# My Project

Generated by Doxygen 1.10.0

# Chapter 1

# Namespace Index

## 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

# Chapter 4

# Namespace Documentation

## 4.1 std Namespace Reference

**Classes**

- class Vector

# Chapter 5

# Class Documentation

## 5.1 std::Vector< T, Allocator > Class Template Reference

```
#include <Vector.h>
```

### Public Types

- using value_type = T
- using allocator_type = Allocator
- using pointer = typename allocator_traits<Allocator>::pointer
- using const_pointer = typename allocator_traits<Allocator>::const_pointer
- using reference = value_type&
- using const_reference = const value_type&
- using size_type = size_t
- using difference_type = ptrdiff_t
- using iterator = pointer
- using const_iterator = const_pointer
- using reverse_iterator = std::reverse_iterator<iterator>
- using const_reverse_iterator = std::reverse_iterator<const_iterator>

### Public Member Functions

- constexpr Vector () noexcept(noexcept(Allocator()))
- constexpr Vector (const Allocator &alloc) noexcept
- constexpr Vector (size_type n, const Allocator &alloc=Allocator())
- constexpr Vector (size_type n, const T &value, const Allocator &alloc=Allocator())
- ~Vector ()
- constexpr Vector (const Vector &x)
- constexpr Vector (Vector &&x) noexcept
- constexpr Vector & operator= (const Vector &x)
- constexpr Vector & operator= (Vector &&x) noexcept(allocator_traits< Allocator >::propagate_on_↩ container_move_assignment::value||allocator_traits< Allocator >::is_always_equal::value)
- constexpr Vector (initializer_list< T > il, const Allocator &alloc=Allocator())
- constexpr Vector & operator= (initializer_list< T > il)
- constexpr iterator begin () noexcept
- constexpr const_iterator begin () const noexcept

- constexpr iterator end () noexcept
- constexpr const_iterator end () const noexcept
- constexpr reverse_iterator rbegin () noexcept
- constexpr const_reverse_iterator rbegin () const noexcept
- constexpr reverse_iterator rend () noexcept
- constexpr const_reverse_iterator rend () const noexcept
- constexpr const_iterator cbegin () const noexcept
- constexpr const_iterator cend () const noexcept
- constexpr const_reverse_iterator crbegin () const noexcept
- constexpr const_reverse_iterator crend () const noexcept
- constexpr bool empty () const noexcept
- constexpr size_type size () const noexcept
- constexpr size_type max_size () const noexcept
- constexpr size_type capacity () const noexcept
- constexpr void resize (size_type sz)
- constexpr void resize (size_type sz, const T &c)
- constexpr void reserve (size_type n)
- constexpr void shrink_to_fit ()
- constexpr reference operator[ ] (size_type n)
- constexpr const_reference operator[ ] (size_type n) const
- constexpr const_reference at (size_type n) const
- constexpr reference at (size_type n)
- constexpr reference front ()
- constexpr const_reference front () const
- constexpr reference back ()
- constexpr const_reference back () const
- constexpr pointer data () noexcept
- constexpr const_pointer data () const noexcept
- constexpr void assign (size_type n, const T &u)
- template< class InputIterator >
  constexpr void assign (InputIterator first, InputIterator last)
- constexpr void assign (initializer_list< T > il)
- constexpr void push_back (const T &x)
- constexpr void push_back (T &&x)
- template< class... Args>
  constexpr reference emplace_back (Args &&... args)
- constexpr void pop_back ()
- constexpr iterator insert (const_iterator position, const T &x)
- constexpr iterator insert (const_iterator position, T &&x)
- constexpr iterator insert (const_iterator position, size_type n, const T &x)
- template< class InputIterator >
  constexpr iterator insert (const_iterator position, InputIterator first, InputIterator last)
- constexpr iterator insert (const_iterator position, initializer_list< T > il)
- template< class... Args>
  constexpr iterator emplace (const_iterator position, Args &&... args)
- constexpr iterator erase (const_iterator position)
- constexpr iterator erase (const_iterator first, const_iterator last)
- constexpr void swap (Vector &x) noexcept(allocator_traits< Allocator >::propagate_on_container_swap↩
  ::value||allocator_traits< Allocator >::is_always_equal::value)
- constexpr void clear () noexcept
- constexpr Allocator get_allocator () const noexcept

### 5.1.1 Detailed Description

**template<class T, class Allocator = allocator<T>>**
**class std::Vector< T, Allocator >**

Definition at line 14 of file Vector.h.

### 5.1.2 Member Typedef Documentation

#### 5.1.2.1 allocator_type

```
template<class T , class Allocator = allocator<T>>
using std::Vector< T, Allocator >::allocator_type = Allocator
```

Definition at line 23 of file Vector.h.

#### 5.1.2.2 const_iterator

```
template<class T , class Allocator = allocator<T>>
using std::Vector< T, Allocator >::const_iterator = const_pointer
```

Definition at line 31 of file Vector.h.

#### 5.1.2.3 const_pointer

```
template<class T , class Allocator = allocator<T>>
using std::Vector< T, Allocator >::const_pointer = typename allocator_traits<Allocator>↩
::const_pointer
```

Definition at line 25 of file Vector.h.

