CSCI 104L Lecture 3: Linked Lists

- Advantage: they are easy to grow and shrink.
- Disadvantage: you can't search a sorted list efficiently.

```
struct Item {
  int value;
  Item *next;
  Item *prev;
  Item (int val, Item *n, Item *p) { ... }
};
```

Adding to the front of the list:

```
void DoublyLinkedList::prepend (int n) {
  Item *newElement = new Item (n, head, nullptr);
  head = newElement;
  if (head->next != nullptr) head->next->prev = head;
}
```

Adding to the back of the list (if no tail pointer):

```
void DoublyLinkedList::append(int n) {
  if (head == nullptr) head = new Item (n, nullptr, nullptr);
  else {
    Item *temp = head;
    while (temp->next) temp = temp->next;
    temp->next = new Item (n, nullptr, temp);
  }
}
```

Removing, when given a pointer to the item to be removed:

```
void DoublyLinkedList::remove(Item *toRemove) {
  if (toRemove != head) toRemove->prev->next = toRemove->next;
  else head = toRemove->next;
  if (toRemove->next != nullptr) toRemove->next->prev = toRemove->prev;
  delete toRemove;
}
```

Figure 1: XKCD # 379: Of course, the assert doesn't work.

Some good website exercises (under recursion):

1. Recursion and Linked Lists
   (a) llsum_head
   (b) llsum_tail
2. Recursion and Combos
   (a) bin_combo_str
   (b) prime_products
   (c) basen_combos_str
   (d) all_letter_combos