# 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;
}
```

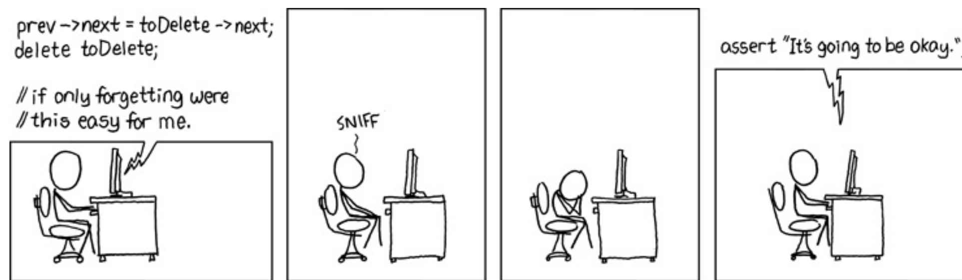


Figure 1: XKCD # 379: Of course, the assert doesn't work.

## Copy Semantics

```
class IntArray {
  public:
    int& operator[] (int index) {
      return data[index];
    }
  private:
    int size; //number of things in the array
    int *data;
    int mem; //allocated memory
};

IntArray& IntArray::operator= (const IntArray &otherArray) {
  if (this == *otherArray) return *this;
  delete [] data;
  this->size = otherArray.size;
  this->data = new int[this->size];
  this->mem = this->size;
  for (int i = 0; i < this->size; i++) this->data[i] = otherArray.data[i];
  return *this;
}
```

**Question 1.** Why did we return IntArray&?.

**Question 2.** What happens in the following code if our copy constructor does a shallow copy and the destructor is implemented as indicated?

```
IntArray::~IntArray () { delete [] data; data = nullptr; }
int main () {
  IntArray a1;
  IntArray a2 (a1);
  return 0;
}
```

In order to avoid this problem, you will need to do a *deep copy*:

```
IntArray::IntArray(const IntArray &a) {
  size = a.size;
  data = new int[size];
  mem = size;
  for (int i = 0; i < size; i++) data[i] = a.data[i];
}
```

The **Rule of Three** states that if you need implement one of the following three functions, then you should implement all three of them:

- Destructor
- Copy Constructor
- Assignment operator