CS1010S — Programming Methodology
School of Computing
National University of Singapore

Mid-Term Test

30 September 2015 **Time allowed:** 1 hour 45 minutes

Matriculation 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 **FOUR (4) questions** and **EIGHTEEN (18) 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]

```
y = 10
x = 2
def f(x):
    x = 13 * x
    y = 12
    return x + y
print(f(y) + x)
```

```
144
```

B. [5 marks]

```
x = 150
for i in range(1, 5):
    if x < 10:
        continue
    x = x // i
print(x)
```

```
6
```

C. [5 marks]

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

```
(1, 2, 6, (4,), 6, (4,))
```

D. [5 marks]

```
x = 10
y = 20
if x > y:
    print("Good")
elif x < y:
    print("Bad")
if x <= y:
    print("Maybe")
```

```
Bad
Maybe
```

E. [5 marks]

```
def boo(x):
    return lambda y: x(x(y))
print(boo(boo)(lambda x:x+1)(5))
```

```
9
```

F. [5 marks]

```
s = "How are you?"
x = 0
while len(s) > x:
    x = x + 1
    if x % 2 == 0:
        continue
    s = s[x:]
print(s)
```

```
ou?
```

Question 2: Digital Sums [24 marks]

The digit sum of an non-negative integer is simply the sum of all its digits. For example, the digit sum of 65536 is $6+5+5+3+6=25$.

A. **[Warm-up]** Write a function `digit_sum` which takes in a non-negative integer n and returns the digit sum of n. [4 marks]

```
def digit_sum(n):
    total = 0
    while n > 0:
        total += n % 10
        n //= 10
    return total

def digit_sum(n):
    if n == 0:
        return 0
    else:
        return n % 10 + digit_sum(n // 10)
```

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.

Hint: The number of digits of an integer n is $\lfloor \log_{10} n \rfloor + 1$ [4 marks]

Time: $O(log_n)$, where n is the input number. Either the number of recursive calls or number of iterations in the loop is the number of digits of n, which is $\lg n$.

Space: $O(1)$ for iterative solution
$O(log_n)$ for recursive solution, where n is the input number.

C. A digital root or also known as a repeated digital sum, of a non-negative integer is the single digit value obtained by repeatedly summing the digits, using the result from the previous iteration to compute the digit sum. This process continues until a single-digit number is reached.

For example, the digital root of 65536 is 7. Because $6+5+5+3+6=25$ and then $2+5=7$.

Write a recursive function `digital_root` which takes in a non-negative integer n and returns the digital root of n.

You may if you wish, reuse the function you defined in part A. [4 marks]

```
def digital_root(n):
    if n < 10:
        return n
    else:
        return digital_root(digital_sum(n))
```

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. Briefly explain your answer.

Remember to account for the function you defined in part A if you had used it. [4 marks]

Time: $O(\log\log n)$ because there are $\log n$ recursive calls, and each call evaluates `digital_sum` which takes $O(\log n)$. But we will accept $O(\log n)$ answers as long as the explanation makes sense.

Space: $O(\log n)$ because there are $\log n$ recursive calls, and each call evaluates `digital_sum` which needs at most $O(\log n)$ space (if defined recursively). So $O(\log n+\log n)=O(\log n)$.

E. Write an iterative function `digital_root` which takes in a non-negative integer n and returns the digital root of n.

You may if you wish, reuse the function you defined in part A. [4 marks]

```
def digital_root(n):
    total = n
    while total > 9:
        total = digital_sum(total)
    return total
```

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. Briefly explain your answer.

Remember to account for the function you defined in part A if you had used it. [4 marks]

Time: $O(\log\log n)$ because the loop will loop some $\log n$ number of times, and each iteration it will evaluate `digital_sum` which takes $\log n$ time. We will however, accept $O(\log n)$ as an answer as long as the explanation is correct.

Space: $O(1)$ if part A was an iterative solution, otherwise $O(\log n)$.