#### 5.1.2.4 const_reference

```
template<class T , class Allocator = allocator<T>>
using std::Vector< T, Allocator >::const_reference = const value_type&
```

Definition at line 27 of file Vector.h.

#### 5.1.2.5 const_reverse_iterator

```
template<class T , class Allocator = allocator<T>>
using std::Vector< T, Allocator >::const_reverse_iterator = std::reverse_iterator<const_iterator>
```

Definition at line 33 of file Vector.h.

### 5.1.2.6 difference_type

```
template<class T , class Allocator = allocator<T>>
using std::Vector< T, Allocator >::difference_type = ptrdiff_t
```

Definition at line 29 of file Vector.h.

### 5.1.2.7 iterator

```
template<class T , class Allocator = allocator<T>>
using std::Vector< T, Allocator >::iterator = pointer
```

Definition at line 30 of file Vector.h.

### 5.1.2.8 pointer

```
template<class T , class Allocator = allocator<T>>
using std::Vector< T, Allocator >::pointer = typename allocator_traits<Allocator>::pointer
```

Definition at line 24 of file Vector.h.

### 5.1.2.9 reference

```
template<class T , class Allocator = allocator<T>>
using std::Vector< T, Allocator >::reference = value_type&
```

Definition at line 26 of file Vector.h.

### 5.1.2.10 reverse_iterator

```
template<class T , class Allocator = allocator<T>>
using std::Vector< T, Allocator >::reverse_iterator = std::reverse_iterator<iterator>
```

Definition at line 32 of file Vector.h.

### 5.1.2.11 size_type

```
template<class T , class Allocator = allocator<T>>
using std::Vector< T, Allocator >::size_type = size_t
```

Definition at line 28 of file Vector.h.

### 5.1.2.12 value_type

```
template<class T , class Allocator = allocator<T>>
using std::Vector< T, Allocator >::value_type = T
```

Definition at line 22 of file Vector.h.

### 5.1.3 Constructor & Destructor Documentation

#### 5.1.3.1 Vector() [1/7]

```
template<class T , class Allocator = allocator<T>>
constexpr std::Vector< T, Allocator >::Vector ( )  [inline], [constexpr], [noexcept]
```

Definition at line 36 of file Vector.h.

#### 5.1.3.2 Vector() [2/7]

```
template<class T , class Allocator = allocator<T>>
constexpr std::Vector< T, Allocator >::Vector (
             const Allocator & alloc )  [inline], [explicit], [constexpr], [noexcept]
```

Definition at line 37 of file Vector.h.

#### 5.1.3.3 Vector() [3/7]

```
template<class T , class Allocator = allocator<T>>
constexpr std::Vector< T, Allocator >::Vector (
             size_type n,
             const Allocator & alloc = Allocator() )  [inline], [explicit], [constexpr]
```

Definition at line 41 of file Vector.h.

#### 5.1.3.4 Vector() [4/7]

```
template<class T , class Allocator = allocator<T>>
constexpr std::Vector< T, Allocator >::Vector (
             size_type n,
             const T & value,
             const Allocator & alloc = Allocator() )  [inline], [constexpr]
```

Definition at line 45 of file Vector.h.

#### 5.1.3.5 ∼Vector()

```
template<class T , class Allocator = allocator<T>>
std::Vector< T, Allocator >::∼Vector ( )  [inline]
```

Definition at line 50 of file Vector.h.

#### 5.1.3.6 Vector() [5/7]

```
template<class T , class Allocator = allocator<T>>
constexpr std::Vector< T, Allocator >::Vector (
             const Vector< T, Allocator > & x )  [inline], [constexpr]
```

Definition at line 55 of file Vector.h.

#### 5.1.3.7 Vector() [6/7]

```
template<class T , class Allocator = allocator<T>>
constexpr std::Vector< T, Allocator >::Vector (
            Vector< T, Allocator > && x )  [inline], [constexpr], [noexcept]
```

Definition at line 67 of file Vector.h.

#### 5.1.3.8 Vector() [7/7]

```
template<class T , class Allocator = allocator<T>>
constexpr std::Vector< T, Allocator >::Vector (
            initializer_list< T > il,
            const Allocator & alloc = Allocator() )  [inline], [constexpr]
```

Definition at line 132 of file Vector.h.

### 5.1.4 Member Function Documentation

#### 5.1.4.1 assign() [1/3]

```
template<class T , class Allocator = allocator<T>>
constexpr void std::Vector< T, Allocator >::assign (
            initializer_list< T > il )  [inline], [constexpr]
```

Definition at line 370 of file Vector.h.

#### 5.1.4.2 assign() [2/3]

```
template<class T , class Allocator = allocator<T>>
template<class InputIterator >
constexpr void std::Vector< T, Allocator >::assign (
            InputIterator first,
            InputIterator last )  [inline], [constexpr]
```

Definition at line 364 of file Vector.h.

