Standard library header <vector>

This header is part of the containers library.

Includes

`<initializer_list>` (C++11)

Classes

vector	dynamic contiguous array (class template)
vector`<bool>`	space-efficient dynamic bitset (class template specialization)
std::hash`<std::vector<bool>>` (C++11)	hash support for `std::vector<bool>` (class template specialization)

Functions

operator== **operator!=** **operator<** **operator<=** **operator>** **operator>=**	lexicographically compares the values in the vector (function template)
std::swap`(std::vector)`	specializes the `std::swap` algorithm (function template)

Synopsis

```
#include <initializer_list>
namespace std {
    template <class T, class Allocator = allocator<T> > class vector;

    template <class T, class Allocator>
        bool operator==(const vector<T,Allocator>& x,const vector<T,Allocator>& y);
    template <class T, class Allocator>
        bool operator< (const vector<T,Allocator>& x,const vector<T,Allocator>& y);
    template <class T, class Allocator>
        bool operator!=(const vector<T,Allocator>& x,const vector<T,Allocator>& y);
    template <class T, class Allocator>
        bool operator> (const vector<T,Allocator>& x,const vector<T,Allocator>& y);
    template <class T, class Allocator>
        bool operator>=(const vector<T,Allocator>& x,const vector<T,Allocator>& y);
    template <class T, class Allocator>
        bool operator<=(const vector<T,Allocator>& x,const vector<T,Allocator>& y);

    template <class T, class Allocator>
        void swap(vector<T,Allocator>& x, vector<T,Allocator>& y);

    template <class Allocator> class vector<bool,Allocator>;

    // hash support
    template <class T> struct hash;
    template <class Allocator> struct hash<vector<bool, Allocator> >;
}
```

Class `std::vector`

```
template <class T, class Allocator = allocator<T> >
class vector {
```

```
public:
    // types:
    typedef value_type&                                          reference;
    typedef const value_type&                                    const_reference;
    typedef /*implementation-defined*/                           iterator;
    typedef /*implementation-defined*/                           const_iterator;
    typedef /*implementation-defined*/                           size_type;
    typedef /*implementation-defined*/                           difference_type;
    typedef T                                                    value_type;
    typedef Allocator                                            allocator_type;
    typedef typename allocator_traits<Allocator>::pointer        pointer;
    typedef typename allocator_traits<Allocator>::const_pointer  const_pointer;
    typedef std::reverse_iterator<iterator>                      reverse_iterator;
    typedef std::reverse_iterator<const_iterator>                const_reverse_iterator;

    // construct/copy/destroy:
    explicit vector(const Allocator& = Allocator());
    explicit vector(size_type n);
    vector(size_type n, const T& value,const Allocator& = Allocator());
    template <class InputIterator>
        vector(InputIterator first, InputIterator last,const Allocator& = Allocator());
    vector(const vector<T,Allocator>& x);
    vector(vector&&);
    vector(const vector&, const Allocator&);
    vector(vector&&, const Allocator&);
    vector(initializer_list<T>, const Allocator& = Allocator());

    ~vector();
    vector<T,Allocator>& operator=(const vector<T,Allocator>& x);
    vector<T,Allocator>& operator=(vector<T,Allocator>&& x);
    vector& operator=(initializer_list<T>);
    template <class InputIterator>
        void assign(InputIterator first, InputIterator last);
    void assign(size_type n, const T& t);
    void assign(initializer_list<T>);
    allocator_type get_allocator() const noexcept;

    // iterators:
    iterator               begin() noexcept;
    const_iterator         begin() const noexcept;
    iterator               end() noexcept;
    const_iterator         end() const noexcept;

    reverse_iterator       rbegin() noexcept;
    const_reverse_iterator rbegin() const noexcept;
    reverse_iterator       rend() noexcept;
    const_reverse_iterator rend() const noexcept;

    const_iterator         cbegin() noexcept;
    const_iterator         cend() noexcept;
    const_reverse_iterator crbegin() const noexcept;
    const_reverse_iterator crend() const noexcept;

    // capacity:
    size_type size() const noexcept;
    size_type max_size() const noexcept;
    void      resize(size_type sz);
    void      resize(size_type sz, const T& c);
    size_type capacity() const noexcept;
    bool      empty() const noexcept;
    void      reserve(size_type n);
    void      shrink_to_fit();

    // element access:
    reference       operator[](size_type n);
    const_reference operator[](size_type n) const;
    reference       at(size_type n);
    const_reference at(size_type n) const;
    reference       front();
    const_reference front() const;
    reference       back();
    const_reference back() const;

    //data access
    T*       data() noexcept;
```

```
    const T* data() const noexcept;

    // modifiers:
    template <class... Args> void emplace_back(Args&&... args);
    void push_back(const T& x);
    void push_back(T&& x);
    void pop_back();

    template <class... Args> iterator emplace(const_iterator position, Args&&... args);
    iterator insert(const_iterator position, const T& x);
    iterator insert(const_iterator position, T&& x);
    iterator insert(const_iterator position, size_type n, const T& x);
    template <class InputIterator>
        iterator insert (const_iterator position, InputIterator first,
                         InputIterator last);
    iterator insert(const_iterator position, initializer_list<T>);

    iterator erase(const_iterator position);
    iterator erase(const_iterator first, const_iterator last);
    void     swap(vector<T,Allocator>&);
    void     clear() noexcept;
};
```

