CS1010FC — Programming Methodology
School of Computing
National University of Singapore

Re-Mid-Term Quiz Solutions

19 April 2014 **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 quiz**. You are allowed to bring one A4 sheet of notes (written on both sides).
3. This paper comprises **FOUR (4) questions** and **NINETEEN (19) pages**. The time allowed for solving this quiz is **1 hour 45 minutes**.
4. The maximum score of this quiz 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 quiz.
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: Sound of Python [25 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.
```
def doh(x):
    return x[0]-1 if x[0] else x[0]+1
print(doh((0,)))
```
[5 marks]

```
1
```

This question tests if the student understands the ternery `if` statement and tuple indexing.

B.
```
def reh(x):
    return x('2')
print(reh(lambda z: str(z)))
```
[5 marks]

```
2
```

This question tests if the student understands types and functions as arguments to other functions.

C.
```
def mee(x):
    s = 'earth'
    output = ''
    while x:
        output += s[x[0]]
        x = x[1:]
    return output
print(mee((3,4,2,0,0)))
```
[5 marks]

```
`three'
```

This question tests if the student understands string manipulation.

D.
```
x, y, z = 1, 3, 5
def fa(w):
    x, y, z = 0, 2, 4
    def f(x):
        x = 2
        return x + z
    def g(y):
        z = 2
        return x + f(y) + z
    return g(w) - y - z
print(fa(x))
```

[5 marks]

```
2
```

This question tests if the student understands scoping.

E.
```
def soh(x):
    counter = 0
    for i in range(x):
        if i%2 == 0:
            continue
        for j in range(i):
            counter = counter + i
    return counter
print(soh(4))
```

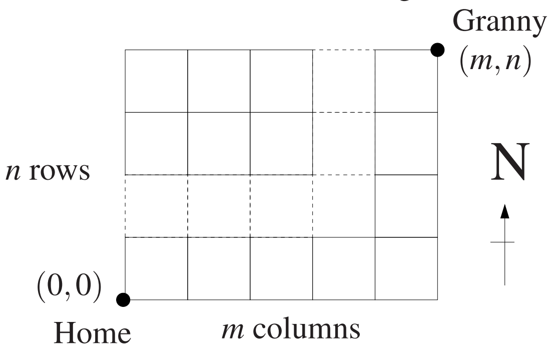
[5 marks]

```
10
```

This question tests if the student understands the `for` loop control structure and the modulus operator.

Question 2: The Riding Hood [24 marks]

In this problem, we model the problem of the Little Red Riding Hood who wants to visit her sick Granny. In order to get from her home to Granny's place, Riding Hood has to walk through a magical grid-like forest. We model this forest with a $m \times n$ grid as shown in the following figure.



The forest is magical because in each step, Riding Hood can either walk north or east (towards Granny's place) at each intersection. She cannot walk backwards.

A. Fill in the code for the function paths_granny that computes the number of distinct paths that Riding Hood can take from her home (at the point $(0,0)$) to reach Granny's place (at the point (m,n)). [6 marks]

```
def paths_granny(m,n):
    if <T1>:
        return 1
    else:
        return <T2>
```

Sample Execution:

```
>>> paths_granny(2,1)
3

>>> paths_granny(2,2)
6

>>> paths_granny(4,3)
35
```

T1:
```
m == 0 or n == 0
```

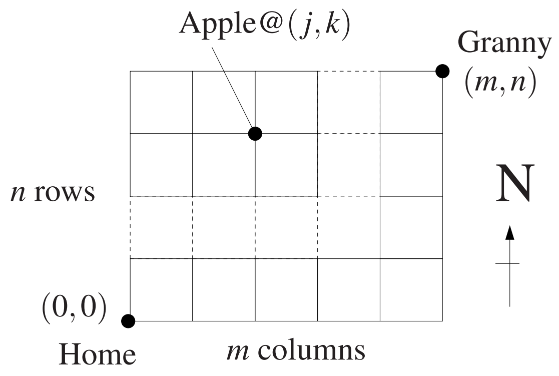
T2:
```
paths_granny(m-1,n) + paths_granny(m,n-1)
```

B. Suppose $m = n$, what is the time complexity (order of growth) for `paths_granny` in terms of n? [2 marks]

$O(2^{2n})$

The recursion tree is binary and has a depth of $2n$.

