CS1010X — Programming Methodology
School of Computing
National University of Singapore

Re-Midterm Test

21 April 2018 **Time allowed:** 1 hour 45 minutes

Student No:	S	O	L	U	T	I	O	N	S

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.

```
def bar(x,y,z):
    return (x,)+y+(z,)
a = bar(1,((1,2),3), (1,2))
print(bar(a[:1],a[:2],a[3:]))
```

[4 marks]

```
((1,), 1, (1, 2), ((1, 2),))
```

This is to test that the student really understands tuple concatenation and slicing.

```
a => (1, (1, 2), 3, (1, 2))
a[:1] => (1,)
a[:2] => (1, (1, 2))
a[3:] => ((1, 2),)
bar(a[:1],a[:2],a[3:]) => ((1,),) + (1, (1, 2)) + (((1, 2),),)
= ((1,), 1, (1, 2), ((1, 2),))
```

B.

```
result = 1
for j in range(10):
    for i in range(j):
        if i+j%10==0:
            result += i
        elif j>5:
            result = 2*result
            break
print(result)
```

[4 marks]

```
16
```

This is to test that students can trace nested `for` loops and reason about them. Should be clear that `j` goes from 0 to 9 and `i` goes from 0 to one less than `j`. First line of `if` says `i+j` is divisible by 10. That cannot happen until `j` is at least 6, but once `j` is 6, it activates the 2nd conditional and `break`s. Basically, we double result for $j = 6,7,8,9$, so the result is $2^4 = 16$. 2 points in some cases where student reaches $j = 6$ and stopped there so they gave 2 as an answer, but working must be shown. No credit for just “2”.

C.

```python
x = 1
y = 2
z = 3
def f(x):
    y = 1
    return y + 2*x
def g(z):
    def h(x):
        return f(z) + y
    return h(y)
print(g(f(y)))
```

[4 marks]

```
13
```

This is to test that the student has a very clear understanding of scope and how variables are bounded when functions are called.

2 points for “9”. This means there’s a mistake at $g(5)$.

D.

```python
def hahaha(x):
    x = x*2
    y = ""
    while x:
        y = y+x[1:2]
        x = x[3:]
    return y
print(hahaha("12345"))
```

[4 marks]

```
253
```

This is to test that the student understands string slicing and that slicing out of index doesn’t produce an error unlike indexing. The following is how x and y evolves in the loop

```
x = "1234512345", y = ""
x = "4512345",    y = "2"
x = "2345",       y = "25"
x = "5",          y = "253"
```

E.

```
print(sum(lambda x: 2*x, 1, lambda x: 3*x, 10))
```

where `sum` is given in the Appendix. [4 marks]

```
26
```

This is to test that the students understand how `sum` works when `next` is not `lambda x:x+1`. We are summing `f(1) + f(3) + f(9)` where $f(x) = 2x$.

F.

```
f = lambda y, x: y(y(x))
g = lambda x: lambda y: x(x(x(y)))
print(f(g,lambda x:x+2)(4))
```

[4 marks]

```
22
```

This is to test that the students understand `lambda`. Denote `lambda x:x+2` with `h`

```
f(g,lambda x:x+2)(4)
=> f(g,h)(4)
=> g(g(h))(4)
=> g(lambda y: h(h(h(y))))(4) # Let a = lambda y: h(h(h(y)))
=> g(a)(4)
=> (lambda y: a(a(a(y))))(4)
=> a(a(a(4))), but a = lambda x: x+6
=> 22
```

Question 2: More Fibonacci (when you're not expecting...) [23 marks]

Consider the following functions:

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

def fob(n):
a,b = 2,1
    for i in range(n):
        a,b = b,a+b
    return a
```

which generate the following series:

n	0	1	2	3	4	5	6
fib(n)	0	1	1	2	3	5	8
fob(n)	2	1	3	4	7	11	18

A. [Warm Up] Write a function `alternating` that takes in 2 serieses and an integer `n` and returns terms from the 2 serieses in alternating orders starting from the first series. [3 marks]

Example execution:

```
>>> alternating(fib,fob,0)
0
>>> alternating(fib,fob,1)
2
>>> alternating(fib,fob,2)
1
>>> alternating(fib,fob,3)
1
```

```
def alternating(f,g,n):
    if n%2==0:
        return f(n//2)
    else:
        return g(n//2)
```

B. Consider the following sum

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

which is the sum of the first $n+1$ Fibonacci numbers. Write a function `series_sum` that takes in a series `f` and produces the sum of the series. In other words, we can define the sum of the first $n+1$ Fibonacci numbers `fib_sum` as follows:

```
>>> fib_sum = series_sum(fib)
>>> fib_sum(0)
0
>>> fib_sum(1)
1
>>> fib_sum(2)
2
>>> fib_sum(3)
4
```

[5 marks]

```
def series_sum(f):
    def helper(n):
        if n == 0:
            return f(0)
        else:
            return f(n)+helper(n-1)
    return helper
```

-2 points for returning `helper(x)` instead of `helper`.
-3 points for returning a function. No error carry forward for Part (C).

C. What is the order of growth in terms of time and space for the function you wrote in Part (B) in terms of n, assuming that `f` is $O(1)$ for both time and space? [2 marks]

Time: $O(n)$

Space: $O(n)$

D. If your implementation of `series_sum` in Part (B) was recursive, provide also the iterative one here. If it was iterative, please provide the recursive version. [5 marks]

```
def series_sum(f):
    def helper(n):
        result = 0
        for i in range(n+1):
            result += f(i)
        return result
    return helper
```

E. What is the order of growth in terms of time and space for the function you wrote in Part (D) in terms of n, assuming that `f` is $O(1)$ for both time and space? [2 marks]

Time: $O(n)$

Space: $O(1)$

F. Given a series `f`, we say that a number is a series sum of `f` if it is equivalent to the sum of the first k terms for some integer k. Write the function `is_series_sum` that takes the function for a series and an integer n and returns `True` if n is a series sum of the series, or `False` otherwise. You can assume that all the terms of the serious `f` are non-negative. [6 marks]

```
>>> is_series_sum(fib,2) # fib_sum(2) = 2
True
>>> is_series_sum(fib,3)
False
>>> is_series_sum(fib,4) # fib_sum(3) = 4
True
>>> is_series_sum(fib,5) # fib_sum(4) = 7 > 5
False
```

```
def is_series_sum(f,n):
    my_sum = series_sum(f)
    i = 0
    current_sum = my_sum(0)
    while current_sum<n:
        i += 1
        current_sum = my_sum(i)
    return current_sum == n
```

This question checks if the student understood `is_fib`.

Question 3: Higher Order Functions [20 marks]

A. **[Warm Up]** The function `alternating_series` takes in 2 serieses returns a new series with terms that alternate between the 2 series. Express `alternating_series` in terms of `alternating` from Question 2 Part (A) [3 marks]

Example execution:

```
>>> f2 = alternating_series(fib,fob)
>>> f2(0) # fib(0)
0
>>> f2(1) # fob(0)
2
>>> f2(2) # fib(1)
1
>>> f2(3) # fob(1)
1
>>> f2(4) # fib(2)
1
>>> f2(5) # fob(2)
3
>>> f2(6) # fib(3)
2
>>> f2(7) # fob(3)
4
```

```
def alternating_series(f,g):
    return lambda n: alternating(f,g,n)
```

Zero if student just returns `alternating(f,g,n)`.

B. The function `odd_sum` takes in a series `f` and a positive integer `n` and returns the sum of the first odd-indexed terms for the series.

Example execution:

```
>>> odd_sum(f2,1) # f2(1) = 2
2
>>> odd_sum(f2,2) # f2(1) + f2(3)
3
>>> odd_sum(f2,3) # f2(1) + f2(3) + f2(5)
6
>>> odd_sum(f2,4) # f2(1) + f2(3) + f2(5) + f2(7)
10
```

We note the we can define `odd_sum(n)` in terms of `sum` (see Appendix) as follows:

```
def odd_sum(f,n):
    return sum(<T1>,<T2>,<T3>,<T4>)
```

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

`<T1>`: [2 marks]	`f`
`<T2>`: [1 marks]	`1`
`<T3>`: [2 marks]	`lambda n:n+2`
`<T4>`: [1 marks]	`2*n`

C. The function `even_sum` takes in a series `f` and a positive integer `n` and returns the sum of the first *n* even-indexed terms for the series.

Example execution:

```
>>> even_sum(f2,1) # f2(0) = 0
0
>>> even_sum(f2,2) # f2(0) + f2(2)
1
>>> even_sum(f2,3) # f2(0) + f2(2) + f2(4)
2
>>> even_sum(f2,4) # f2(0) + f2(2) + f2(4) + f2(6)
4
```

Express `even_sum` in terms of `odd_sum` using your implementation in Part (B). If it is not possible, explain. [3 marks]

```
def even_sum(f,n):
    return odd_sum(lambda x: f(x-1),n)
```

No credit if Part (B) is wrong since clearly won't be able to express `even_sum` in terms of `odd_sum`. Another possible solution is:

```
    return series_sum(f)(n) - odd_sum(f,n)
```

D. We can also define `odd_sum(n)` in terms of `fold` (see Appendix) as follows:

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

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

`<T5>:` [2 marks]	`lambda x,y:x+y`
`<T6>:` [2 marks]	`lambda n: f(n) if n\%2==1 else 0` or `lambda n: f(2*n+1)`
`<T7>:` [1 marks]	`2*n` or `n-1`

E. Is it possible to express `even_sum` in terms of `odd_sum` using your implementation in Part (D)? If so, do it. If not, explain. [3 marks]

For our implementation in Part(D), we can use the answer from Part (C).

Question 4: Building Your Own Shopping Cart! [30 marks]

You just joined Ladaza, an up and coming e-commerce company. Guess what? Your new boss found out that you are currently taking CS1010X and decides that you are the best person to implement the company's new budget shopping cart. You decided that this is the opportunity to impress him with your coding skills.

The new online cart isn't quite like a regular shopping cart because it will allow the user to specify a maximum budget. It will not allow new items to be added to it, if adding the item will cause the total cost of the items in the cart to exceed the specified budget.

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 7 functions in this problem:

1. `create_cart` takes in a budget and returns a new empty cart.
2. `get_budget` returns the budget for a cart.
3. `get_items` returns a tuple containing all the items currently in the cart.
4. `is_cart` takes a object and returns `True` if it is a valid cart, or `False` otherwise.
5. `cost` will return the total cost of all the items contained in a cart.
6. `add_to_cart` will return a new car with the new item added, or `False` if the addition fails because it will exceed the budget for the cart.
7. `remove_item(cart, item)` will return a new cart where the specified item is removed, or the original cart if the specified item is not in the cart.

Example execution:

```
>>> my_cart = create_cart(100)
>>> get_budget(my_cart)
100
>>> get_items(my_cart)
()
>>> cost(my_cart)
0

>>> is_cart(my_cart)
True
>>> is_cart("Holy cow!")
False

>>> new_cart =  add_to_cart(my_cart, "Bicycle", 51)
>>> get_budget(new_cart)
100
>>> get_items(new_cart)
('Bicycle',)
>>> cost(new_cart)
51
```

```
>>> is_cart(new_cart)
True

>>> failed_cart =  add_to_cart(new_cart, "Bicyle", 51)
>>> failed_cart
False

>>> good_cart = new_cart
>>> for i in range(7):
    good_cart =  add_to_cart(good_cart, "Chicken", 7)

>>> get_budget(good_cart)
100
>>> get_items(good_cart)
('Bicycle', 'Chicken', 'Chicken', 'Chicken', 'Chicken', 'Chicken',
'Chicken', 'Chicken')
>>> cost(good_cart)
100

>>> better_cart = remove_item(good_cart, "Chicken")
>>> get_budget(better_cart)
100
>>> get_items(better_cart)
('Bicycle', 'Chicken', 'Chicken', 'Chicken', 'Chicken', 'Chicken', 'Chicken')
>>> cost(better_cart)
93