#### 5.1.4.3 assign() [3/3]

```
template<class T , class Allocator = allocator<T>>
constexpr void std::Vector< T, Allocator >::assign (
            size_type n,
            const T & u )  [inline], [constexpr]
```

Definition at line 352 of file Vector.h.

### 5.1.4.4 at() [1/2]

```
template<class T , class Allocator = allocator<T>>
constexpr reference std::Vector< T, Allocator >::at (
             size_type n )  [inline], [constexpr]
```

Definition at line 318 of file Vector.h.

### 5.1.4.5 at() [2/2]

```
template<class T , class Allocator = allocator<T>>
constexpr const_reference std::Vector< T, Allocator >::at (
             size_type n ) const  [inline], [constexpr]
```

Definition at line 310 of file Vector.h.

### 5.1.4.6 back() [1/2]

```
template<class T , class Allocator = allocator<T>>
constexpr reference std::Vector< T, Allocator >::back ( )  [inline], [constexpr]
```

Definition at line 334 of file Vector.h.

### 5.1.4.7 back() [2/2]

```
template<class T , class Allocator = allocator<T>>
constexpr const_reference std::Vector< T, Allocator >::back ( ) const  [inline], [constexpr]
```

Definition at line 338 of file Vector.h.

### 5.1.4.8 begin() [1/2]

```
template<class T , class Allocator = allocator<T>>
constexpr const_iterator std::Vector< T, Allocator >::begin ( ) const  [inline], [constexpr],
[noexcept]
```

Definition at line 155 of file Vector.h.

### 5.1.4.9 begin() [2/2]

```
template<class T , class Allocator = allocator<T>>
constexpr iterator std::Vector< T, Allocator >::begin ( )  [inline], [constexpr], [noexcept]
```

Definition at line 150 of file Vector.h.

#### 5.1.4.10 capacity()

```
template<class T , class Allocator = allocator<T>>
constexpr size_type std::Vector< T, Allocator >::capacity ( ) const  [inline], [constexpr],
[noexcept]
```

Definition at line 223 of file Vector.h.

#### 5.1.4.11 cbegin()

```
template<class T , class Allocator = allocator<T>>
constexpr const_iterator std::Vector< T, Allocator >::cbegin ( ) const  [inline], [constexpr],
[noexcept]
```

Definition at line 190 of file Vector.h.

#### 5.1.4.12 cend()

```
template<class T , class Allocator = allocator<T>>
constexpr const_iterator std::Vector< T, Allocator >::cend ( ) const  [inline], [constexpr],
[noexcept]
```

Definition at line 195 of file Vector.h.

#### 5.1.4.13 clear()

```
template<class T , class Allocator = allocator<T>>
constexpr void std::Vector< T, Allocator >::clear ( )  [inline], [constexpr], [noexcept]
```

Definition at line 575 of file Vector.h.

#### 5.1.4.14 crbegin()

```
template<class T , class Allocator = allocator<T>>
constexpr const_reverse_iterator std::Vector< T, Allocator >::crbegin ( ) const  [inline],
[constexpr], [noexcept]
```

Definition at line 200 of file Vector.h.

#### 5.1.4.15 crend()

```
template<class T , class Allocator = allocator<T>>
constexpr const_reverse_iterator std::Vector< T, Allocator >::crend ( ) const  [inline], [constexpr],
[noexcept]
```

Definition at line 205 of file Vector.h.

#### 5.1.4.16 data() [1/2]

```
template<class T , class Allocator = allocator<T>>
constexpr const_pointer std::Vector< T, Allocator >::data ( ) const  [inline], [constexpr],
[noexcept]
```

Definition at line 346 of file Vector.h.

#### 5.1.4.17 data() [2/2]

```
template<class T , class Allocator = allocator<T>>
constexpr pointer std::Vector< T, Allocator >::data ( )  [inline], [constexpr], [noexcept]
```

Definition at line 342 of file Vector.h.

#### 5.1.4.18 emplace()

```
template<class T , class Allocator = allocator<T>>
template<class... Args>
constexpr iterator std::Vector< T, Allocator >::emplace (
             const_iterator position,
             Args &&... args )  [inline], [constexpr]
```

Definition at line 502 of file Vector.h.

#### 5.1.4.19 emplace_back()

```
template<class T , class Allocator = allocator<T>>
template<class... Args>
constexpr reference std::Vector< T, Allocator >::emplace_back (
             Args &&... args )  [inline], [constexpr]
```

Definition at line 387 of file Vector.h.

#### 5.1.4.20 empty()

```
template<class T , class Allocator = allocator<T>>
constexpr bool std::Vector< T, Allocator >::empty ( ) const  [inline], [constexpr], [noexcept]
```

Definition at line 211 of file Vector.h.

#### 5.1.4.21 end() [1/2]

```
template<class T , class Allocator = allocator<T>>
constexpr const_iterator std::Vector< T, Allocator >::end ( ) const  [inline], [constexpr],
[noexcept]
```

Definition at line 165 of file Vector.h.

### 5.1.4.22 end() [2/2]

```
template<class T , class Allocator = allocator<T>>
constexpr iterator std::Vector< T, Allocator >::end ( )  [inline], [constexpr], [noexcept]
```

Definition at line 160 of file Vector.h.

