# COSC 365 Programming Assignment 3

# Scheme

For this assignment, you will be implementing several Scheme functions in a single file called `pa3.scm`. Your file will be loaded into the Guile Scheme interpeter and the functions will be evaluated with various inputs by the provided grade script (`grade.sh`). You can run your code by using the grading script or by loading your script directly and manually testing inputs:

```
UNIX$ ./grade.sh
UNIX$ guile -l pa3.scm # load your script and start the REPL
```

## Functions

The functions you must implement are outlined below. Their definitions are included for you in the starter code. **Do not change the name or parameters of the provided functions.**

### `list-len` (10%)

This function should take a single list as a parameter and compute evaluate the number of elements in the list. **You may not use the built-in `length` function to implement `list-len`.**

### `inc-list` (10%)

The function `inc-list` will take a positive integer `n` as its only parameter and evaluate to a list containing every integer from 1 to `n`.

`(inc-list 0)` should produce an empty list.

`(inc-list 3)` should produce the list `(1 2 3)`

### `rev-list` (10%)

This function takes a list as its only parameter and evaluates to the reverse of the list. **You may not use the built-in `reverse` function to implement `rev-list`.**

## `my-map` (20%)

You will be emulating the built-in `map` function (only worrying about unary functions). `my-map` will take a function and a single list as parameters. It will apply the function to each element of the list and produce a new list containing the results of those function applications.

For example,

```
(my-map (lambda (x) (+ x 1)) '(1 2 3))
```

Produces the list `(2 3 4)`.

**You may not use the built-in `map` function to implement `my-map`.**

**You SHOULD use the built-in function `apply` to apply the function to an element of the list.**

## `merge-sort` (50%)

The final function you will complete is an implementation of the Merge Sort algorithm. The `merge-sort` function takes a list as its only parameter and evaluates to the same list in sorted order. The `split` part of the algorithm is already provided for you as a local function in `merge-sort`. It takes a list and evaluates to a pair, where the first element is the first half of the list and the second is the last half of the list.

You should complete this function by creating a new local function called `merge`, which takes two lists as arguments and evaluates to the sorted combination of the lists.

Finally, you should use `split` and `merge` to complete the `merge-sort` function.

## Grading

- **Program Correctness: 50%**

Does your program run correctly and produce the right output for the given inputs? This part of your grade can be computed by using the provided grading script.

- **Paradigm Adherence: 50%**

Does your program solve the problem using the concepts discussed in class and required from this write up? Your solutions should be recursive in nature and compose functions to accomplish tasks. If your code looks imperative or loop-like, you are doing it incorrectly and will get points deducted (this kind of code is hard to write in Scheme anyway).

**NOTE: Your program will be graded for correctness by running it with the provided grading script on the Hydra lab machines.** Ensure that you test on those machines before submitting so that you may be confident in that portion of your grade.

## Tips

- Use the grading script to determine where your output differs from that of the reference directories in order to narrow down missing functionality or output formatting.
- Break the problem into smaller parts.
- Turn the problem into something recursive.
- Watch out for infinite recursion! Do you have a base case?