NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

EXAMINATION FOR
Special Term Part I AY 2013/2014

CS1010FC - PROGRAMMING METHODOLOGY

19 June 2014 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. The examination paper contains **SIX (6) questions** and comprises **EIGHTEEN (18) pages**.
2. Weightage of questions is given in square brackets. The maximum attainable score is 100.
3. This is a **CLOSED** book examination, but you are allowed to bring **TWO** double-sided A4 sheets of notes for this exam.
4. Write all your answers in the space provided in this booklet.
5. **Please write your matriculation number below.**

MATRICULATION NUMBER: _____

(this portion is for the examiner's use only)

Question	Marks	Remark
Q1		
Q2		
Q3		
Q4		
Q5		
Q6		
Total		

Question 1: Warm Up [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. If the interpreter produces an error message, or enters an infinite loop, explain why.

A. [4 marks]

```
a = [1]*3
b = [a]*2
a[1] = 99
print(b)
```

B. [4 marks]

```
a = [1,2,3,4,5,8]
count = 0
for i in range(len(a)):
    if i%2 == 0:
        count += a[i]
    else:
        a.pop()
print(count)
```

C. [4 marks]

```
try:
    y = 21
    for y in range(1,y):
        x = y%7
        if y//x < 3:
            break
except ZeroDivisionError:
    print("Zero!")
except:
    print("Some bad stuff")
else:
    print("Cool!")
```

D. [4 marks]

```
def p():
    return lambda : print("P!")

if not p:
    p()
elif p:
    p()()
else:
    print(not p)
```

E. [4 marks]

```
def new_if(pred, then_clause, else_clause):
    if pred:
        then_clause
    else:
        else_clause
def p(x):
    new_if(x>5, print(x), p(2*x))
p(1)
```

F. [4 marks]

```
a = [1,2,3]
def foo(lst):
    lst.pop()
    lst = list(map(lambda x: 2*x,lst))
foo(a)
print(a)
```

Question 2: Looping Dispute [12 marks]

Ben and Alyssa are engaged in a heated argument over whether `for` loops or `while` loops are better. Ben thinks `for` loops are better. Alyssa thinks otherwise.

A. Alyssa says that anything that `for` loops can do, `while` loops can do too. You agree with Alyssa. To prove your case, please explain how the following two `for` loops can be written in terms of `while` loops:

```
for i in range(a,b,c):
    <do ... i something>

for e in seq:
    <do ... e something>
```

[8 marks]

B. Alyssa says that there are some computations that `while` loops can do that is hard to do with `for` loops. Do you agree? Explain. [4 marks]

Question 3: List Processing Mania [24 marks]

A. The function `mapf` takes a list of functions and an input value and returns a list of the results of the functions applied to the input value. For example,

```
>>> mapf([lambda x: x*2, lambda x: x*x], 4)
[8, 16]

>>> mapf([lambda x: x*2, lambda x: x*x, lambda x: x/3], 3)
[6, 9, 1.0]
```

Suppose the `mapf` is implemented as follows:

```
def mapf(fns, val):
    return list(map(<T1>, <T2>))
```

Please provide possible implementations for the terms `T1` and `T2`. [6 marks]

`T1`:
[3 marks]

`T2`:
[3 marks]

B. Next, suppose we want to operate on a vector of input values (represented by a list), instead of just a single value. Write the function `mapv` that will take in a list of functions and a list of values and returns a list of the lists of the results of the functions applied to the various input values. For example, the input value. For example,

```
>>> mapv([lambda x: x*2, lambda x: x*x], [4])
[[8, 16]]

>>> mapv([lambda x: x*2, lambda x: x*x], [4, 2])
[[8, 16], [4, 4]]

>>> mapv([lambda x: x, lambda x: x*x, lambda x: x**3], [1,2,3])
[[1, 1, 1], [2, 4, 8], [3, 9, 27]]
```

[5 marks]

C. **[Think Differently]** It is likely that you would have implemented `mapv` in Part (B) in terms of `mapf` from Part (A), but suppose for a moment that `mapv` is given to you. Express `mapf` in terms of `mapv`. This should only be one line. [3 marks]

D. **[Generalizing Filter]** Now, let's examine how we can create a generalized version of `filter` called `filterl` that takes in either a predicate or a list of predicates and a list of items and returns a list of items that will satisfy all the predicates. For example,

```
>>> filterl(lambda x: x<5,[1,20,2,3,11])
[1, 2, 3]

>>> filterl([lambda x: x<5, lambda x: x%2 == 1], [1,20,2,3,11])
[1, 3]

>>> filterl([lambda x: x<5, lambda x: x%2 == 1, lambda x: x%2 != 1], [1,20,2,3,11])
[]
```

Please provide a possible implementation of `filterl`. [5 marks]

E. **[Generalizing to Vectors]** Suppose you have a function called `vectorize` such that:

```
mapv = vectorize(mapf)
```

and we can define a more general form of `filterl` called `filterlv` that will allow `filterl` to be applied to lists of lists as follows:

```
>>> filterlv = vectorize(filterl)
>>> filterlv(lambda x: x<5,[[1,20,2,3,11]])
[[1, 2, 3]]

>>> filterlv(lambda x: x<5,[[1,20,2,3,11],[1,5,8]])
[[1, 2, 3], [1]]

>>> filterlv([lambda x: x<5, lambda x: x%2 == 1], [[1,20,2,3,11]])
[[1, 3]]

>>> filterlv([lambda x: x<5, lambda x: x%2 == 1, lambda x: x%2 != 1],
             [[1,20,2,3,11],[1,2]])
[[], []]
```

Please provide a possible implementation of `vectorize`. [5 marks]

Question 4: Money No Enough II [20 marks]

In this question, we will explore a variation of the *Count Change* problem that we discussed in Lecture 4. The code for Lecture 4 is reproduced in the Appendix for your convenient reference.

