Generalized List

Description A list.

State $\mathbf{l}$: A sequence of type $\mathbf{T}$.

Operations

- *list*() — Constructor.
  Post: $\mathbf{l} = \_$, $\mathbf{l}$ is the empty sequence.
- ~*list*() Destructor.
- *push_front*($\mathbf{T}$ x) — Mutator. Adds x to the front of the list.
  Post: $\mathbf{l}' = x\mathbf{l}$, x has been inserted at the begining of $\mathbf{l}$.
- *pop_front*() — Mutator. Removes the front element.
  Pre: $|\mathbf{l}| > 0$, $\mathbf{l}$ is not empty.
  Post: $\mathbf{l}' = \mathbf{l}_{\{1,\ldots|l|-1\}}$, The front element of $\mathbf{l}$ has been removed.
- *push_back*($\mathbf{T}$ x) — Mutator. Adds x to the back of the list.
  Post: $\mathbf{l}' = \mathbf{l}x$, x has been appended to the end of $\mathbf{l}$.

- *pop_back*() — Mutator. Removes the back element.
  Pre: $|\mathbf{l}| > 0$, $\mathbf{l}$ is not empty.
  Post: $\mathbf{l}' = \mathbf{l}_{\{0,\ldots|l|-2\}}$, The last element has been removed from $\mathbf{l}$.
- *insert*($\mathbf{T}$ x, `int` i) — Mutator. Inserts x in the i^{th} position of the list.
  Post: $\mathbf{l}' = \mathbf{l}_{\{0,\ldots i-1\}}x\mathbf{l}_{\{i,\ldots|l|-1\}}$, $\mathbf{l}$ contains x at position i, the elements before i are unchanged, and those after i are shifted right by 1.
- *erase*(`int` i) — Mutator. Removes the i^{th} element.
  Pre: $|\mathbf{l}| > 0$, $\mathbf{l}$ is not empty.
  Post: $\mathbf{l}' = \mathbf{l}_{\{0,\ldots i-1,i+1,\ldots|l|-1\}}$, Elements of $\mathbf{l}$ before i are not changed, the length of $\mathbf{l}$ is one less, and elements after i are shifted left by one.
- $\mathbf{T}$ *front*() — Accessor. Returns the front element of the list.
  Pre: $|\mathbf{l}| > 0$, $\mathbf{l}$ is not empty.
  Post: $\mathit{Result} = \mathbf{l}_0 \wedge \mathbf{l}' = \mathbf{l}$, *Result* is the first element of $\mathbf{l}$.

- $\mathbf{T}$ *back*() — Accessor. Returns the back element of the list.
  Pre: $|\mathbf{l}| > 0$, $\mathbf{l}$ is not empty.
  Post: $\mathit{Result} = \mathbf{l}_n \wedge \mathbf{l}' = \mathbf{l}$, *Result* is the last element of $\mathbf{l}$.
- $\mathbf{Bool}$ *empty*() — Accessor. Returns True if the list is empty, false otherwise.
  Post: $\mathit{Result} = (|\mathbf{l}| = 0)$, *Result* is true if $\mathbf{l}$ 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 $\mathtt{i}^{\text{th}}$ 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.

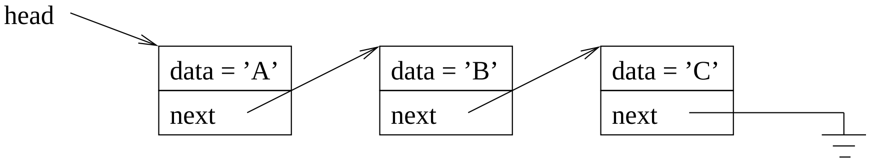
A *linked list* is a data structure formed by a sequence of Nodes, each of which contains a pointer to one or more other Node.

```
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

data = 'B'
next

data = 'C'
next

Insert by creating new node and setting the pointers.

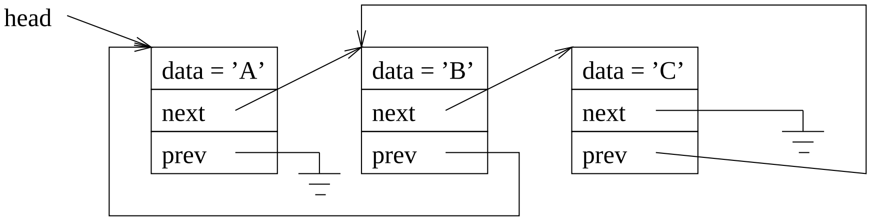
Delete by fixing the pointers then deleting the unused node.

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



head

data = 'A'
next
prev

data = 'B'
next
prev

data = 'C'
next
prev

```
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) { }
};
```