda x: x, 0, lambda x: x+1, 10)`? [4 marks]

Question 4: Lambda Sets! [28 marks]

Your job is to implement a new container that we call a lambda set that is used to store functions (lambdas). It has 8 associated functions:

1. `make_lambda_set` creates a new empty lambda set.
2. `is_lambda_set` takes one argument and returns `True` if the argument is a lambda set, or `False` otherwise.
3. `add_lambda` takes a lambda set s and an object x and returns a new lambda set that contains x, if x is a function (lambda), or the original (unchanged) lambda set s otherwise.
4. `same_set` takes two lambda sets s_1 and s_2 and returns `True` if they are the same set (though they might contain different elements), i.e. if they were created from the same original empty lambda set created by `make_lambda_set`.
5. `size` takes a lambda set s and returns the number of functions it contains.
6. `contains` takes a lambda set s and an object x and returns `True` if s contains x, or `False` otherwise.
7. `has_duplicate` takes a lambda set s and returns `True` if it contains at least 2 copies of the same object.
8. `max_duplicate` takes a lambda set s and returns the maximum number of duplicate objects in the set.

Example execution (please study carefully to understand how lambda sets work):

```
>>> s = make_lambda_set()
>>> is_lambda_set(s)
True

>>> s2 = add_lambda(s,4)
>>> same_set(s,s2)
True

>>> size(s2)
0

>>> s3 = add_lambda(s,lambda x:x+1)
>>> same_set(s,s3)
True

>>> size(s3)
1

>>> t = make_lambda_set()
>>> same_set(s,t)
False

>>> same_set(s3,t)
False
```

```
>>> add = lambda x:x+1
>>> contains(s3,add)
False

>>> s4 = add_lambda(s3,add)
>>> contains(s4,add)
True

>>> has_duplicate(s4)
False

>>> size(s4)
2

>>> max_duplicate(s4)
1

>>> s5 = add_lambda(s4,add)
>>> has_duplicate(s5)
True

>>> max_duplicate(s5)
2

>>> s6 = add_lambda(s5,add)
>>> max_duplicate(s6)
3
```

A. Decide on an implementation for the lambda set object and implement `make_lambda_set`. Describe how the state is stored in your implementation as lambdas are added to it. [4 marks]

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

```
def make_lambda_set():
```

B. Implement the function `is_lambda_set(s)` that returns `True` if `s` is a lambda set, or `False` otherwise. [3 marks]

```
def is_lambda_set(s):
```

C. Implement the function `add_lambda(s,x)` that will return a new lambda set containing *x* if *x* is a function or *s* otherwise. [3 marks]

```
def add_lambda(s,x):
```

D. Implement the function `same_set(s1,s2)` that takes two lambda set s_1 and s_2 and returns `True` if they were created from the same original empty lambda set, or `False` otherwise. [3 marks]

```
def same_set(s1,s2):
```

E. Implement the function `size(s)` that takes a lambda set s and returns the number of functions it contains. [3 marks]

```
def size(s):
```

F. Implement the function `contains(s,x)` that returns `True` if lambda set `s` contains *x*, or `False` otherwise. [4 marks]

```
def contains(s,x):
```

G. Implement the function `has_duplicate(s)` that returns `True` if `s` contains more than one copy of some function, or `False` otherwise. Note that we can use the `is` operator to check if two functions are the same. [4 marks]

```
def has_duplicate(s):
```

H. Implement the function `max_duplicate(s)` that returns the maximum number of duplicates in the set *s*. [4 marks]

```
def max_duplicate(s):
```

Question 5: Redemption [3 marks]

List 3 new things that you have learnt since the midterms that you think will help you to do better this time. Explain.

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 —