C. **[Apples for Granny]**. Suppose there is an apple tree in the forest at the location (j,k), where $0 < j < m$ and $0 < k < n$, and Riding Hood wants to pick some apples for Granny on her way to Granny's place.

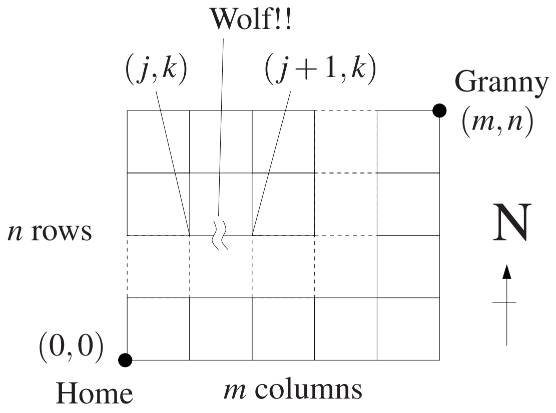


Write the function `paths_to_apple(m,n,j,k)` that computes the number of distinct paths that Riding Hood can take to Granny's place, starting from her home at $(0,0)$ that passes through (j,k). *[Hint: this is very similar to* `paths_granny`*.]* [5 marks]

The key idea in this problem is to recognize that the problem can be decomposed into 2 sub-problems. Even if the student cannot do Part (A), he could in principle still do this problem.

```
def paths_to_apple(m,n,j,k):
    return paths_granny(j,k) * paths_granny(m-j,n-k)
```

D. **[Big Bad Wolf]**. For this question, you can temporarily ignore the apple tree. Suppose someone tells Riding Hood that a Big Bad Wolf is laying in wait along the road between intersections (j,k) and $(j+1,k)$, where $0 < j < m$ and $0 < k < n$.



Write the function `safe_paths(m,n,j,k)` that computes the number of distinct paths that Riding Hood can take to Granny's place, starting from her home at $(0,0)$ such that she will avoid being eaten by the wolf. [5 marks]

This question is a simple variation of Part (C).

```
def safe_paths(m,n,j,k):
    return paths_granny(m,n) - paths_granny(j,k) * paths_granny(m-j-1,n-k)
```

E. [Apple & Wolf]. Now, we will combine Parts (C) and (D). Suppose the apple tree is at (j,k) and the wolf is along the road between intersections (a,b) and $(a+1,b)$, such that $0 < j,a < m$ and $0 < k,b < n$. Write the function `paths_to_apple_no_wolf(m,n,j,k,a,b)` (you can abbreviate with `panw(m,n,j,k,a,b)` if you wish) that computes the number of distinct paths that Riding Hood can take to Granny's place, starting from her home at $(0,0)$ such that she can both pick apples and avoid being eaten by the wolf. Briefly explain your solution. [6 marks]

```
def paths_to_apple_no_wolf(m,n,j,k,a,b):
    if a<=k and b<j:
        return safe_paths(j,k,a,b) * paths_granny(m-j,n-k)
    elif a>=k and b>=j:
        return paths_granny(j,k) * safe_paths(m-j,n-k,a-j,b-k)
    else:
        return paths_to_apple(m,n,j,k)
```

The three cases above are:

1. The wolf blocks at least one route between Riding Hood's home and the tree.
2. The wolf blocks at least one route between the tree and Granny's.
3. Riding Hood will not encounter the wolf in any route that the magical forest allows her to take.

Question 3: Tuple Magic [22 marks]

A. Write the function `level_up` that given a tuple of length n will return a new tuple which has the middle $n-2$ elements raised by one level. If $n < 3$, the returned tuple is the same as the input tuple. [4 marks]

Sample Execution:

```
>>> level_up((1,))
(1,)

>>> level_up((1,2))
(1, 2)

>>> level_up((1,2,3))
(1, (2,), 3)

>>> level_up((1,2,3,4))
(1, (2, 3), 4)

>>> level_up((1,2,3,4,5))
(1, (2, 3, 4), 5)

>>> level_up((1,2,3,4,5,6))
(1, (2, 3, 4, 5), 6)
```

```
def level_up(tup):
    if len(tup) < 3:
        return tup
    else:
        return (tup[0], tup[1:-1], tup[-1])
```

