

MyVector

Generated by Doxygen 1.13.2

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 myVector< T > Class Template Reference	5
3.1.1 Constructor & Destructor Documentation	7
3.1.1.1 myVector() [1/6]	7
3.1.1.2 myVector() [2/6]	7
3.1.1.3 myVector() [3/6]	7
3.1.1.4 myVector() [4/6]	8
3.1.1.5 myVector() [5/6]	8
3.1.1.6 myVector() [6/6]	8
3.1.1.7 ~myVector()	9
3.1.2 Member Function Documentation	9
3.1.2.1 assign() [1/2]	9
3.1.2.2 assign() [2/2]	9
3.1.2.3 at()	9
3.1.2.4 back()	10
3.1.2.5 begin()	10
3.1.2.6 capacity()	10
3.1.2.7 cbegin()	11
3.1.2.8 cend()	11
3.1.2.9 clear()	11
3.1.2.10 data()	11
3.1.2.11 empty()	11
3.1.2.12 end()	12
3.1.2.13 front()	12
3.1.2.14 insert() [1/2]	12
3.1.2.15 insert() [2/2]	12
3.1.2.16 operator!=(())	13
3.1.2.17 operator=() [1/3]	13
3.1.2.18 operator=() [2/3]	13
3.1.2.19 operator=() [3/3]	14
3.1.2.20 operator==(())	14
3.1.2.21 operator[]()	14
3.1.2.22 output()	15
3.1.2.23 pop_back()	15
3.1.2.24 push_back()	15
3.1.2.25 reserve()	15
3.1.2.26 shrink_to_fit()	16

3.1.2.27 size()	16
4 File Documentation	17
4.1 myvector.h File Reference	17
4.2 myvector.h	17
Index	23

Chapter 1

Class Index

1.1 Class List

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

myVector< T >	5
---	---

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

myvector.h	17
----------------------------	-------	----

Chapter 3

Class Documentation

3.1 myVector< T > Class Template Reference

```
#include <myvector.h>
```

Public Member Functions

- [myVector](#) ()
Creates an empty vector with default capacity of 10.
- [myVector](#) (const int &num, const T &val)
Creates a vector with given size and fills it with given value
.
- [myVector](#) (const T *begin, const T *end)
Creates a vector with values from given range
.
- [myVector](#) (const [myVector](#) &other)
Creates a vector with values from given vector. Values in original vector are not moved
.
- [myVector](#) ([myVector](#) &&other)
Creates a vector with values from given vector and "moves" the values from it
.
- [myVector](#) (std::initializer_list< T > list)
Creates a vector with values from given initializer list
.
- [myVector](#) operator= (std::initializer_list< T > list)
Assigns values from given initializer list to the vector
.
- [myVector](#) & operator= (const [myVector](#) &other)
Assigns values from given vector to the vector. Values in original vector are not moved
.
- [myVector](#) & operator= ([myVector](#) &&other)
Assigns values from given vector to the vector and "moves" the values from it
.
- [~myVector](#) ()
Frees the memory and resets size_, capacity_, begin_ and end_.
- int [size](#) () const

- Returns the size of the vector*
- int `capacity` () const
Returns the capacity of the vector
- T * `begin` ()
Returns the pointer to the beginning of the vector
- T * `cbegin` () const
Returns the pointer to the beginning of the vector, but as const
- T * `end` ()
Returns the pointer to element past the end of the vector
- T * `cend` () const
Returns the pointer to element past the end of the vector, but as const
- T & `front` () const
Returns the first element of the vector
- T & `back` () const
Returns the last element of the vector
- T & `operator[]` (const int &index) const noexcept
Returns the element at given index
- T & `at` (const int &index) const
Returns the element at given index (with bounds checking)
- void `push_back` (const T &value)
Adds an element to the end of the vector. If the vector is full, it doubles the capacity.
- void `pop_back` ()
Removes the last element of the vector.
- bool `empty` () const
Checks if the vector is empty
- void `reserve` (const int &num)
Reserves memory for the vector. If the given number is smaller than the current capacity, it does nothing
- void `shrink_to_fit` ()
Shrinks the vector to fit the current size. If the capacity is equal to the size, it does nothing.
- void `clear` ()
Clears the vector. Frees the old memory, but the capacity remains the same.
- std::wstring `output` () const
Outputs the vector to a string (for testing purposes)
- bool `operator==` (const `myVector` &second) const
Compares two vectors. If the sizes are different, it returns false. If the sizes are the same, it compares the elements.
- bool `operator!=` (const `myVector` &second) const
Compares two vectors. If the sizes are different, it returns true. If the sizes are the same, it compares the elements.
- void `assign` (const int &num, const T &val)

Replaces the elements with num copies of val value.