Question 3: Higher-Order Function [24 marks]

Consider the following higher-order function that we call `tail`:

```
def tail(f, n, a, b, c):
    if a > b:
        return c
    else:
        return tail(f, n, n(a), b, f(a, c))
```

A. Suppose the function `sum_integers(n)` computes the sum of integers from 1 to n (inclusive) and `sum_integers(n)` is defined as follows:

```
def sum_integers(n):
    <PRE>
    return tail(<T1>,
                <T2>,
                <T3>,
                <T4>,
                <T5>)
```

Please provide possible implementations for the terms `T1`, `T2`, `T3`, `T4` and `T5`. You may also optionally define other helper functions in `<PRE>` if needed. [6 marks]

*optional <PRE>:	
<T1>:	`lambda x, y: x + y`
<T2>:	`lambda x: x + 1`
<T3>:	`1`
<T4>:	`n`
<T5>:	`0`

B. Suppose the function `sum_even_integers(n)` computes the sum of even integers from 1 to n (inclusive) and `sum_even_integers(n)` is defined as follows:

```
def sum_even_integers(n):
    <PRE>
    return tail(<T6>,
                <T7>,
                <T8>,
                <T9>,
                <T10>)
```

Please provide possible implementations for the terms `T6`, `T7`, `T8`, `T9` and `T10`. You may also optionally define other helper functions in `<PRE>` if needed. [6 marks]

*optional <PRE>:	
<T6>:	`lambda x, y: x + y`
<T7>:	`lambda x: x + 2`
<T8>:	`2`
<T9>:	`n`
<T10>:	`0`

C. We can also express the `sum` function presented in lecture (reproduced in the Appendix) in terms of tail as follows:

```
def sum(term, a, next, b):
    <PRE>
    return tail(<T11>,
                <T12>,
                <T13>,
                <T14>,
                <T15>)
```

Please provide possible implementations for the terms `T11`, `T12`, `T13`, `T14` and `T15`. You may also optionally define other helper functions in `<PRE>` if needed. [6 marks]

*optional <PRE>:	
<T11>:	`lambda x, y: term(x) + y`
<T12>:	`next`
<T13>:	`a`
<T14>:	`b`
<T15>:	`0`

D. **[Warning: HARD!]** We can also express `digit_sum` (described in part A) in terms of tail as follows:

```
def digit_sum(n):
    <PRE>
    return tail(<T16>,
                <T17>,
                <T18>,
                <T19>,
                <T20>)
```

Without using the `digit_sum` function, provide a possible implementation for the terms `T16`, `T17`, `T18`, `T19` and `T20`. You may also optionally define other helper functions in `<PRE>` if needed.

Hint: It might be useful to define a function that returns the i^{th} digit of a number. [6 marks]

*optional `<PRE>`:

```
# returns the digit in the ith place of n
# e.g. if i is 100, return the digit at the 100th place
def digit_at(i, n):
    return (n // i) % 10
```

`<T16>`:

```
lambda x, y: digit_at(x, n) + y
```

`<T17>`:

```
lambda x: x * 10
```

`<T18>`:

```
1
```

`<T19>`:

```
n
```

`<T20>`:

```
0
```

Question 4: Ballot Boxes [22 marks]

Warning: Please read the entire question clearly before you attempt this problem!!

For the upcoming elections, the Elections Committee has decided to use an electronic voting system. You task is to model the ballot boxes which are used to store the ballots, or votes. In addition, each ballot box also identifies the names of the contesting candidates.

A ballot box is created with the function `create_empty_box` which takes as input a tuple of candidates and returns an empty ballot box.

The function `get_candidates` takes as input a ballot box and returns a tuple of candidates contesting in the ballot box, and the function `get_contents` will return a tuple of the ballots in the box in any order.

The function `add_vote` takes as inputs a vote, which is represented represented by a String and a ballot box, and returns a ballot box with the ballot added to its contents. A voter can supply any string in his vote, even leaving it blank.

The function `count_votes` takes as input a ballot box and returns the total number of votes it contains.

Example:

```
>>> box = create_empty_box(('Mouse', 'Tiger'))
>>> count_votes(box)
0

>>> box = add_vote('Tiger', box)
>>> box = add_vote('Mouse', box)
>>> box = add_vote('Ben', box)
>>> box = add_vote('Mouse', box)
>>> box = add_vote('', box)
>>> count_votes(box)
5

>>> get_candidates(box)
('Mouse', 'Tiger')

>>> get_contents(box)
('Tiger', 'Mouse', 'Ben', 'Mouse', '')
```

A. Explain how you will represent the state of a ballot box. [2 marks]

I will use a tuple of two elements, where the first element will be the tuple of candidates and the second element is a tuple of the votes.