### 5.1.4.23 erase() [1/2]

```
template<class T , class Allocator = allocator<T>>
constexpr iterator std::Vector< T, Allocator >::erase (
            const_iterator first,
            const_iterator last )  [inline], [constexpr]
```

Definition at line 537 of file Vector.h.

### 5.1.4.24 erase() [2/2]

```
template<class T , class Allocator = allocator<T>>
constexpr iterator std::Vector< T, Allocator >::erase (
            const_iterator position )  [inline], [constexpr]
```

Definition at line 523 of file Vector.h.

### 5.1.4.25 front() [1/2]

```
template<class T , class Allocator = allocator<T>>
constexpr reference std::Vector< T, Allocator >::front ( )  [inline], [constexpr]
```

Definition at line 326 of file Vector.h.

### 5.1.4.26 front() [2/2]

```
template<class T , class Allocator = allocator<T>>
constexpr const_reference std::Vector< T, Allocator >::front ( ) const  [inline], [constexpr]
```

Definition at line 330 of file Vector.h.

### 5.1.4.27 get_allocator()

```
template<class T , class Allocator = allocator<T>>
constexpr Allocator std::Vector< T, Allocator >::get_allocator ( ) const  [inline], [constexpr],
[noexcept]
```

Definition at line 584 of file Vector.h.

#### 5.1.4.28 insert() [1/5]

```
template<class T , class Allocator = allocator<T>>
constexpr iterator std::Vector< T, Allocator >::insert (
            const_iterator position,
            const T & x )  [inline], [constexpr]
```

Definition at line 404 of file Vector.h.

#### 5.1.4.29 insert() [2/5]

```
template<class T , class Allocator = allocator<T>>
constexpr iterator std::Vector< T, Allocator >::insert (
            const_iterator position,
            initializer_list< T > il )  [inline], [constexpr]
```

Definition at line 496 of file Vector.h.

#### 5.1.4.30 insert() [3/5]

```
template<class T , class Allocator = allocator<T>>
template<class InputIterator >
constexpr iterator std::Vector< T, Allocator >::insert (
            const_iterator position,
            InputIterator first,
            InputIterator last )  [inline], [constexpr]
```

Definition at line 471 of file Vector.h.

#### 5.1.4.31 insert() [4/5]

```
template<class T , class Allocator = allocator<T>>
constexpr iterator std::Vector< T, Allocator >::insert (
            const_iterator position,
            size_type n,
            const T & x )  [inline], [constexpr]
```

Definition at line 446 of file Vector.h.

#### 5.1.4.32 insert() [5/5]

```
template<class T , class Allocator = allocator<T>>
constexpr iterator std::Vector< T, Allocator >::insert (
            const_iterator position,
            T && x )  [inline], [constexpr]
```

Definition at line 425 of file Vector.h.

#### 5.1.4.33 max_size()

```
template<class T , class Allocator = allocator<T>>
constexpr size_type std::Vector< T, Allocator >::max_size ( ) const  [inline], [constexpr],
[noexcept]
```

Definition at line 219 of file Vector.h.

#### 5.1.4.34 operator=() [1/3]

```
template<class T , class Allocator = allocator<T>>
constexpr Vector & std::Vector< T, Allocator >::operator= (
             const Vector< T, Allocator > & x )  [inline], [constexpr]
```

Definition at line 73 of file Vector.h.

#### 5.1.4.35 operator=() [2/3]

```
template<class T , class Allocator = allocator<T>>
constexpr Vector & std::Vector< T, Allocator >::operator= (
             initializer_list< T > il )  [inline], [constexpr]
```

Definition at line 140 of file Vector.h.

#### 5.1.4.36 operator=() [3/3]

```
template<class T , class Allocator = allocator<T>>
constexpr Vector & std::Vector< T, Allocator >::operator= (
             Vector< T, Allocator > && x )  [inline], [constexpr], [noexcept]
```

Definition at line 103 of file Vector.h.

#### 5.1.4.37 operator[]() [1/2]

```
template<class T , class Allocator = allocator<T>>
constexpr reference std::Vector< T, Allocator >::operator[] (
             size_type n )  [inline], [constexpr]
```

Definition at line 302 of file Vector.h.

#### 5.1.4.38 operator[]() [2/2]

```
template<class T , class Allocator = allocator<T>>
constexpr const_reference std::Vector< T, Allocator >::operator[] (
             size_type n ) const  [inline], [constexpr]
```

Definition at line 306 of file Vector.h.

#### 5.1.4.39 pop_back()

```
template<class T , class Allocator = allocator<T>>
constexpr void std::Vector< T, Allocator >::pop_back ( )  [inline], [constexpr]
```

Definition at line 398 of file Vector.h.

#### 5.1.4.40 push_back() [1/2]

```
template<class T , class Allocator = allocator<T>>
constexpr void std::Vector< T, Allocator >::push_back (
             const T & x )  [inline], [constexpr]
```

Definition at line 376 of file Vector.h.