A. [Warm-Up]. Assuming that there are 5 coin denominations: 1 cent, 5 cents, 10 cents, 20 cents and 50 cents, provide an implementation of `first_denomination(kinds_of_coins)`. [3 marks]

B. [Iterative Count Change]. We discussed in class that that the recursion for *Count Change* will generate a computation tree. In this question, we will explore how *Count Change* can be computated iteratively instead of recursively. The general approach to doing so is to keep track of the computation tree using a list of the nodes.

Recall that in each computation step, what we do is to consider the node (a,d). Depending on a and d, the node either terminates with a 0 or a 1, or it generates 2 new nodes. In this light, what we will do in this question is to use a list to keep track of the nodes. In each computation step, we will take one node out of the list and either do nothing, add 1 to a counter or create 2 nodes that we will put back into the list. When the list is empty, the computation is complete. The skeleton of the code with we call `cc_iter` looks like this:

```
def cc_iter(a,d):
    to_do = [[a,d]]
    count = 0

    while to_do:
        a,d = to_do.pop()

        if a<0 or d==0:
            <T3>
        elif a == 0:
            <T4>
        else:
            <T5>
    return count
```

Please provide possible implementations for the terms `T3`, `T4` and `T5`. [6 marks]

`T3`:
[2 marks]

`T4`:
[2 marks]

`T5`:
[2 marks]

C. [Interpretating Code]. Suppose we want to write a function called `trace` that will allow us to track the number of times that a function is called. Alyssa write the following implementation of `trace`:

```
def trace(f):
    def helper(*args):
        count = [0]
        if args[0] == "count":
            return count[0]
        else:
            count[0] += 1
            return f(*args)
    return helper
```

Read this code, and explain what you think `trace` is supposed to do and how it is supposed to be used in simple English. [3 marks]

D. **[Debugging 101]**. Unfortunately the code for `trace` in Part (C) doesn't quite work as intended. Describe how we might be able to fix it. [3 marks]

E. **[Tracing Functions]**. After fixing `trace` in Part (D), Alyssa tried the following code:

```
>>> def double(x):
    return 2*x

>>> double_trace = trace(double)
>>> print(double_trace(3))
6
>>> print(double_trace("count"))
1
>>> print(double_trace(2))
4
>>> print(double_trace("count"))
2
```

Suppose she tries:

```
>>> cc_trace = trace(cc)
>>> cc_trace(10,5)
4

>>> print(cc_trace("count"))
```

What would be printed for the last `print` statement? Explain. [5 marks]

Question 5: Your Favourite Function in C [17 marks]

The following is the generalized form of the Fibonacci sequence:

$$\begin{aligned} fib(0) &= a \\ fib(1) &= b \\ fib(n) &= c.fib(n-1)+d.fib(n-2), \text{ for} n > 1 \end{aligned}$$

A. Write a C function `fib` that takes in 5 `int` parameters corresponding to a, b, c, d and n returns the nth Fibonacci number $fib(n)$. [5 marks]

B. What is the order of growth in terms of time and space of the function you wrote in Part (A). [2 marks]

C. Ben Bitdiddle also wrote a recursive version of `fib`, but he found that his implementation was too slow. He decided to apply what he learnt in CS1010FC and *memoize* his `fib` implementation. The resulting function which he calls `memo_fib` is as follows:

```
int memo_fib(int a, int b, int c, int d, int n) {

    int memo_table[n+1];

    // Some memoization magic.
}
```

Given your understanding of memoization, briefly explain how Ben can implement the "memoization magic" part of the code in C. There is no need to write any code in this question. Just explain in the words a sketch of how it ought to be done. [5 marks]

D. It turns out that the code that Ben wrote in Part(C) doesn't quite work as intended. It runs, but it doesn't seem any faster than before. Explain why and how you would fix the code. Again, you are not required to write code in this question. [5 marks]

Question 6: 42 and the Meaning of Life [3 marks]

Tell us an interesting story from from your experiences with CS1010FC. Or tell us what you think you learnt this semester. This is your chance to write a short essay instead of code. Or perhaps draw a picture? Do something to convince us that you deserve a good grade for the class. :-)

Appendix

The following are some functions that were introduced in class:

Count Change [Lecture 4]

```
def cc(amount, kinds_of_coins):
    if amount == 0:
        return 1
    elif amount < 0 or kinds_of_coins == 0:
        return 0
    else:
        return cc(amount, kinds_of_coins-1)
             + cc(amount - first_denomination(kinds_of_coins), kinds_of_coins)

def first_denomination(kinds_of_coins):
  ... <left as an exercise>

def count_change(amount)
  return cc(amount,5)
```

— E N D O F P A P E R —

Scratch Paper

– H A P P Y H O L I D A Y S ! –