Generalized List

Description A list.

State $l$: A sequence of type $T$.

Operations

- list() — Constructor.
  
  Post: $l = \emptyset$ $l$ is the empty sequence.
- ~list() Destructor.
- push_front($T$ $x$) — Mutator. Adds $x$ to the front of the list.
  
  Post: $l' = x$ $l$ has been inserted at the beginning of $l$.
- pop_front() — Mutator. Removes the front element.
  
  Post: $l' = l_{\{1, \ldots, |l|-1\}}$ The front element of $l$ has been removed.
- push_back($T$ $x$) — Mutator. Adds $x$ to the back of the list.
  
  Post: $l' = lx$ $l$ has been appended to the end of $l$.

- $T$ back() — Accessor. Returns the back element of the list.
  
  Pre: $|l| > 0$ $l$ is not empty.
  
  Post: Result = $l_n$ $l$ is the last element of $l$.
- Bool empty() — Accessor. Returns True if the list is empty, false otherwise.
  
  Post: Result = ($|l| = 0$) $l$ is true if $l$ is empty, false otherwise.

Iterators

ADT representing position in a sequence.

- list<int>::iterator $i$ — $i$ is a position in a list of ints.
- $i++$ — increment $i$ to the next position.
- $i--$ — decrement $i$ to the previous position.
- *$i$ — the item at the $i$th position (like a pointer).
- list<int>::const_iterator $i$ — $i$ is a position in a const list of ints.
- $l$.begin() — returns an iterator pointing to the first element in $l$.
- $l$.end() — returns an iterator pointing to one past the end of $l$.

See iterator.cpp
Linked Lists

Implementing lists using arrays may be inefficient in terms of memory — if the maximum list sized is much larger than needed most of the time.

A linked list is a data structure formed by a sequence of Nodes, each of which contains a pointer to one or more other Node.

```
class Node {
  public:
    char data;
    Node* next;
};
```

Aside this is the same as:
```
struct Node {
  char data;
  Node* next;
};
```

The pointers connect the Nodes to form a list. E.g., the list {′A′,′B′,′C′}:
```
head
  data = 'A'
  next
  prev

  data = 'C'
  next

  data = 'B'
  next

head
```

Insert by creating new node and setting the pointers.
Delete by fixing the pointers then deleting the unused node.

Linked List Stack

```
template <class T> class Stack {
  // ...
  private:
    struct Node {
      T data;
      Node* next;
    };

    Node *head; // Pointer to begining of the stack.
};
```

Doubly-linked List

```
class Node {
  public:
    char data;
    Node* next;
    Node* prev;
  Node(char d = 0, Node *p = 0, Node *n = 0)
    : data(d), prev(n), next(n) { }
};
```