Vector, part 1: Deep Copy Problem

This is part one of the Vector we will implement. The objective with this part is to exemplify the problem with objects owning dynamic resources (memory) and having value semantic. It is also en example of how containers of STL-type works.
This assignment will be extended and complicated in a later assignment.
A container in STL consists of the container class itself and several iterator classes. The iterator classes has normally no visible name but it accessed through type definitions in the container, i.e. `std::vector<float>::iterator`. Often the type of the iterator is not written out at all but auto is used. `"for (auto it=myVector.begin(); ..."`
For simplicity, we in this document describe it as if the iterator is declared:

```
template<class X> class VectorItt;
```

but you should probably rather have T as a parameter (or declare the VectorItt inside the Vector class and also a parameter (int or bool) for the reverse case.

1 Vector class

Minimum requirements: (See std::vector for the exact specifications for the functions)
[const] anger att funktionen finns i två versioner, med const och utan const.
The rightmost column is the order the test program needs and test the operations.

Everything works as std::vector	**Comments**	**Nr**
`using iterator = ... , same for all iterators`	Typdef to enable using of iterators	9
`~Vector();`		1
`Vector() noexcept;`		1
`Vector(const Vector& other);`		4
`Vector(Vector&& other) noexcept;`		7
`Vector(const char* other);`	For testing, other is a “c-string” (assume an existing conversion from char to T)	2
`Vector& operator=(const Vector& other);`		4
`Vector& operator=(Vector&& other) noexcept;`		7
`[const] T& operator[](size_t i) [const];`	Indexing without range check	5
`[const] T& at(size_t i) [const];`	Indexing with range check (throws `std::out_of_range`)	6
`[const]T* data() noexcept [const];`	gives a reference to the internal array	9
`iterator begin() noexcept;`		10
`iterator end() noexcept;`		10
`const_iterator begin() const noexcept;`		10
`const_iterator end() const noexcept;`		10
`const_iterator cbegin() const noexcept;`		10
`const_iterator cend() const noexcept;`		10
`alla ovanför med "iterator" bytt mot "reverse_iterator"`		10
`size_t size() const noexcept;`	aktuellt antal element	1
`size_t capacity() const noexcept;`	Hur mycket kan size() bli utan att omallokera	1
`void reserve(size_t n);`	Öka capacity till >= n	8
`void shrink_to_fit():`	till skillnad från std så kräver vi att utrymmet krymps maximalt (size()==capacity())	8
`void push_back(T c)`	lägger till ett tecken sist	6
`void resize(size_t n)`	Ändrar size() till n, om n>size() så fylls det på med T()	8

Everything works as std::vector	Comments	Nr
`friend bool operator==(const Vector& lhs, const Vector& other)`	global function	2
och alla de andra (!=,<,>,<=,>=). Observera att om du implementerar == och < så kan de andra uttryckas med dem (t.ex. är x<=y det samma som !(y<x)	global functions	3
`#define CHECK assert(Invariant());`	Macro used inside the class.	1
`bool Invariant() const;`	For testing	1
`std::ostream& operator<<(std::ostream& cout , const Vector& other) { for (size_t i = 0; i < other.size(); ++i) cout << other[i]; return cout; }`	global function for testing	2
`template< class T> void swap(Vector<T>& lhs, Vector<T>& rhs );`	global function	8

2 Iterator class

See https://en.cppreference.com/w/cpp/named_req/RandomAccessIterator for details. Observe that a RandomAcessIterator is also a BidirectionalIterator …
Described here are the "iterator", you will also need a const_iterator, reverse_iterator and const_reverse_iterator. VectorItt is used as name of the class, you should probably use another class name.
Iterators are tested at level >= 10

Funktioner/typedefs	Comments
`iterator_category, value_type, difference_type, pointer, reference`	typdefs needed to make the standard algorithms work.
`VectorItt(X* p);`	
`VectorItt();`	
`VectorItt(const VectorItt& other);`	
`VectorItt& operator=(const VectorItt& other);`	
`const_iterator(iterator&)`	
`const_iterator& operator=(iterator&)`	
`X & operator*();`	
`X* operator->();`	
`X& operator[](size_t i);`	Indexering
`VectorItt& operator++();`	`++it;`
`VectorItt& operator--()`	`--it;`
`VectorItt operator++(int);`	`it++`
`VectorItt operator--(int)`	`it--;`
`VectorItt operator+(difference_type i) const;`	
`VectorItt operator-(difference_type i) const;`	
`difference_type operator-(const VectorItt& other) const;`	
`friend bool operator==(const VectorItt& lhs, const VectorItt& rhs);`	global funktion
och alla de andra (!=,<,>,<=,>=)	globala funktioner

OBS!
An iterator in STL is always as light weight as possible and normally just consists of a pointer.