#### 5.1.4.41 push_back() [2/2]

```
template<class T , class Allocator = allocator<T>>
constexpr void std::Vector< T, Allocator >::push_back (
             T && x )  [inline], [constexpr]
```

Definition at line 381 of file Vector.h.

#### 5.1.4.42 rbegin() [1/2]

```
template<class T , class Allocator = allocator<T>>
constexpr const_reverse_iterator std::Vector< T, Allocator >::rbegin ( ) const  [inline],
[constexpr], [noexcept]
```

Definition at line 175 of file Vector.h.

#### 5.1.4.43 rbegin() [2/2]

```
template<class T , class Allocator = allocator<T>>
constexpr reverse_iterator std::Vector< T, Allocator >::rbegin ( )  [inline], [constexpr],
[noexcept]
```

Definition at line 170 of file Vector.h.

#### 5.1.4.44 rend() [1/2]

```
template<class T , class Allocator = allocator<T>>
constexpr const_reverse_iterator std::Vector< T, Allocator >::rend ( ) const  [inline], [constexpr],
[noexcept]
```

Definition at line 185 of file Vector.h.

#### 5.1.4.45 rend() [2/2]

```
template<class T , class Allocator = allocator<T>>
constexpr reverse_iterator std::Vector< T, Allocator >::rend ( )  [inline], [constexpr], [noexcept]
```

Definition at line 180 of file Vector.h.

#### 5.1.4.46 reserve()

```
template<class T , class Allocator = allocator<T>>
constexpr void std::Vector< T, Allocator >::reserve (
             size_type n )  [inline], [constexpr]
```

Definition at line 274 of file Vector.h.

#### 5.1.4.47 resize() [1/2]

```
template<class T , class Allocator = allocator<T>>
constexpr void std::Vector< T, Allocator >::resize (
             size_type sz )  [inline], [constexpr]
```

Definition at line 228 of file Vector.h.

#### 5.1.4.48 resize() [2/2]

```
template<class T , class Allocator = allocator<T>>
constexpr void std::Vector< T, Allocator >::resize (
             size_type sz,
             const T & c )  [inline], [constexpr]
```

Definition at line 251 of file Vector.h.

#### 5.1.4.49 shrink_to_fit()

```
template<class T , class Allocator = allocator<T>>
constexpr void std::Vector< T, Allocator >::shrink_to_fit ( )  [inline], [constexpr]
```

Definition at line 293 of file Vector.h.

#### 5.1.4.50 size()

```
template<class T , class Allocator = allocator<T>>
constexpr size_type std::Vector< T, Allocator >::size ( ) const  [inline], [constexpr], [noexcept]
```

Definition at line 215 of file Vector.h.

#### 5.1.4.51 swap()

```
template<class T , class Allocator = allocator<T>>
constexpr void std::Vector< T, Allocator >::swap (
             Vector< T, Allocator > & x )  [inline], [constexpr], [noexcept]
```

Definition at line 555 of file Vector.h.

The documentation for this class was generated from the following file:

- Vector.h

# Chapter 6

# File Documentation

## 6.1 Testai.cpp File Reference

```
#include "catch2/catch.hpp"
#include "Vector.h"
```

#### Macros

- #define CATCH_CONFIG_MAIN

### 6.1.1 Macro Definition Documentation

#### 6.1.1.1 CATCH_CONFIG_MAIN

```
#define CATCH_CONFIG_MAIN
```

Definition at line 1 of file Testai.cpp.

## 6.2 Testai.cpp

Go to the documentation of this file.

```
#define CATCH_CONFIG_MAIN
#include "catch2/catch.hpp"
#include "Vector.h"




```

## 6.3 Vector.h File Reference

```
#include <memory>
#include <iterator>
#include <algorithm>
#include <initializer_list>
#include <stdexcept>
#include <utility>
```

