

---

<b>Midterm (30 pts)</b>
-------------------------

## 1 Recursive Definition (6 pts)

Define a function `sumNumbers` that takes a nested list of numbers and symbols as input, and returns the sum of all numbers in the input list.

```
(sumNumbers '(a)) returns 0
(sumNumbers '(2 56 x (1 1))) returns 60
(sumNumbers '(((a)) -2 (2 (a) (-1 0 1)))) returns 0
```

## 2 Higher-Order Function (6 pts)

Define a function `VAG` (an acronym for `variableArityGeneralize`) that takes a binary function `f` and its identity `id`, and returns its *variable arity* function equivalent.

```
( (VAG + 0) 1 3 5) returns 9
( (VAG * 1) 1 3 5) returns 15
( (VAG append '()) '(1) '(a (b) c) '()) returns (1 a (b) c)
( (VAG * 100) ) returns 100
```

## 3 Conditional Construct (4 pts)

Explain the behavior of the Scheme interpreter on the following code fragment.

```
(define (mystery tse te ee)
  (cond
    (tse te)
    (else ee) ) )
(define (midterm n)
  (mystery (zero? n) 0 (midterm (- n 1)) ) )
(midterm 2)
```

## 4 ADT Specification (3 + 11 pts)

A *bag* is an unordered collection of values of the same type, possibly with duplicates. You are required to specify the ADT **Int\_Bag** that supports the following operations: `create`, `insertOne`, `removeOne`, `isEmpty`, `count`, and `difference`. Informally,

- `create`: Yields the empty bag.
- `insertOne`: Takes an integer and a bag as input, and yields the bag resulting from introducing one occurrence of the integer into the bag.
- `removeOne`: Takes an integer and a bag as input, and yields the bag resulting from deleting one occurrence of the integer from the bag.
- `removeAll`: Takes an integer and a bag as input, and yields the bag resulting from deleting all occurrences of the integer from the bag. (That is, `removeAll(1,[1,1,2,2,3]) = [2,2,3]`, etc)
- `isEmpty`: Checks to see if a bag is empty.
- `count`: Takes an integer and a bag as input, and yields the number of occurrences of the integer in the bag.
- `difference`: Takes two bags as input, and yields the bag containing integers that belong to either the first bag or the second bag, but not both. (That is, `difference([1,1,2,2], [2,3]) = [1,1,2,3]`, `difference([], [3]) = [3]`, etc.)

Recall that a bag is a homogeneous collection of values where duplication is significant, but the order of values is not.

1. Specify the signatures and classify the aforementioned operations on ADT **Int\_Bag**.
2. Give an algebraic specification of the semantics of ADT **Int\_Bag**.