Consider the following higher-order function `tuple_magic`:

```
def tuple_magic(func, pred, base):
    def helper(tup):
        if pred(tup):
            return base(tup)
        else:
            return func(helper, tup)
    return helper
```

B. Suppose:

```
level_up = tuple_magic(<T3>,
                       <T4>,
                       <T5>)
```

Please provide possible implementations for the terms `T3`, `T4` and `T5`, so that this new expression for `level_up` will be equivalent to that in Part(A). [6 marks]

`T3:`
[2 marks]

```
lambda helper, tup:  (tup[0], tup[1:-1], tup[-1])
```

`T4:`
[2 marks]

```
lambda tup:  len(tup) < 3
```

`T5:`
[2 marks]

```
lambda tup:  tup
```

C. Next consider the function `level_down` that undoes the effect of `level_up`, i.e. `level_down(level_up(a))` = `tup`. A possible implementation of `level_down` is:

```
level_down = tuple_magic(<T6>,
                         <T7>,
                         <T8>)
```

Sample Execution:

```
>>> level_down((1,2))
(1, 2)

>>> level_down((1,(2,),3))
(1, 2, 3)

>>> level_down((1,(2,(3,),4),5))
(1, 2, (3,), 4, 5)

>>> level_down((1,(2,3,4),5))
(1, 2, 3, 4, 5)
```

Please provide possible implementations for the terms `T6`, `T7` and `T8`. [6 marks]

`T6:`
[2 marks]

```
lambda helper, tup:  (tup[0],) + tup[1] + (tup[2],)
```

`T7:`
[2 marks]

```
lambda tup:  len(tup) < 3
```

`T8:`
[2 marks]

```
lambda tup:  tup
```

D. `level_up` will only affect one level. Suppose we have another function `level_up_n` that will recursively apply the `level_up` operation to build a pyramid-like structure:

```
level_up_n = tuple_magic(<T9>,
                         <T10>,
                         <T11>)
```

Sample Execution:

```
>>> level_up_n((1,2))
(1, 2)

>>> level_up_n((1,2,3))
(1, (2,), 3)

>>> level_up_n((1,2,3,4))
(1, (2, 3), 4)

>>> level_up_n((1,2,3,4,5))
(1, (2, (3,), 4), 5)

>>> level_up_n((1,2,3,4,5,6))
(1, (2, (3, 4), 5), 6)
```

Please provide possible implementations for the terms `T9`, `T10` and `T11`. [6 marks]

`T9:` [2 marks]

```
lambda helper, tup:  (tup[0], helper(tup[1:-1]), tup[-1])
```

`T10:` [2 marks]

```
lambda tup:  len(tup) < 3
```

`T11:` [2 marks]

```
lambda tup:  tup
```

Question 4: Return to the Pacific Rim [29 marks]

In 2013, human cities have come under attack by the Kaiju, colossal beasts who come to Earth through a portal on the Pacific Ocean floor. To combat them the Pacific Rim nations build the Jaegers, equally colossal robotic war machines. Each Jaeger is piloted by two people whose brains are linked to share the mental load of operating the machine. The Jaegers are initially effective, but many are destroyed as the Kaijus grow more powerful and their attacks more frequent. In 2025, the governments deem the Jaeger program ineffective and discontinue it in favor of building massive coastal walls. The four remaining Jaegers are redeployed to Hong Kong to defend the coast until the wall is completed. Jaeger commander Stacker Pentecost devises a plan to end the threat permanently by destroying the portal with a nuclear bomb.

Pentecost recruits retired Jaeger pilot Raleigh Becket to return and pilot Gipsy Danger, the Jaeger he and his brother Yancy once operated together. During a mission in 2020, Yancy was killed by a Kaiju while still mentally linked to his brother. Raleigh travels to Hong Kong with Pentecost and begins the process of selecting a new co-pilot. Raleigh selects Mako Mori, the director of the Jaeger refurbishment project and Pentecost's adopted daughter. Pentecost initially refuses but relents and allows Mako to participate in a test run with Raleigh. During the test, Mako loses control and nearly discharges Gipsy Danger's cannon in the hangar. Pentecost deems her unready for combat and grounds the pair. When two Kaiju attack Hong Kong, Pentecost is forced to allow them to pilot after two Jaegers are destroyed and one is disabled.

