# Homework 5
# Linear Logic and Linear Type Systems

98-317: Hype for Types

## Introduction

This week, we learned about linear logic and linear type systems. In this homework, you will use the C0-like language we saw in lecture to write some functions on lists in a memory-safe setting.

**Running Your Code:** For this assignment, you'll be using an extension of the C0 language. You can access the binary to run your code on the Andrew servers at `~hgrodin/public/h4t/lc0`.

Alternatively, if you run the following command, you'll be able to run `lc0 <filename>` to run a Linear C0 file: `export PATH="$PATH:$(eval echo `~hgrodin/public/h4t')"`.

**Turning in the Homework:** Submit your `handin.zip` file to Gradescope.

# Operations on Lists

Recall the `lists` that we defined in lecture:

```
struct list {
  int head;
  struct list* tail;
};

struct list* nil() {
  return NULL;
}

struct list* cons(int x, struct list* xs) {
  struct list* node = alloc(struct list);
  node->head = x;
  node->tail = xs;
  return node;
}
```

In the following tasks, you'll extend our implementation of `lists` from lecture by writing some additional functions.

**Task 1.** Write a function called `append` that takes in two `lists` `l1` and `l2` and produces a `list` representing all elements of `l1`, followed by all elements of `l2` (in other words, it should have the same behavior as `@` in SML).

**Task 2.** Write a function called `double` that takes in a `list l` and returns a `list` containing each element of `l`, but repeated twice. For example (using an abstract representation of `lists`),

```
double([1, 2, 3]) = [1, 1, 2, 2, 3, 3]
```

**Task 3.** Using `append`, write a function `repeat` that takes in an `int n` and a `list l` and returns a `list` containing `n` copies of `l`. For instance,

```
repeat(3, [1, 2, 3]) = [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

**Hint:** You might find `drop_list`, `struct pair`, and `copy_list`, from lecture, to be useful.