Specialization `std::vector<bool>`

```
template < class Allocator >
class vector<bool, Allocator> {
public:
    // types:
    typedef bool                                                 const_reference;
    typedef /*implementation-defined*/                           iterator;
    typedef /*implementation-defined*/                           const_iterator;
    typedef /*implementation-defined*/                           size_type;
    typedef /*implementation-defined*/                           difference_type;
    typedef bool                                                 value_type;
    typedef Allocator                                            allocator_type;
    typedef /*implementation-defined*/                           pointer;
    typedef /*implementation-defined*/                           const_pointer;
    typedef std::reverse_iterator<iterator>                      reverse_iterator;
    typedef std::reverse_iterator<const_iterator>                const_reverse_iterator;

    // bit reference:
    class reference {
        friend class vector;
        reference() noexcept;
    public:
        ~reference();
        operator bool() const noexcept;
        reference& operator=(const bool x) noexcept;
        reference& operator=(const reference& x) noexcept;
        void flip() noexcept; // flips the bit
    };

    // construct/copy/destroy:
    explicit vector(const Allocator& = Allocator());
    explicit vector(size_type n);
    vector(size_type n, const bool& value,const Allocator& = Allocator());
    template <class InputIterator>
        vector(InputIterator first, InputIterator last,const Allocator& = Allocator());
    vector(const vector<bool,Allocator>& x);
    vector(vector&&);
    vector(const vector&, const Allocator&);
    vector(vector&&, const Allocator&);
    vector(initializer_list<bool>, const Allocator& = Allocator());

    ~vector();
    vector<bool,Allocator>& operator=(const vector<bool,Allocator>& x);
    vector<bool,Allocator>& operator=(vector<bool,Allocator>&& x);
    vector& operator=(initializer_list<bool>);
    template <class InputIterator>
        void assign(InputIterator first, InputIterator last);
    void assign(size_type n, const bool& t);
```

```
    void assign(initializer_list<bool>);
    allocator_type get_allocator() const noexcept;

    // iterators:
    iterator               begin() noexcept;
    const_iterator         begin() const noexcept;
    iterator               end() noexcept;
    const_iterator         end() const noexcept;

    reverse_iterator       rbegin() noexcept;
    const_reverse_iterator rbegin() const noexcept;
    reverse_iterator       rend() noexcept;
    const_reverse_iterator rend() const noexcept;

    const_iterator         cbegin() noexcept;
    const_iterator         cend() noexcept;
    const_reverse_iterator crbegin() const noexcept;
    const_reverse_iterator crend() const noexcept;

    // capacity:
    size_type size() const noexcept;
    size_type max_size() const noexcept;
    void      resize(size_type sz);
    void      resize(size_type sz, const bool& c);
    size_type capacity() const noexcept;
    bool      empty() const noexcept;
    void      reserve(size_type n);
    void      shrink_to_fit();

    // element access:
    reference       operator[](size_type n);
    const_reference operator[](size_type n) const;
    reference       at(size_type n);
    const_reference at(size_type n) const;
    reference       front();
    const_reference front() const;
    reference       back();
    const_reference back() const;

    // modifiers:
    template <class... Args> void emplace_back(Args&&... args);
    void push_back(const bool& x);
    void push_back(bool&& x);
    void pop_back();

    template <class... Args> iterator emplace(const_iterator position, Args&&... args);
    iterator insert(const_iterator position, const bool& x);
    iterator insert(const_iterator position, bool&& x);
    iterator insert(const_iterator position, size_type n, const bool& x);
    template <class InputIterator>
        iterator insert (const_iterator position, InputIterator first,
                         InputIterator last);
    iterator insert(const_iterator position, initializer_list<bool>);

    iterator erase(const_iterator position);
    iterator erase(const_iterator first, const_iterator last);
    void     swap(vector<bool,Allocator>&);
    static void swap(reference x, reference y) noexcept;
    void     flip() noexcept;// flips all bits
    void     clear() noexcept;
};
```