Meanwhile, Newton Geiszler, a scientist studying the Kaijus, creates a device that allows him to establish a mental link with a Kaiju brain. He discovers that the Kaijus are not wild beasts but are cloned living weapons that share a hive mind. The Kaiju fight at the behest of a race of alien colonists who wish to invade Earth. Pentecost demands that Geiszler repeat the experiment on a different Kaiju brain and sends him to Hannibal Chau, a criminal who sells Kaiju body parts on the black market. Geiszler commissions Chau to recover him an intact brain, which they attempt to do from the two Kaiju that just attacked Hong Kong. The men discover a dead Kaiju that was pregnant, and Geiszler uses the brain of the newborn for his experiment after it devours Chau. Geiszler links with the brain and learns that the ocean portal only opens for Kaiju DNA, meaning the bombing mission will fail.

The two remaining Jaegers commence the plan to bomb the portal. Pentecost is forced to pilot the Jaeger Striker Eureka after one of her pilots is injured; Raleigh and Mako pilot Gipsy Danger. Arriving at the portal, they are ambushed by three Kaiju including the largest one they have ever encountered. A violent battle ensues that cripples Gipsy Danger and renders Striker Eureka unable to deliver the bomb. Geiszler warns the two Jaegers that they must take a Kaiju into the portal with them for the bombing to work, and Pentecost decides to detonate his bomb immediately to clear a path to the portal for Raleigh and Mako. Since Gipsy Danger has a nuclear reactor core they can use it to destroy the portal. Pentecost detonates his bomb, destroying two Kaiju and his Jaeger. Gipsy Danger kills the third Kaiju and uses its *body to open the portal. Once inside, Raleigh ejects Mako's escape pod and then sets the Jaeger to self-destruct. The Jaeger erupts on the other side of the portal, destroying the invading aliens and the portal. Raleigh is able jettison his escape pod and travel back before the portal collapses. Mako and Raleigh's escape pods surface safely in the Pacific, and the duo embrace as rescue helicopters arrive.*

– Source: Wikipedia

In this problem, you will model the great battles between the Kaiju and the Jaegers.
Hint: Please read the entire problem before you do this question. While this question is divided into many smaller parts, the parts are related and there are dependencies between them.

You will be expected to decide on the representations for the Kaiju and Jaegars and implement the following functions:

- `make_kaiju(size)` – Creates a new Kaiju object of size `size`.
- `is_kaiju(o)` – Returns `True` if object `o` is a Kaiju. Or `False`, otherwise.
- `make_jaeger(size)` – Creates a new Jaeger object of size `size`.
- `is_jaeger(o)` – Returns `True` if object `o` is a Jaeger. Or `False`, otherwise.
- `size(o)` – Returns the size of the object `o`, where `o` is either a Kaiju or a Jaeger.
- `is_dead(o)` – Returns `True` if object `o` is *dead*, where `o` is either a Kaiju or a Jaeger. New Kaiju and Jaeger objects are not dead by default when first created.
- `get_kills(j)` – Returns the number of kills for Jaeger `j`. New Jaeger objects have 0 kills when first created.
- `fight(a,b)` – Takes in two objects `a` and `b`, which can either be Kaiju or Jaeger, and returns the result after they fight. For details, see Part (F) below.
- `same(o1,o2)` – Returns `True` if `o1` and `o2` are refer to the same original Kaiju or Jaegar object.

A. Describe how you would represent a Kaiju object and write a function `make_kaiju(size)` that takes in the size of the Kaiju and returns a *new* Kaiju object. Write also a function `is_kaiju(o)`, that returns `True` if object `o` is a Kaiju and `False`, otherwise. [4 marks]

```
def make_kaiju(size):
    return ('kaiju', False, (size,))

def is_kaiju(o):
    return type(o) == tuple and len(o) == 3 and o[0] == "kaiju"
```

(Explanation on next page)

A Kaiju is represented by a tuple with three elements:

1. All Kaiju have the string `'kaiju'` as their first element. This allows us to identify whether an object (a tuple in our case) represents a Kaiju.
2. Second element is a boolean that is `True` if the Kaiju is dead and `False` if it is alive.
3. The last element serves the dual purpose of tracking the size of the Kaiju and allowing us to identify whether two Kaiju are identical. (Detailed explanation in part G.)