### Classes

- class std::Vector< T, Allocator >

### Namespaces

- namespace std

## 6.4 Vector.h

Go to the documentation of this file.

```
#ifndef VECTOR_H
#define VECTOR_H

#include <memory>
#include <iterator>
#include <algorithm>
#include <initializer_list>
#include <stdexcept>
#include <utility>

namespace std
{
template<class T, class Allocator = allocator<T>>
class Vector
{
    typename allocator_traits<Allocator>::pointer data__;
    size_t size__;
    size_t capacity__;
    Allocator alloc__;
public:
    // types
    using value_type             = T;
    using allocator_type         = Allocator;
    using pointer                = typename allocator_traits<Allocator>::pointer;
    using const_pointer          = typename allocator_traits<Allocator>::const_pointer;
    using reference              = value_type&;
    using const_reference        = const value_type&;
    using size_type              = size_t;
    using difference_type        = ptrdiff_t;
    using iterator               = pointer;
    using const_iterator         = const_pointer;
    using reverse_iterator       = std::reverse_iterator<iterator>;
    using const_reverse_iterator = std::reverse_iterator<const_iterator>;

    // construct/copy/destroy
    constexpr Vector() noexcept(noexcept(Allocator())) : Vector(Allocator()) { }
    constexpr explicit Vector(const Allocator& alloc) noexcept : data__(nullptr), size__(0),
capacity__(0), alloc__(alloc)
    {
        reserve(1);
    };
    constexpr explicit Vector(size_type n, const Allocator& alloc = Allocator()) : data__(nullptr),
size__(0), capacity__(0), alloc__(alloc)
    {
        resize(n);
    };
    constexpr Vector(size_type n, const T& value, const Allocator& alloc = Allocator()):
data__(nullptr), size__(0), capacity__(0), alloc__(alloc)
    {
        resize(n, value);
    };

    ~Vector() // I. destructor
    {
        if(data__)
            allocator_traits<Allocator>::deallocate(alloc__, data__, capacity__);
    };
    constexpr Vector(const Vector& x) // II. copy constructor
        : data__(nullptr), size__(0),
        capacity__(0),
        alloc__(allocator_traits<Allocator>::select_on_container_copy_construction(x.alloc__))
    {
        reserve(x.size__);
        for (size_type i = 0; i < x.size__; ++i) {
            allocator_traits<Allocator>::construct(alloc__, &data__[i], x.data__[i]);
        }
        size__ = x.size__;
    }

    constexpr Vector(Vector&& x) noexcept  // III. move constructor
        : data__(std::exchange(x.data__, nullptr)),
        size__(std::exchange(x.size__, 0)),
        capacity__(std::exchange(x.capacity__, 0)),
        alloc__(std::move(x.alloc__)) {}

    constexpr Vector& operator=(const Vector& x) { // IV. copy assignment
        if (this != &x) {
            if (allocator_traits<Allocator>::propagate_on_container_copy_assignment::value && alloc__
!= x.alloc__) {
                if (data__) {
                    clear();
                    allocator_traits<Allocator>::deallocate(alloc__, data__, capacity__);
                }
                alloc__ = x.alloc__;
                data__ = nullptr;
                capacity__ = 0;
            }
            reserve(x.size__);
            if (x.size__ <= size__) {
                std::copy(x.data__, x.data__ + x.size__, data__);
                for (size_type i = x.size__; i < size__; ++i) {
                    allocator_traits<Allocator>::destroy(alloc__, &data__[i]);
                }
            } else {
                for (size_type i = 0; i < size__; ++i) {
                    allocator_traits<Allocator>::construct(alloc__, &data__[i], x.data__[i]);
                }
                for (size_type i = size__; i < x.size__; ++i) {
                    allocator_traits<Allocator>::construct(alloc__, &data__[i], x.data__[i]);
                }
            }
            size__ = x.size__;
        }
        return *this;
    }

    constexpr Vector& operator=(Vector&& x) // V. move assignment
      noexcept(
        allocator_traits<Allocator>::propagate_on_container_move_assignment::value ||
        allocator_traits<Allocator>::is_always_equal::value
      ){
        if (this != &x) {
            if (allocator_traits<Allocator>::propagate_on_container_move_assignment::value) {
                if (data__) {
                    clear();
                    allocator_traits<Allocator>::deallocate(alloc__, data__, capacity__);
                }
                alloc__ = std::move(x.alloc__);
                data__ = std::exchange(x.data__, nullptr);
                size__ = std::exchange(x.size__, 0);
                capacity__ = std::exchange(x.capacity__, 0);
            } else {
                if (data__) {
                    clear();
                    allocator_traits<Allocator>::deallocate(alloc__, data__, capacity__);
                }

                data__ = std::exchange(x.data__, nullptr);
                size__ = std::exchange(x.size__, 0);
                capacity__ = std::exchange(x.capacity__, 0);
            }
        }
        return *this;
    }

    constexpr Vector(initializer_list<T> il, const Allocator& alloc = Allocator()) : data__(nullptr),
size__(0), capacity__(0), alloc__(alloc)
    {
        reserve(il.size());
        for (auto& elem : il) {
            allocator_traits<Allocator>::construct(alloc__, &data__[size__++], elem);
        }
    }

    constexpr Vector& operator=(initializer_list<T> il) {
        clear();
        reserve(il.size());
        for (auto& elem : il) {
            allocator_traits<Allocator>::construct(alloc__, &data__[size__++], elem);
        }
        return *this;
    }

    // iterators
    constexpr iterator begin() noexcept
    {
        return data__;
    }

    constexpr const_iterator begin() const noexcept
    {
        return data__;
    }