- void `assign` (const T *start, const T *end)

Replaces the elements with elements from range from start to end

- T * `data` () const

Returns pointer to the underlying array.

- void `insert` (const T *pos, const T &val)

Inserts an copy of element val into place before position pos.

- void `insert` (const T *pos, const int num, const T &val)

Inserts a num copies of element val into place before position pos.

3.1.1 Constructor & Destructor Documentation

3.1.1.1 myVector() [1/6]

```
template<typename T>
myVector< T >::myVector () [inline]
```

Creates an empty vector with default capacity of 10.

3.1.1.2 myVector() [2/6]

```
template<typename T>
myVector< T >::myVector (
    const int & num,
    const T & val) [inline]
```

Creates a vector with given size and fills it with given value

Parameters

<i>num</i>	- size of the vector
<i>val</i>	- value to fill the vector with

3.1.1.3 myVector() [3/6]

```
template<typename T>
myVector< T >::myVector (
    const T * begin,
    const T * end) [inline]
```

Creates a vector with values from given range

Parameters

<i>begin</i>	- pointer to the beginning of the range
<i>end</i>	- pointer to the end of the range

3.1.1.4 myVector() [4/6]

```
template<typename T>
myVector< T >::myVector (
    const myVector< T > & other) [inline]
```

Creates a vector with values from given vector. Values in original vector are not moved
.

Parameters

<i>other</i>	- vector to copy from
--------------	-----------------------

3.1.1.5 myVector() [5/6]

```
template<typename T>
myVector< T >::myVector (
    myVector< T > && other) [inline]
```

Creates a vector with values from given vector and "moves" the values from it
.

Parameters

<i>other</i>	- vector to move from
--------------	-----------------------

3.1.1.6 myVector() [6/6]

```
template<typename T>
myVector< T >::myVector (
    std::initializer_list< T > list) [inline]
```

Creates a vector with values from given initializer list
.

Parameters

<i>list</i>	- initializer list to copy from
-------------	---------------------------------

3.1.1.7 ~myVector()

```
template<typename T>
myVector< T >::~~myVector () [inline]
```

Frees the memory and resets size_, capacity_, begin_ and end_.

3.1.2 Member Function Documentation

3.1.2.1 assign() [1/2]

```
template<typename T>
void myVector< T >::assign (
    const int & num,
    const T & val) [inline]
```

Replaces the elements with num copies of val value.

.

Parameters

<i>num</i>	- number of elements to replace
<i>val</i>	- value to replace with

3.1.2.2 assign() [2/2]

```
template<typename T>
void myVector< T >::assign (
    const T * start,
    const T * end) [inline]
```

Replaces the elements with elements from range from start to end

.

Parameters

<i>start</i>	- pointer to a start of a range
<i>end</i>	- pointer to an end of a range

3.1.2.3 at()

```
template<typename T>
T & myVector< T >::at (
    const int & index) const [inline]
```

Returns the element at given index (with bounds checking)

.

Parameters

<i>index</i>	- index of the element
--------------	------------------------

Returns

Element at given index (template typename T)

3.1.2.4 back()

```
template<typename T>
T & myVector< T >::back () const [inline]
```

Returns the last element of the vector

.

Returns

Last element of the vector (template typename T)

3.1.2.5 begin()

```
template<typename T>
T * myVector< T >::begin () [inline]
```

Returns the pointer to the beginning of the vector

.

Returns

Pointer to the beginning of the vector (template typename T*)

3.1.2.6 capacity()

```
template<typename T>
int myVector< T >::capacity () const [inline]
```

Returns the capacity of the vector

.

Returns

Capacity of vector (integer)

3.1.2.7 cbegin()

```
template<typename T>
T * myVector< T >::cbegin () const [inline]
```

Returns the pointer to the beginning of the vector, but as const

.

Returns

Pointer to the beginning of the vector (template typename T*)

3.1.2.8 cend()

```
template<typename T>
T * myVector< T >::cend () const [inline]
```

Returns the pointer to element past the end of the vector, but as const

.

Returns

Pointer to element past the end of the vector (template typename T*)

3.1.2.9 clear()

```
template<typename T>
void myVector< T >::clear () [inline]
```

Clears the vector. Frees the old memory, but the capacity remains the same.

3.1.2.10 data()

```
template<typename T>
T * myVector< T >::data () const [inline]
```

Returns pointer to the underlying array.

Returns

Pointer to start of the data_array

3.1.2.11 empty()

```
template<typename T>
bool myVector< T >::empty () const [inline]
```

Checks if the vector is empty

.

Returns

True if the vector is empty, false otherwise (bool)

3.1.2.12 end()

```
template<typename T>
T * myVector< T >::end () [inline]
```

Returns the pointer to element past the end of the vector

.

