C++FA 2.1 List Class Example

This simple example shows the inner workings of a List class. The Element class is a self-referential class that contains an instance of the template type T. The List class wraps a T inside an instance of the Element class, and maintains a pointer to the first and last Element in the List. Each Element points to the next Element in the List. An Iterator is a smart pointer that can point to each Element in the list sequentially. This example fits in section C++FA 2.1 of ***C++ with Financial Applications*** by Ben Van Vliet, available at **www.benvanvliet.net**.

Element.h

```
#pragma once

template< class T >
class Element
{
private:
	T val;

public:
	Element< T > *next;

	Element( T v ) : val( v ), next( NULL ) {}

	T &get_val()
	{
		return val;
	}
	T *get_address()
	{
		return &val;
	}
};
```

List.h

```
#include "Element.h"

template< class T >
class List
{
public:
	Element< T > *first;
	Element< T > *last;

	List() : first( NULL ), last( NULL ) {}

	~List()
	{
```

```
            if ( first != NULL )
            {
                  Element< T > *current = first;
                  Element< T > *temp;
                  while( current != NULL )
                  {
                        temp = current;
                        current = current->next;
                        delete temp;
                  }
            }
      }

      void push_back( T v )
      {
            Element< T > *newptr = new Element< T >( v );
            if ( first == NULL )
            {
                  first = last = newptr;
            }
            else
            {
                  last->next = newptr;
                  last = newptr;
            }
      }
      Element< T > *begin()
      {
            return first;
      }
      Element< T > *end()
      {
            return NULL;
      }

      class iterator
      {
      private:
            Element< T > *current;
      public:
            void operator=( Element< T > *temp )
            {
                  current = temp;
            }
            bool operator!=( Element< T > *temp )
            {
                  return current != temp;
            }
            void operator++( int )
            {
                  current = current->next;
            }
            T *operator->()
            {
                  return current->get_address();
            }
            T &operator*()
```

```
            {
                  return current->get_val();
            }
      };
};
```

Main.cpp for List of Integers

```
#include <iostream>
#include "List.h"
using namespace std;

int main()
{
      List< int > a;
      a.push_back( 4 );
      a.push_back( 5 );
      a.push_back( 7 );
      a.push_back( 3 );

      List< int >::iterator iter;
      for ( iter = a.begin(); iter != a.end(); iter++ )
      {
            cout << *iter << endl;
      }

      return 0;
}
```

Main.cpp for List of MyClasses

```
#include <iostream>
#include "List.h"
using namespace std;

class MyClass
{
private:
      int number;
public:
      MyClass( int n ) : number( n ) {}
      int get_Number() { return number; }
};

int main()
{
      List< MyClass > a;
      a.push_back( MyClass( 4 ) );
      a.push_back( MyClass( 5 ) );
      a.push_back( MyClass( 7 ) );
      a.push_back( MyClass( 3 ) );
```

```
	List< MyClass >::iterator iter;
	for ( iter = a.begin(); iter != a.end(); iter++ )
	{
		cout << iter->get_Number() << endl;
	}

	return 0;
}
```