B. Provide an implementation for the functions `create_empty_box`, `get_candidates` and `get_contents`. [4 marks]

```
def create_empty_box(candidates):
    return (candidates, ())



def get_candidates(box):
    return box[0]



def get_contents(box):
    return box[1]
```

C. Provide an implementation for the functions `add_vote` and `count_votes`. [4 marks]

```
def add_vote(vote, box):
    return (box[0], box[1] + (vote,))



def count_votes(box):
    return len(box[1])
```

D. When counting the votes, it is also essential to know how many spoilt votes there are. Write a function `count_spoilt_votes` that takes as input a ballot box and return the number of spoilt votes, i.e., the vote is not equal to any of the candidates. [4 marks]

Example:

```
>>> count_spoilt_votes(box)
2

>>> box2 = create_empty_box(('Mouse', 'Tiger'))
>>> box2 = add_vote('Tiger', box2)
>>> box2 = add_vote('Lion', box2)
>>> box2 = add_vote('Mickey', box2)
>>> box2 = add_vote('#whatever', box2)

>>> count_spoilt_votes(box2)
3
```

```
def count_spoilt_votes(box):
    count = 0
    for vote in get_contents(box):
        if vote not in get_candidates(box):
            count += 1
    return count
```

By using the previously defined functions, you can get this question correct even if you gave an incorrect representation of the ballot boxes. But even if not, we may assume some generic representation so you might still be awarded marks if your code is sound.

E. At the counting station, there is a need to combine the votes from two ballot boxes together into a bigger box for easier storage. Write a function `combine_boxes` which takes as inputs two ballot boxes, and returns one ballot box containing the combined votes, provided that both boxes have the same contesting candidates. Otherwise, if the candidates differ, an empty ballot box with no candidates is returned.

You may assume that the candidate lists are always in alphabetical order. [4 marks]

Example:

```
>>> count_votes(box)
5

>>> count_votes(box2)
4

>>> box3 = combine_boxes(box, box2)
>>> count_votes(box3)
9

>>> box4 = create_empty_box(('Cow', 'Tiger'))
>>> box5 = combine_boxes(box3, box4)
>>> count_votes(box5)
0

>>> get_candidates(box5)
()
```

```
def combine_boxes(box1, box2):
    if get_candidates(box1) != get_candidates(box2):
        return create_empty_box(()) # different candidates
    else:
        # loop through the votes in box2 and add to box1
        for vote in get_contents(box2):
            box1 = add_vote(vote, box1)
    return box1
```

This solution uses abstraction so it is possible to get this answer correct even if you gave an incorrect representation for the earlier part. Notice that no knowledge on the actual representation of the ballot boxes is needed for a working solution.

F. On no! Instead of using `combine_boxes`, some idi0t has been putting ballot boxes into boxes using `add_vote`! Nooooo... What happens now is that we have a ballot box that, in addition to votes, also contains other boxes! Now our `count_votes` function might not be correct!

Write a function `unpack_boxes` that takes as input a ballot box and unpacks all the inner boxes and return a ballot box that only contains the votes of all the boxes.

You may assume that all the inner boxes have the same candidates. [4 marks]

Example:

```
>>> bad_box = add_vote(box2, box)
>>> count_votes(bad_box)
6 # This is an incorrect result

>>> fixed_box = unpack_boxes(bad_box)
>>> count_votes(fixed_box)
9

>>> get_contents(fixed_box)
('Tiger', 'Mouse', 'Ben', 'Mouse', '', 'Tiger', 'Lion', 'Mickey', '#whatever')
```

```
def unpack_boxes(box):
    new_box = create_empty_box(get_candidates(box))
    # iterate through every item in the box
    for item in get_contents(box):
        if type(item) == tuple:
            # item is a box! unpack_boxes will remove all inner boxes
            # and returns a box, which we combine with new_box
            new_box = combine_boxes(new_box, unpack_boxes(item))
        else:
            # item is a vote. add to new_box
            new_box = add_vote(item, new_box)
    return new_box
```

Note again that using the abstracted functions without knowing the exact representation of the ballot boxes. No marks will be deducted this time if you did not use abstraction.
2 marks will be awarded if your function can only remove boxes one level deep.

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 —