CS1010S — Programming Methodology
School of Computing
National University of Singapore

CS1010S Re-Mid-Term Test

20 March 2015 **Time allowed:** 1 hour 45 minutes

Matriculation 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 **FOUR (4) 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		
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. If the interpreter produces an error message, or enters an infinite loop, explain why. Partial marks will be awarded for workings if the final answer is wrong.

A. [5 marks]

```
x = 2
def f(x):
    return g(x) + x
def g(x):
    return x * 2
print(f(g(x ** 2)))
```

24

B. [5 marks]

```
a = 'apple'
b = 'banana'
while a < b:
    a, b = b[1:], a[1:]
print(a + b)
```

```
plenana
```

C. [5 marks]

```
m = 'morning'
n = 'night'
if m < n:
    print('Good ' + m)
else:
    print('Good ' + n)
if len(m) < len(n):
    print('Shorty')
else:
    print('Longy')
```

```
Good morning
Longy
```

D. [5 marks]

```
def foo(x, y):
    return (x,) + y
print(foo(1, foo((2, 3), 4)))
```

```
TypeError: cannot concatenate int to a tuple.
```

E. [5 marks]

```
bar = lambda x: lambda y: y(x)
print(bar(lambda n: n*2)(bar(3)))
```

6

F. [5 marks]

```
j = 0
for i in range(0, 15, 2):
    if j == i:
        continue
    if j >= i:
        break
    if j <= i:
        j += i
print(j)
```

14

Question 2: Budget Baggage [26 marks]

A particular budget airline charges $5 for every 5 kg or part thereof of a passenger's bag. For example, a 7 kg bag will be charged $10 and a 15 kg bag will be charged $15.

A. [Warm-up] Write a function `cost` that takes as input a weight of a bag, and return the luggage charges in dollars. [3 marks]

```
def cost(weight):
    if weight % 5 == 0:
        return weight
    else:
        return 5 * (weight // 5 + 1)
```

B. Now to discourage passengers from carrying too many bags, there is an additional 10% added on top of the bag cost for every additional bag. In other words, the second bag will cost 10% extra, the third bag will cost 20% extra, the fourth bag will cost 30% extra, and so on.

Write a function `cost_n` that takes as inputs, *n*, for the *n*th bag and the weight of the bag, and returns the luggage charges in dollars. For example, `cost_n(2, 10)` will return 11 because of the additional 10% for the second bag and `cost_n(3, 10)` will return 12 because of the additional 20% for the third bag. [3 marks]

Note: For the following sub-questions, you may ignore any floating point inaccuracies in Python.

```
def cost_n(n, weight):
    return (1 + (n-1)/10) * cost(weight)
```

As an employee with new-found Python skills, you are tasked with writing a function to compute the total baggage cost for a passenger, given the weight of each of his bags.

C. Implement an **iterative** function `luggage_cost` that takes as input a tuple of the weight of bags as integers and return the total baggage cost. You should use the function `cost_n` that you wrote in part (B).

You may assume that the tuple has been sorted in decending order to ensure the passenger be billed the minimum charge. [4 marks]

```
def luggage_cost(bags):
    n, cost = 1, 0
    for bag in bags:
        cost += cost_n(n, bag)
        n += 1
    return cost
```

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, where n is the length of the tuple.

Provide a brief explanation for your answer. [4 marks]

Time: $O(n)$. The loop will loop n times and cost_n takes constant time.

Space: $O(1)$. There is no recursive function call so the space needed is constant.

E. Implement a **recursive** version of the function `total_cost` as described in part (C). [4 marks]

```
def luggage_cost(bags):
    def helper(n, b):
        if not b:
            return 0
        else:
            return cost_n(n, b[0]) + helper(n+1, b[1:])
    return helper(1, bags)
```

F. What is the order of growth in terms of time and space for the function you wrote in Part (E) in terms of n, where n is the length of the tuple.

Provide a brief explanation for your answer. [4 marks]

Time: $O(n^2)$. The recursion function is called n times, and the slicing operation in each call takes $O(n-1)$. So in total it take $O(n^2)$ time.

Space: $O(n^2)$. The recursion function is called n times, and each time the slicing operation creates a new tuple of size $n-1$. Thus at the bottom of the recursion there are n delayed operations, which means the largest space needed is $n+(n-1)+(n-2)+\cdots+1=n^2$.

G. Your collegue implemented the sorting function to sort the weight of the bags in descending order using recursion as follows:

```
def selection_sort(bags):
    if bags == ():
        return ()
    else:
        i = heaviest(bags)  #returns the index of the heaviest bag
        return (bags[i],) + selection_sort(bags[:i] + bags[i+1:])
```

You realized that by simply translating the function to an iterative version, you can reduce the order of growth in terms of space from $O(n^2)$ to $O(n)$.

Write an **iterative** version of the `selection_sort` function as stated above. Note that this function is different from the one shown in lecture. [4 marks]

```
def selection_sort(bags):
    result = ()
    while bags:
        i = heaviest(bags)
        result += (bags[i],)
        bags = bags[:i] + bags[i+1:]
    return result
```

Question 3: Higher-Order Baggage [18 marks]

Motivated by your success in solving the airline's baggage cost calculations, you decided to resign and start your own company providing baggage costing solutions to other airlines.

You soon realize that other airlines use different methods of computing the baggage cost. For instance, Python Airlines charges for each bag, \$1 per kg for the first 10 kg, \$2 per kg for the next 10 kg, \$3 per kg for the next 10 kg and so on. They do **not** have extra cost for each new bag.

