National University of Singapore
School of Computing
CS1010X: Programming Methodology
Semester II, 2019/2020

Final Review

The questions are of varying difficulty and are meant to reinforce your fundamentals. Please make sure that you try to work on the questions in IDLE, and not just try to write code on paper (or worse, in your head). This is important. **Simply thinking or writing out the solution and concepts involved on paper is not likely to be sufficient to help prepare you for the practical and final exam!**

We hope that this set of questions will be helpful for your revision on the material for the second half of the course. Good luck!

List processing

1. (a) **[Basic]** Write the **recursive** function `deep_reverse` that takes a list as argument and returns as its value the list with its elements reversed and with all sublists deep-reversed as well.

```
>>> deep_reverse([1, 2,[3, 4], [[5]], [6, [7, 8], 9]])
[[9, [8, 7], 6], [[5]], [4, 3], 2, 1]
```

   (b) **[Basic]** As a test of the concepts introduced in `deep_reverse`, write the **recursive** function `deep_sum` that takes a list as argument and returns as its value the sum of all the number elements in the list and its sublists.

```
>>> deep_sum([1, 2, [3, 4, [[5]], [[6], [7, 8], 9], 10]])
55
```

Implementing Data Structures

2. (a) **[Basic]** In this question, you will learn to build a prefix to infix converter. To achieve this, we first need to build a stack. This data structure will assist in building the converter. Implement a stack, which will accept the functions

```
>>> stk = make_stack()
>>> stk('push',1)
>>> stk('push',2)
>>> stk('push',3)
>>> stk('peek')
3
>>> stk('pop')
3
>>> stk('peek')
2
>>> stk('size')
2
```

(b) **[Hard]** Using the stack you created, write the function `prefix_infix` that takes in a *prefix* expression (represented as a list) and returns the expression in fully-parenthesized *infix* notation. Consider expressions involving only binary operators ($+$, $-$, $*$, $/$). You may find information regarding prefix here: `http://en.wikipedia.org/wiki/Polish_notation`.

```
>>> prefix_infix (['+', '*', 5, 4, '-', 2, 1])
"((5*4)+(2-1))"
```

Signal-Processing View of Computations

3. **[Basic]** Suppose we have `enumerate_interval`, `accumulate`, `map`, `filter` as defined below. Produce the following sequences using these functions (you may add additional functions if necessary). Can you also think of complex sequences that can be created using these functions? Test this on your friend!

   - `[1, 2, 3, 4, 5, 6, 7, 8]`
   - `[5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60]`
   - `[1, 9, 25, 49, 81, 121]`
   - `[1, 1, 9, 2, 25, 3, 49, 4, 81, 5]`
   - `[20, 16, 14, 10, 8, 4, 2]`

```
def enumerate_interval(min, max):
    return list(range(min, max+1))

def map(fn, seq):
    if seq == []:
        return []
    else:
        return [fn(seq[0]),] + map(fn, seq[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:]))
```

4. (a) **[Advanced]** Write the function `power_set` that takes in a list, and returns the power set of that list. The power set is a list that includes all possible subsets of the given list, including the null set. Note that the order of the subsets is not important in this question.

```
>>> power_set([1, 2, 3])
[[1, 2, 3], [1, 2], [1, 3], [2, 3], [1], [2], [3], []]
```

(b) **[Intermediate]** Write the function `power_set_check` that returns `True` if the list is a power set. For simplicity, you may assume that there will be no duplicate sub-sets or nested null sets.

```
>>> power_set_check([[1,2,3],[1,2],[1,3],[2,3],[1],[2],[3],[]])
True

>>> power_set_check([[1, 2, 3]])
False
```

Object-Oriented Programming

5. (a) **[Basic]** Define the class `Number` which makes a "number object". You can assume that the integer value that the number object contains is always divisible to return an integer value. The object has three methods `plus`, `minus`, and `times`:

Sample Execution:

```
>>> two = Number(2)
>>> twelve = Number(12)
>>> thirteen = Number(13)

>>> thirteen.value()
13

>>> five = Number(5)
>>> eight = thirteen.minus(five)
>>> eight.value()
8

>>> twenty_four = two.times(twelve)
>>> twenty_four.value()
24
```

(b) **[Basic]** Let us considered the `divide` method. What happens when we do the following?

```
>>> zero = Number(0)
>>> something = one.divide(zero)
```

(c) **[Intermediate]** Add the `divide` method to `Number`. Handle division by zero in the following way:

```
>>> one = Number(1)
>>> something = one.divide(zero)
>>> something.value()
"Undefined"

>>> another_thing = something.plus(one)
>>> another_thing.value()
"Undefined"
```

```
>>> yet_another_thing = one.minus(something)
>>> yet_another_thing.value()
"Undefined"
```

We hope this has been a good exercise for you. Good luck for your PE and finals!