`is_kaiju` should be able to take in *any* object and return `True` or `False` without resulting in an error. Before we check if the first element is `'kaiju'`, we need to ensure that it is subscriptable (more specifically, we can check that it is a tuple) and that we will not access an index that is out of range (instead of checking `len(o) > 1`, we can be again more specific and check that `len(o) == 3`).

B. Describe how you would represent a Jaeger object and write a function `make_jaeger(size)` that takes in the size of the Jaeger and returns a *new* Jaeger object. Write also a function `is_jaeger(o)`, that returns `True` if object `o` is a Jaeger and `False`, otherwise. [4 marks]

```
def make_jaeger(size):
    return ('jaeger', False, (size,), 0)

def is_jaeger(o):
    return type(o) == tuple and len(o) == 4 and o[0] == "jaeger"
```

This is similar to the Kaiju case. `'jaeger'` identifies a Jaeger instead, and we additionally want to keep track of the number of kills, which is initialized to `0`.

C. Write a function `size` that takes in object `o` and returns the size of the object `o`, where `o` is either a Kaiju or a Jaeger. [3 marks]

```
def size(o):
    return o[2][0]
```

D. Write a function `is_dead` that takes in object `o` and returns `True` if object `o` is *dead*, where `o` is either a Kaiju or a Jaeger. `is_dead` should return `False`, otherwise. [2 marks]

```
def is_dead(o):
    return o[1]
```

E. Write a function `get_kills` that takes in a Jaeger object `j` and returns the number of kills for this Jaeger. When a new Jaeger is created, it has 0 kills. Kills accumulate after a Jaeger gets into fights and wins (see Part (F)). [3 marks]

```
def get_kills(j):
    return j[3]
```

F. [Fight!] The function `fight` takes in two objects `a` and `b`, each of which can either be a Kaiju or Jaeger, and returns a tuple with objects corresponding input objects after they fight. The following are the rules for fighting:

- Kaiju and Jaegers do not fight among themselves, so if `a` and `b` are both Kaiju, or if they are both Jaegers, `fight` will return the tuple `(a,b)`.
- Dead Kaiju and Jaegers cannot fight, so if a fight involves a dead Kaiju or Jaeger, `fight` will return the tuple `(a,b)`.
- If the Kaiju and Jaeger in a fight are of the same size, they will end up in a draw and `fight` will again return the tuple `(a,b)`.
- If the two fighters are of different sizes, the larger one wins and the smaller one dies. `fight` will then return the tuple `(a′,b′)`, where `a′` and `b′` are new objects corresponding to the states of `a` and `b`, respectively. Winning Jaegers will increment their kill count by one.

Please provide a possible implementation for `fight`. [8 marks]

```
def fight(a,b):
    def make_winner(o):
        return o if is_kaiju(o) else o[0:3] + (o[3] + 1,)
    def kill(o):
        return (o[0], True) + o[2:]
    if ((is_jaeger(a) and is_jaeger(b)) or
         (is_kaiju(a) and is_kaiju(b)) or
         is_dead(a) or is_dead(b) or
         size(a) == size(b)):
        return (a,b)
    else:
        if size(a) > size(b):
            return (make_winner(a), kill(b))
        else:
            return (kill(a), make_winner(b))
```

G. The function `fight` will return new objects after a fight if something dies. Given two objects `o1` and `o2`, the function `same` will return `True` if both `o1` and `o2` refer to the same original object. For example, suppose `(a′,b′)` is the result of `fight(a,b)`. Then, `same(a,a′)` and `same(b,b′)` will both return `True`. Please provide a possible implementation for `same`. [5 marks]

```
def same(o1,o2):
    return o1[2] is o2[2]
```

Although two Jaegers (or Kaiju) might be equal, they might not be identical. We cannot simply use `is` to compare two Jaegers because the tuples that represent them have to be recreated every time they kill or die. Instead, each time we update a Jaeger's state, we ensure that the third element points to the same memory address as before. We are able to do this because each tuple `(size,)` is created at a unique memory location upon the creation of a Jaeger or Kaiju, and we retain this object even as we update the the Jaeger or Kaiju after a fight.

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 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 —