>>> better_cart = remove_item(better_cart, "Bicycle")
>>> get_budget(better_cart)
100
>>> get_items(better_cart)
('Chicken', 'Chicken', 'Chicken', 'Chicken', 'Chicken', 'Chicken')
>>> cost(better_cart)
42

>>> better_cart = remove_item(better_cart, "Bicycle")
>>> better_cart
False
```

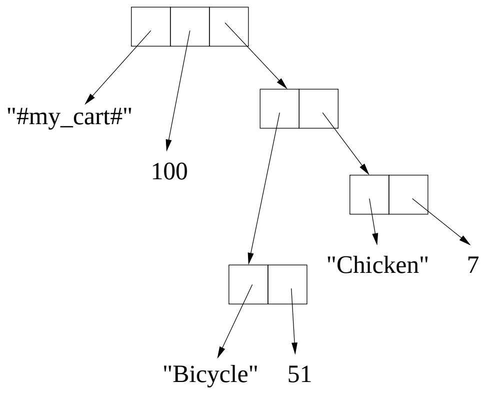
A. Decide on an implementation for the cart object and implement `create_cart`. Describe how the state is stored in your implementation, and draw the box-and-pointer diagram for a cart object with 2 items. [7 marks]

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

There are many possibilities. Here, we store the state as the budget followed by a tuple of (item,price) pairs.

```
def create_cart(budget):
    return("#my_cart#", budget, ())
```

`"#my_cart#"` is the label for the schedule object.



-2 for not keeping track of the prices for individual items.
-2 not doing pointers in the box-and-pointer diagram.

B. Implement the function `get_budget` that returns the budget for a cart. [2 marks]

```
def get_budget(cart):
    return cart[1]
```

C. Implement the function `get_items` that returns the items currently in the cart. [3 marks]

```
def get_items(cart):
    result = ()
    for item, price in cart[2]:
        result += (item,)
    return result
```

D. Implement the function `is_cart(c)` that returns `True` if `c` is a valid cart, or `False` otherwise. [4 marks]

```
def is_cart(obj):
    return type(obj)==tuple and obj != () and obj[0] == "#my_cart#"
```

-2 marks for missing out `type(object) == tuple`.
-1 mark for `type(object) == tuple` at the end of an `and` statement (it needs to be the first thing!).
-2 marks for missing out a label, though this can be forgiven if the student defines a very special structure that can be tested.

E. Implement the function `cost` that will take a cart and return the total cost of the items in the cart. [4 marks]

```
def cost(cart):
    total = 0
    for item, price in cart[2]:
        total += price
    return total
```

F. Implement the function `add_to_cart(cart, item, price)` that will add a new item at the specified price to the cart and return a new card, or `False` if the addition fails because it will exceed the budget for the cart. [4 marks]

```
def add_to_cart(cart, item, price):
    if cost(cart) + price > get_budget(cart):
        return False
    else:
        return cart[:2] + (cart[2] + ((item,price),),)
```

G. Implement the function `remove_item(cart, item)` that will return a new cart where the specified item is removed, or the original cart if the specified item is not in the cart. [6 marks]

```
def remove_item(cart, item):
    def remove(lst, item):
        if lst == ():
            return ()
        elif lst[0][0] == item:
            return lst[1:]
        else:
            return (lst[0],) + remove(lst[1:], item)

    new_lst = remove(cart[2], item)
    if new_lst == cart[2]:
        return cart
    else:
        return cart[:2] + (new_lst,)
```

-3 for not dealing with repeated items correctly, i.e. some students wrote code that would remove all copies of the item in the cart instead of just one.

Question 5: What did you did you learn since last midterm? [3 marks]

If you are taking this exam as a re-exam, tell us what 3 things you learnt from the midterms that has helped you do better this time. If you are taking this exam as your first midterm, tell us what you think are the 3 most important concepts you have learnt in CS1010X thus far – or you can tell us 3 things you learnt from taking this exam.

Student will get points for any reasonably attempt that demonstrates effort. Roughly 1 point is given for each "reasonable" point made.

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 —