    constexpr iterator end() noexcept
    {
        return data__ + size__;
    }

    constexpr const_iterator end() const noexcept
    {
        return data__ + size__;
    }

    constexpr reverse_iterator rbegin() noexcept
    {
        return reverse_iterator(end());
    }

    constexpr const_reverse_iterator rbegin() const noexcept
    {
        return const_reverse_iterator(end());
    }

    constexpr reverse_iterator rend() noexcept
    {
        return reverse_iterator(begin());
    }

    constexpr const_reverse_iterator rend() const noexcept
    {
        return const_reverse_iterator(begin());
    }

    constexpr const_iterator cbegin() const noexcept
    {
        return data__;
    }

    constexpr const_iterator cend() const noexcept
    {
        return data__ + size__;
    }

    constexpr const_reverse_iterator crbegin() const noexcept
    {
        return const_reverse_iterator(end());
    }

    constexpr const_reverse_iterator crend() const noexcept
    {
        return const_reverse_iterator(begin());
    }

    // capacity
    [[nodiscard]] constexpr bool empty() const noexcept
    {
        return size__==0;
    }
    constexpr size_type size() const noexcept
    {
        return size__;
    }
    constexpr size_type max_size() const noexcept
    {
        return allocator_traits<Allocator>::max_size(alloc__);
    }
    constexpr size_type capacity() const noexcept
    {
        return capacity__;
    }

    constexpr void      resize(size_type sz)
    {
        if (sz > capacity__)
        {
            reserve(sz);
        }
        if (sz > size__)
        {
            for (size_type i = size__; i < sz; ++i)
            {
                allocator_traits<Allocator>::construct(alloc__, &data__[i]);
            }
        }
        else
        {
            for (size_type i = sz; i < size__; ++i)
            {
                allocator_traits<Allocator>::destroy(alloc__, &data__[i]);
            }
        }
        size__ = sz;
    };

    constexpr void      resize(size_type sz, const T& c)
    {
        if (sz > capacity__)
        {
            reserve(sz);
        }
        if (sz > size__)
        {
            for (size_type i = size__; i < sz; ++i)
            {
                allocator_traits<Allocator>::construct(alloc__, &data__[i], c);
            }
        }
        else
        {
            for (size_type i = sz; i < size__; ++i)
            {
                allocator_traits<Allocator>::destroy(alloc__, &data__[i]);
            }
        }
        size__ = sz;
    }

    constexpr void      reserve(size_type n)
    {
        if (n > capacity__)
        {
            pointer new_data = allocator_traits<Allocator>::allocate(alloc__, n);
            for (size_type i = 0; i < size__; ++i)
            {
                allocator_traits<Allocator>::construct(alloc__, &new_data[i], std::move(data__[i]));
                allocator_traits<Allocator>::destroy(alloc__, &data__[i]);
            }
            if (data__)
            {
                allocator_traits<Allocator>::deallocate(alloc__, data__, capacity__);
            }
            data__ = new_data;
            capacity__ = n;
        }
    }

    constexpr void      shrink_to_fit()
    {
        if (size__ < capacity__)
        {
            resize(size__);
        }
    };

    // element access
    constexpr reference       operator[](size_type n)
    {
        return data__[n];
    }
    constexpr const_reference operator[](size_type n) const
    {
        return data__[n];
    }
    constexpr const_reference at(size_type n) const
    {
        if (n >= size__)
        {
            throw std::out_of_range("Vector::at: index out of range");
        }
        return data__[n];
    }
    constexpr reference       at(size_type n)
    {
        if (n >= size__)
        {
            throw std::out_of_range("Vector::at: index out of range");
        }
        return data__[n];
    }
    constexpr reference       front()
    {
        return data__[0];
    }
    constexpr const_reference front() const
    {
        return data__[0];
    }
    constexpr reference       back()
    {
        return data__[size__ - 1];
    }
    constexpr const_reference back() const
    {
        return data__[size__ - 1];
    }
    constexpr pointer         data() noexcept
    {
        return data__;
    }
    constexpr const_pointer   data() const noexcept
    {
        return data__;
    }

    // modifiers
    constexpr void assign(size_type n, const T& u)
    {
        clear();
        reserve(n);
        for (size_type i = 0; i < n; ++i)
        {
            allocator_traits<Allocator>::construct(alloc__, &data__[i], u);
        }
        size__ = n;
    }

    template<class InputIterator>
    constexpr void assign(InputIterator first, InputIterator last)
    {
        clear();
        insert(begin(), first, last);
    }

    constexpr void assign(initializer_list<T> il)
    {
        clear();
        insert(begin(), il);
    }

    constexpr void push_back(const T& x)
    {
        emplace_back(x);
    }

    constexpr void push_back(T&& x)
    {
        emplace_back(std::move(x));
    }

    template<class... Args>
    constexpr reference emplace_back(Args&&... args)
    {
        if (size__ == capacity__)
        {
            reserve(size__ > 0 ? 2 * size__ : 1);
        }
        allocator_traits<Allocator>::construct(alloc__, &data__[size__], std::forward<Args>(args)...);
        ++size__;
        return data__[size__ - 1];
    }

    constexpr void pop_back()
    {
        allocator_traits<Allocator>::destroy(alloc__, &data__[size__ - 1]);
        --size__;
    }

    constexpr iterator insert(const_iterator position, const T& x)
    {
        difference_type offset = position - cbegin();
        if (size__ == capacity__)
        {