For example, a bag weighing 14 kg will cost $\$1 \times 10 + \$2 \times 4 = \$18$, and a bag weighing 26 kg will cost $\$1 \times 10 + \$2 \times 10 + \$3 \times 6 = \$48$.

A. Modify the function `cost_n` from the previous question to reflect the new baggage calculation method of Python Airlines. Name your function `pa_cost_n`.

Hint: Recall that Python Airlines has no use for `n`.

Another Hint: Do not try to solve this in a "mathemathical" manner. [6 marks]

```
def pa_cost_n(n, weight):
    cost = 0
    multiplier = 1
    while weight > 10:
        cost += multiplier * 10
        multiplier += 1
        weight -= 10
    cost += multiplier * weight
    return cost
```

B. After writing a couple of separate functions for each airline, you realized they have the same pattern. It dawns on you that you can use higher-order functions to simplify the task.

You decided to implement a higer-order function `make_luggage_cost` to be used as such:

```
pa_luggage_cost = make_luggage_cost(pa_cost_n)
```

thereafter the function `pa_luggage_cost` can be used as `pa_luggage_cost(bags)` where `bags` is a tuple of the weight of a passenger's luggage.

To better understand higher-order functions, for each function, provide a description of the type of the arguments and the type of the return value. [6 marks]

`make_luggage_cost(cost_n)`
The input `cost_n` is a function and the output of `make_luggage_cost` is a function that takes in a tuple of baggage weights and outputs the total cost.

`cost_n(n, weight)`
The input `n` is the bag number and `weight` is the weight of the current bag. The output is the cost for this bag.

C. Provide an implementation of `make_luggage_cost`. [6 marks]

```
def make_luggage_cost(cost_n):
    def helper(bags):
        result = 0
        for i in range(len(bags)):
            result += cost_n(i+1, bags[i])
     return result
    return helper
```

Question 4: Lambda Luggages [26 marks]

Lambda Luggages is a mysterous bag manufactured by Lambda Inc. It is able to overcome the physical limitations of any object by warping it to another dimenstion and storing the name of the object as a string!

For example, it can store a car by simply storing the string "car". By this logic, an "apple" takes up more space than a "car"!

Lambda Luggages come in different sizes, where "size" is the total number of characters a luggage can contain. So a "Macbook" and its "Charger" cannot both fit into a luggage of size 10 because both objects have 7 characters and combined, exceeds the capacity of the luggage.

Multiple objects can be stored in a Lambda Luggage provided that the total character count does not exceed its size.

A. How would you use a tuple to express the state of a Lambda Luggage? [2 marks]

The first element of the tuple will be the size of the luggage. The remaining elements will be the object name in string.
OR
A pair with the first element being the size of the luggage, and the second element is a tuple of the items.

B. The function `create_luggage` takes the a positive integer n as input and creates a new empty Lambda Luggage of size n. Provide an implementation for `create_luggage`. [2 marks]

```
def create_luggage(n):
    return (n,)
```
or
```
    return (n, ())
```
depending on your chosesn implementation

C. Interestingly, the weight of a Lambda Luggage is equal to the total character count it currently contains. Implement the functions `size` and `weight` which takes as input a lambda luggage and returns the size and weight of the luggage respectively. [4 marks]

```
def size(luggage):
    return luggage[0]
```

```
def weight(luggage):
    weight = 0
    for i in range(1, len(luggage):
        weight += len(luggage[i])
    return weight
```

D. The function `contents` will take as inputs a Lambda Luggage and print out the name of each item it contains.

The function `store` will take as inputs a Lambda Luggage and an item represented by its name, and store the item in the luggage and return an updated Lambda Luggage. If there is insufficient capacity in the luggage to store the item, the unmodified luggage is returned.

Example:

```
>>> luggage1 = create_luggage(20)
>>> luggage2 = store(luggage1, "MacBook")
>>> luggage3 = store(luggage2, "Charger")
>>> contents(luggage3)
"MacBook"
"Charger"

>>> luggage4 = store(luggage3, "AndroidPhone")
>>> contents(luggage4)
"MacBook"
"Charger"
```

Provide an implementation for the functions `contents` and `store`. [8 marks]

```
def contents(luggage):
    for s in luggage[1:]:
        print s









def store(luggage, item):
    if size(luggage) - weight(luggage) < len(item):
        return luggage
    else:
        return luggage + (item,)
```

E. Write a function `remove` that takes as inputs a Lambda Luggage and the name of an item, and returns a Lambda Luggage with the item removed. If the item is not in the luggage, then there is no difference in the luggage returned. [4 marks]

```
def remove(luggage, item):
    for i in range(1, len(luggage):
        if luggage[i] == item:
            return luggage[:i] + luggage[i+1:]
    return luggage
```

F. For some airlines, it is cheaper to carry fewer bags. Now suppose it is possible to store a Lambda Luggage in another Lambda Luggage. However, due to the limits of warping space-time, a Lambda Luggage A can only be stored in Lambda Luggage B if the remaining capacity of luggage B if greater or equal to the current weight of luggage A.

After storing luggage A, the weight of luggage B will be increased by the weight of luggage A. In other words, the weight of luggage B will include the weight of lugguage A.

Modify the function `store` to handle the ability to store a Lambda Luggage in another Lambda Luggage in addition to storing normal objects. Also provide or describe any modification to your implementation of the other functions if required. [6 marks]

```
def store(luggage, item):
    if type(item) == tuple:
        item_weight = weight(item)
    else:
        item_weight = len(item)
    if size(luggage) - weight(luggage) < item_weight:
        return luggage
    else:
        return luggage + (item,)
```

The `weight` function has to be modified to recursively call on a Lambda Luggage when it is looping through the items.

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

— END OF PAPER —