Returns

Pointer to element past the end of the vector (template typename T*)

3.1.2.13 front()

```
template<typename T>
T & myVector< T >::front () const [inline]
```

Returns the first element of the vector

.

Returns

First element of the vector (template typename T)

3.1.2.14 insert() [1/2]

```
template<typename T>
void myVector< T >::insert (
    const T * pos,
    const int num,
    const T & val) [inline]
```

Inserts a num copies of element val into place before position pos.

Parameters

<i>pos</i>	- pointer to position
<i>num</i>	- number of copies to insert
<i>val</i>	- value to insert

3.1.2.15 insert() [2/2]

```
template<typename T>
void myVector< T >::insert (
    const T * pos,
    const T & val) [inline]
```

Inserts an copy of element val into place before position pos.

Parameters

<i>pos</i>	- pointer to position
<i>val</i>	- value to insert

3.1.2.16 operator!=(())

```
template<typename T>
bool myVector< T >::operator!= (
    const myVector< T > & second) const [inline]
```

Compares two vectors. If the sizes are different, it returns true. If the sizes are the same, it compares the elements.

.

Parameters

<i>second</i>	- vector to compare with
---------------	--------------------------

Returns

True if the vectors are not equal, false otherwise (bool)

3.1.2.17 operator=() [1/3]

```
template<typename T>
myVector & myVector< T >::operator= (
    const myVector< T > & other) [inline]
```

Assigns values from given vector to the vector. Values in original vector are not moved

.

Parameters

<i>other</i>	- vector to copy from
--------------	-----------------------

Returns

the vector

See also

[myVector\(const myVector& other\)](#)

3.1.2.18 operator=() [2/3]

```
template<typename T>
myVector & myVector< T >::operator= (
    myVector< T > && other) [inline]
```

Assigns values from given vector to the vector and "moves" the values from it

.

Parameters

<i>other</i>	- vector to move from
--------------	-----------------------

Returns

the vector

See also

[myVector\(myVector&& other\)](#)

3.1.2.19 operator=() [3/3]

```
template<typename T>
myVector myVector< T >::operator= (
    std::initializer_list< T > list) [inline]
```

Assigns values from given initializer list to the vector

.

Parameters

<i>list</i>	- initializer list to copy from
-------------	---------------------------------

Returns

the vector

3.1.2.20 operator==()

```
template<typename T>
bool myVector< T >::operator== (
    const myVector< T > & second) const [inline]
```

Compares two vectors. If the sizes are different, it returns false. If the sizes are the same, it compares the elements.

.

Returns

True if the vectors are equal, false otherwise (bool)

3.1.2.21 operator[]()

```
template<typename T>
T & myVector< T >::operator[] (
    const int & index) const [inline], [noexcept]
```

Returns the element at given index

.

Parameters

<i>index</i>	- index of the element
--------------	------------------------

Returns

Element at given index (template typename T)

3.1.2.22 output()

```
template<typename T>
std::wstring myVector< T >::output () const [inline]
```

Outputs the vector to a string (for testing purposes)

.

Returns

String with the vector values (std::wstring)

3.1.2.23 pop_back()

```
template<typename T>
void myVector< T >::pop_back () [inline]
```

Removes the last element of the vector.

3.1.2.24 push_back()

```
template<typename T>
void myVector< T >::push_back (
    const T & value) [inline]
```

Adds an element to the end of the vector. If the vector is full, it doubles the capacity.

.

Parameters

<i>value</i>	- value to add
--------------	----------------

3.1.2.25 reserve()

```
template<typename T>
void myVector< T >::reserve (
    const int & num) [inline]
```

Reserves memory for the vector. If the given number is smaller than the current capacity, it does nothing

.

Parameters

<i>num</i>	- number of elements to reserve memory for
------------	--

3.1.2.26 shrink_to_fit()

```
template<typename T>
void myVector< T >::shrink_to_fit () [inline]
```

Shrinks the vector to fit the current size. If the capacity is equal to the size, it does nothing.

3.1.2.27 size()

```
template<typename T>
int myVector< T >::size () const [inline]
```

Returns the size of the vector

.

Returns

Size of vector (integer)

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

- [myvector.h](#)

Chapter 4

File Documentation

4.1 myvector.h File Reference

```
#include <iostream>
#include <stdexcept>
#include <memory>
#include <string>
#include <algorithm>
#include <initializer_list>
```