            reserve(size__ > 0 ? 2 * size__ : 1);
        }
        iterator pos = begin() + offset;
        if (pos != end())
        {
            for (iterator it = end(); it != pos; --it)
            {
                allocator_traits<Allocator>::construct(alloc__, it, std::move(*(it - 1)));
                allocator_traits<Allocator>::destroy(alloc__, it - 1);
            }
        }
        allocator_traits<Allocator>::construct(alloc__, pos, x);
        ++size__;
        return pos;
    }

    constexpr iterator insert(const_iterator position, T&& x)
    {
        difference_type offset = position - cbegin();
        if (size__ == capacity__)
        {
            reserve(size__ > 0 ? 2 * size__ : 1);
        }
        iterator pos = begin() + offset;
        if (pos != end())
        {
            for (iterator it = end(); it != pos; --it)
            {
                allocator_traits<Allocator>::construct(alloc__, it, std::move(*(it - 1)));
                allocator_traits<Allocator>::destroy(alloc__, it - 1);
            }
        }
        allocator_traits<Allocator>::construct(alloc__, pos, std::move(x));
        ++size__;
        return pos;
    }

    constexpr iterator insert(const_iterator position, size_type n, const T& x)
    {
        difference_type offset = position - cbegin();
        if (size__ + n > capacity__)
        {
            reserve(size__ + n);
        }
        iterator pos = begin() + offset;
        if (pos != end())
        {
            for (iterator it = end() + n - 1; it != pos + n - 1; --it)
            {
                allocator_traits<Allocator>::construct(alloc__, it, std::move(*(it - n)));
                allocator_traits<Allocator>::destroy(alloc__, it - n);
            }
        }
        for (iterator it = pos; it != pos + n; ++it)
        {
            allocator_traits<Allocator>::construct(alloc__, it, x);
        }
        size__ += n;
        return pos;
    }

    template<class InputIterator>
    constexpr iterator insert(const_iterator position, InputIterator first, InputIterator last)
    {
        difference_type offset = position - cbegin();
        size_type n = std::distance(first, last);
        if (size__ + n > capacity__)
        {
            reserve(size__ + n);
        }
        iterator pos = begin() + offset;
        if (pos != end())
        {
            for (iterator it = end() + n - 1; it != pos + n - 1; --it)
            {
                allocator_traits<Allocator>::construct(alloc__, it, std::move(*(it - n)));
                allocator_traits<Allocator>::destroy(alloc__, it - n);
            }
        }
        for (iterator it = pos; first != last; ++it, ++first)
        {
            allocator_traits<Allocator>::construct(alloc__, it, *first);
        }
        size__ += n;
        return pos;
    }

    constexpr iterator insert(const_iterator position, initializer_list<T> il)
    {
        return insert(position, il.begin(), il.end());
    }

    template<class... Args>
    constexpr iterator emplace(const_iterator position, Args&&... args)
    {
        difference_type offset = position - cbegin();
        if (size__ == capacity__)
        {
            reserve(size__ > 0 ? 2 * size__ : 1);
        }
        iterator pos = begin() + offset;
        if (pos != end())
        {
            for (iterator it = end(); it != pos; --it)
            {
                allocator_traits<Allocator>::construct(alloc__, it, std::move(*(it - 1)));
                allocator_traits<Allocator>::destroy(alloc__, it - 1);
            }
        }
        allocator_traits<Allocator>::construct(alloc__, pos, std::forward<Args>(args)...);
        ++size__;
        return pos;
    }

    constexpr iterator erase(const_iterator position)
    {
        difference_type offset = position - cbegin();
        iterator pos = begin() + offset;
        allocator_traits<Allocator>::destroy(alloc__, pos);
        for (iterator it = pos; it != end() - 1; ++it)
        {
            allocator_traits<Allocator>::construct(alloc__, it, std::move(*(it + 1)));
            allocator_traits<Allocator>::destroy(alloc__, it + 1);
        }
        --size__;
        return pos;
    }

    constexpr iterator erase(const_iterator first, const_iterator last)
    {
        difference_type offset = first - cbegin();
        iterator pos = begin() + offset;
        difference_type n = last - first;
        for (iterator it = pos; it != pos + n; ++it)
        {
            allocator_traits<Allocator>::destroy(alloc__, it);
        }
        for (iterator it = pos; it != end() - n; ++it)
        {
            allocator_traits<Allocator>::construct(alloc__, it, std::move(*(it + n)));
            allocator_traits<Allocator>::destroy(alloc__, it + n);
        }
        size__ -= n;
        return pos;
    }

    constexpr void swap(Vector& x) noexcept(
        allocator_traits<Allocator>::propagate_on_container_swap::value ||
        allocator_traits<Allocator>::is_always_equal::value)
    {
        if (this != &x)
        {
            if (allocator_traits<Allocator>::propagate_on_container_swap::value || alloc__ ==
 x.alloc__)
            {
                std::swap(data__, x.data__);
                std::swap(size__, x.size__);
                std::swap(capacity__, x.capacity__);
                std::swap(alloc__, x.alloc__);
            }
            else
            {
                throw std::runtime_error("Allocator mismatch in Vector::swap");
            }
        }
    }

    constexpr void clear() noexcept
    {
        for (size_type i = 0; i < size__; ++i)
        {
            allocator_traits<Allocator>::destroy(alloc__, &data__[i]);
        }
        size__ = 0;
    }

    constexpr Allocator get_allocator() const noexcept
    {
        return alloc__;
    }
};

}
#endif



```

# Index