Templates

We’ve implemented IntLinkedList: how could we extend it to allow doubles, chars, or strings?

```cpp
template <class T>
struct Item {
    T value;
    Item<T> *prev, *next;
};
template <class T>
class LinkedList {
    public:
        LinkedList();
        LinkedList(T n);
        virtual ~LinkedList();
        void remove(Item<T> *toRemove);
    private:
        Item<T> *head;
};
template <class T>
LinkedList<T>::prepend(T item) {
    ...
}
```  

```cpp
int main() {
    Item<string> *it = new Item<string>;
    LinkedList<string> *ll = new LinkedList<string>;
}
```

You can have multiple template types.

```cpp
template <class keyType, class valueType>
class map {
    ...
};
```

The main problem that arises is linker errors when you construct templated classes in the normal way. To get around this, you should put your entire implementation in the .h file. This is bad programming practice, so you should never do this for non-templated classes.
**Graphs**

A graph consists of a set of vertices (sometimes called nodes, $V$), and their relationships (or edges, $E$). We will generally refer to $|V| = n$ and $|E| = m$

**Question 1.** Are these problems naturally modeled as directed or undirected graphs?

(a) Computer networks
(b) The Internet.
(c) Social networks.
(d) Road systems.
(e) Predator behavior between species

For a **Graph ADT**, we want to be able to do (at minimum) the following:

1. Add a node.
2. Delete a node.
3. Add an edge.
4. Delete an edge.
5. Test if an edge from $u$ to $v$ exists.
6. Enumerate all outgoing edges from a node.
7. Enumerate all incoming edges from a node.

**Question 2.** What are the runtimes of adding/deleting an edge, testing an edge, or enumerating edges, if we store the edges as...

- An unsorted array or linked list?
- A sorted array?
- An **Adjacency list**: for each node, store a list of adjacent nodes.
- An **Adjacency matrix**: in an $n$ by $n$ matrix of bools, $A[u,v]$ indicates whether there is an edge from node $u$ to node $v$.

In **sparse** graphs ($m = O(n)$), an adjacency list is more economical. For **dense** graphs ($m = \Omega(n^2)$), you might as well go for the adjacency matrix.