Abstract Data Type (ADT) Linked List

Adopted from M.A. Weiss, Data Structures & Alg. Analysis in C++, Chapter 3

** some modifications to facilitate presentation **

***** Focus on Iterators *****

```
template <typename T>
class List
{
  private:

    struct Node
    {
        T  data;
        Node   *prev;
        Node   *next;

        Node( const T & d = T{ }, Node * p = nullptr,
              Node * n = nullptr )
          : data{ d }, prev{ p }, next{ n } { }
    };

    //---- class iterator ---------------------------

    class iterator
    {
      public:

        iterator( )
          :current(nullptr)
          { }

        iterator(Node* p)
          :current(p)
         { }

        T & operator* ( )
          { return current->data; }
        iterator & operator++ ( )
        {
            this->current = this->current->next;
            return *this;
        }

        iterator operator++ ( int )
        {
            iterator old = *this;
            ++( *this );
            return old;
        }


      protected:

        Node* current;
        friend class List<T>;  // explain in context later
    };

  public:
    // for class List

    List( )
      { init( ); }


    // … LOTS OF OMMITTED List code …

    // Return iterator representing beginning of list.

    iterator begin( )
      { return iterator( head->next ); }   // mutator

    // Return iterator representing endmarker of list.

    iterator end( )
      { return iterator( tail ); }          // mutator
  private:
     // for class List

    int   theSize;
    Node *head;
    Node *tail;

    void init( ){
        theSize = 0;
        head = new Node;
        tail = new Node;
        head->next = tail;
        tail->prev = head;
    }
};
```