Abstract Data Types Stack and Queue

Chapter 3 of Weiss's textbook, explains the stack and vector data structures, but no explicit code listings are provided. The following are implementations of stack and queue that are consistent with Weiss, but based on other sources (e.g., Timothy Budd, Data Structures in C++).

Approach taken: Data structures Stack<T> and Queue<T> are implemented as **adaptors of the vector or linked list data structures.**

```
template <typename C>
class Stack
{
 public:

  typedef typename C::value_type value_type;

  Stack(){}

  int size() const {return thestack.size();}

  bool empty() const {return thestack.empty();}

  void push(const value_type& x)
  {
      thestack.push_back(x);
  }
  void pop() {thestack.pop_back();}

  value_type top() {return thestack.back();}

 private:

  C thestack;
};
```

A Stack object can now be declared as an adapter of any type of container that has member functions which support the implementation of the push(), pop(), top(), size() and empty() member functions that are typical for the stack ADT.

List<T> and Vector<T> provide this support. Thus:

```
Stack<Vector<int> > stackOne;

Stack<List<string> > stackTwo;

Stack<MagicContainer<double> > stackThree;
```

In order to make the adaptor approach carry through, these additions need to be made to our Vector.h and List.h files:

```
template <typename T>
class Vector
{
  public:

    typedef T value_type;

  private:
    … <cut>
};


template <typename T>
class List
{
  public:

    typedef T value_type;

  private:
    … <cut>
};
```

With these additions, class Stack has the means to determine that type of value that is to be stored in the stack:

```
Stack<Vector<int> > stackOne;
```

➔ value_type is `int`

```
Stack<List<string> > stackTwo;
```

➔ value_type is `string`

Analogous: The implementation of the ADT Queue

```
template <typename C>
class Queue
{
 public:
  typedef typename C::value_type value_type;

  Queue(){}
  int size() const {return thequeue.size();}
  bool empty() const {return thequeue.empty();}

  void enqueue(const value_type& x)
  {
      thestack.push_back(x);
  }
  void dequeue() {thequeue.pop_front();}

  value_type front() {return thequeue.front();}

 private:

  C thequeue;
};
```

Discuss: What are container structures that can suitably adapt to a Queue<C> data structure?

```
Queue<Vector<int> > queueOne;      --- Yes? No?

Queue<List<string> > queueTwo;     --- Yes? No?

Queue<MagicContainer<double> > queueThree;

                                         --- Yes? No?
```