1 Introduction

This proposal is to change template classes stack, queue, and priority queue in sections from 23.1.9 to 23.1.11. With the current design, users have to provide container class with the data type, for example, `stack<deque<int> >`. The change makes it possible to specify the data type only, that is, `stack<int>`. In addition, with the current vector and stack design, `vector<T>` is a vector of T's and `stack<T>` is a stack implemented by T, this would cause needless confusion. The change makes that `stack<T>` is a stack of T's with the same meaning as of `vector<T>`. Notice that apart from the change in the template parameters, the new design is the same as the current one.

1.1 Stack

Any sequence supporting operations `back`, `push_back` and `pop_back` can be used to instantiate the type parameter `Sequence` in `stack`. In particular, `vector`, `list` and `deque` can be used. `deque` is the default.

```cpp
template <class T, template<class Element> class Sequence = deque,
         class Allocator = allocator>
class stack {
friend bool operator==(const stack<T, Sequence, Allocator>& x,
                     const stack<T, Sequence, Allocator>& y);
friend bool operator<(const stack<T, Sequence, Allocator>& x,
                     const stack<T, Sequence, Allocator>& y);
protected:
    typedef Sequence<T, Allocator> Container;
    Container c;
public:
    typedef Container::value_type value_type;
    typedef Container::size_type size_type;
    bool empty() const { return c.empty(); }
    size_type size() const { return c.size(); }
    value_type& top() { return c.back(); }
    const value_type& top() const { return c.back(); }
    void push(const value_type& x) { c.push_back(x); }
    void pop() { c.pop_back(); }
};
```

For example, `stack<int>` is an integer stack made out of `deque`, and `stack<char, vector>` is a character stack made out of `vector`. 
1.2 Queue

Any sequence supporting operations `front`, `back`, `push_back` and `pop_front` can be used to instantiate the type parameter `Sequence` in `queue`. In particular, `list` and `deque` can be used. `deque` is the default since it takes less space.

```cpp
template <class T, template<class Element> class Sequence = deque,
           class Allocator = allocator>
class queue {
  friend bool operator==(const queue<T, Sequence, Allocator>& x,
                        const queue<T, Sequence, Allocator>& y);
  friend bool operator<(const queue<T, Sequence, Allocator>& x,
                      const queue<T, Sequence, Allocator>& y);
protected:
  typedef Sequence<T, Allocator> Container;
  Container c;
public:
  typedef Container::value_type value_type;
  typedef Container::size_type size_type;
  bool empty() const { return c.empty(); }
  size_type size() const { return c.size(); }
  value_type& front() { return c.front(); }
  const value_type& front() const { return c.front(); }
  value_type& back() { return c.back(); }
  const value_type& back() const { return c.back(); }
  void push(const value_type& x) { c.push_back(x); }
  void pop() { c.pop_front(); }
};
```

For example, `queue<int>` is an integer queue made out of `deque`, and `queue<double, list>` is a double queue made out of `list`.

1.3 Priority queue

Any sequence with random access iterator and supporting operations `front`, `push_back` and `pop_back` can be used to instantiate the type parameter `Sequence` in `priority_queue`. In particular, `vector` and `deque` can be used. `vector` is the default since random access on `vector` takes less time than on `deque`.

```cpp
template <class T, template<class Element> class Sequence = vector,
           class Allocator = allocator>
class priority_queue {
  friend bool operator==(const priority_queue<T, Sequence, Allocator>& x,
                         const priority_queue<T, Sequence, Allocator>& y);
  friend bool operator<(const priority_queue<T, Sequence, Allocator>& x,
                      const priority_queue<T, Sequence, Allocator>& y);
protected:
  typedef Sequence<T, Allocator> Container;
  Container c;
  Compare comp;
public:
```
typedef Container::value_type value_type;
typedef Container::size_type size_type;

priority_queue(const Compare& x = Compare()) : c(), comp(x) {
priority_queue(InputIterator first, InputIterator last,
    const Compare& x = Compare()) : c(first, last), comp(x) {
    make_heap(c.begin(), c.end(), comp);
}
bool empty() const { return c.empty(); }
size_type size() const { return c.size(); }
const value_type& top() const { return c.front(); }
void push(const value_type& x) {
    c.push_back(x);
    push_heap(c.begin(), c.end(), comp);
}
void pop() {
    pop_heap(c.begin(), c.end(), comp);
    c.pop_back();
}

// no equality or less_than operators

For example, priority_queue<int> is an integer priority queue made out of vector, and
priority_queue<short, deque> is a short stack made out of deque.

2 Acknowledgment

The idea for this change comes from Bjarne Stroustrup.