# Lecture Notes, Tues May 9

## Course Introduction

**Sites to bookmark**: Canvas, Github

**Syllabus and marking scheme**: see Canvas

**TAs and office hours**: see Canvas

**Content overview**: arrays, linked lists, recursion, analysis tools (e.g. O-notation), stacks, queues, deques, trees, heaps, priority queues, hash tables, maps, search trees, sorting, graph algorithms, memory management, B-trees

## Sum in 10 Different Ways

An overview of basic C++, including pointers, passing by constant reference, vectors, using a standard library function (accumulate), and a generic (templated) function.

See sum.cpp.

What version of summation do you like the most? Which one is most readable?

## Introduction to Linked Lists

Linked lists in pictures, compared to arrays/vectors in pictures.

A basic **singly-linked list** of strings, with these basic operations:

* Test if the list is empty.
* Insert a new node at the front.
* Remove the node at the front.
* Count the number of nodes.
* Print the values of the nodes.
* Remove all elements.

Each “node” of a linked-list stores at least two things: it’s data value, and a pointer to the next node in the list. So we can represent a node like this:

struct Node {  
 string data;  
 Node\* next;  
};

The node stores a string, plus a pointer to the next node in the list. The next pointer for the last node of the list will have the value nullptr.

We need to know what node is at the start of the list, so will use a special Node pointer for that:

Node\* head = nullptr;

head always points to the first element of the list. When head is nullptr, the list is empty.

To create a new node for the list, we do this:

head = new Node{"hello", nullptr};

This creates a new Node, and also makes head point to it. This is now a linked list of length 1.

To delete node, we can do this:

delete head;  
head = nullptr;

**Remember**: Every time we call new, there must, eventually, be a call to delete that de-allocates the new-ed memory. Otherwise you have a **memory leak**.