Classes

- class `myVector< T >`

4.2 myvector.h

[Go to the documentation of this file.](#)

```
00001 #pragma once
00002 #include <iostream>
00003 #include <stdexcept>
00004 #include <memory>
00005 #include <string>
00006 #include <algorithm>
00007 #include <initializer_list>
00008
00009 template<typename T>
00010 class myVector {
00011 private:
00012     int size_;
00013     int capacity_;
00014     T* data_;
00015     T* begin_;
00016     T* end_;
00017 public:
00021     myVector() { // basic constructor
00022         size_ = 0;
00023         capacity_ = 10;
00024         data_ = new T[capacity_];
00025         begin_ = data_;
00026         end_ = data_;
00027     }
00033     myVector(const int& num, const T& val) { // fill constructor
00034         size_ = num;
00035         capacity_ = num;
```

```

00036         data_ = new T[capacity_];
00037         for (int i = 0; i < num; i++) {
00038             data_[i] = val;
00039         }
00040         begin_ = &data_[0];
00041         end_ = &data_[size_];
00042     }
00043 myVector(const T* begin, const T* end) { //range constructor
00044     size_ = end - begin;
00045     capacity_ = size_;
00046     data_ = new T[capacity_];
00047     for (int i = 0; i < size_; i++) {
00048         data_[i] = *(begin + i);
00049     }
00050     begin_ = &data_[0];
00051     end_ = &data_[size_];
00052 }
00053 myVector(const myVector& other) { // copy constructor
00054     size_ = other.size_;
00055     capacity_ = other.capacity_;
00056     data_ = new T[capacity_];
00057     for (int i = 0; i < size_; i++) {
00058         data_[i] = other.data_[i];
00059     }
00060     begin_ = &data_[0];
00061     end_ = &data_[size_];
00062 }
00063 myVector(myVector&& other) { //move constructor
00064     size_ = other.size_;
00065     capacity_ = other.capacity_;
00066     data_ = other.data_;
00067     begin_ = &data_[0];
00068     end_ = &data_[size_];
00069     other.data_ = nullptr;
00070     other.size_ = 0;
00071     other.capacity_ = 0;
00072     other.begin_ = nullptr;
00073     other.end_ = nullptr;
00074 }
00075 myVector(std::initializer_list<T> list) { //initializer list constructor
00076     size_ = static_cast<int>(list.size());
00077     capacity_ = size_;
00078     data_ = new T[capacity_];
00079     std::copy(list.begin(), list.end(), data_);
00080     begin_ = &data_[0];
00081     end_ = &data_[size_];
00082 }
00083 myVector operator=(std::initializer_list<T> list) { // operator = with initializer list
00084     this->~myVector();
00085     size_ = static_cast<int>(list.size());
00086     capacity_ = size_;
00087     data_ = new T[capacity_];
00088     std::copy(list.begin(), list.end(), data_);
00089     begin_ = &data_[0];
00090     end_ = &data_[size_];
00091     return *this;
00092 }
00093 myVector& operator= (const myVector& other) { // copy assignment
00094     if (this == &other) return *this;
00095     this->~myVector();
00096     size_ = other.size_;
00097     capacity_ = other.capacity_;
00098     data_ = new T[capacity_];
00099     for (int i = 0; i < size_; i++) {
00100         data_[i] = other.data_[i];
00101     }
00102     begin_ = &data_[0];
00103     end_ = &data_[size_];
00104     return *this;
00105 }
00106 myVector& operator= (myVector&& other) { // move assignment
00107     if (this == &other) return *this;
00108     this->~myVector();
00109     size_ = other.size_;
00110     capacity_ = other.capacity_;
00111     data_ = other.data_;
00112     begin_ = &data_[0];
00113     end_ = &data_[size_];
00114     other.data_ = nullptr;
00115     other.size_ = 0;
00116     other.capacity_ = 0;
00117     other.end_ = nullptr;
00118     other.begin_ = nullptr;
00119     return *this;
00120 }
00121 ~myVector() {
00122     size_ = 0;

```

```

00160         capacity_ = 0;
00161         delete[] data_;
00162         begin_ = nullptr;
00163         end_ = nullptr;
00164     }
00169     int size() const {
00170         return size_;
00171     }
00176     int capacity() const {
00177         return capacity_;
00178     }
00183     T* begin() {
00184         return begin_;
00185     }
00190     T* cbegin() const {
00191         return begin_;
00192     }
00197     T* end() {
00198         return end_;
00199     }
00204     T* cend() const {
00205         return end_;
00206     }
00211     T& front() const {
00212         if (this->empty()) throw std::out_of_range("Vector is empty");
00213         return *begin_;
00214     }
00219     T& back() const {
00220         if (this->empty()) throw std::out_of_range("Vector is empty");
00221         return *(end_ - 1);
00222     }
00228     T& operator[] (const int& index) const noexcept {
00229         //if (index < 0 || index >= size_) throw std::out_of_range("Out of range!");
00230         return this->data_[index];
00231     }
00237     T& at(const int& index) const {
00238         if (index < 0 || index >= size_) {
00239             throw std::out_of_range("Out of range!");
00240             std::wcerr << "Out of range!";
00241             std::terminate();
00242         }
00243         return data_[index];
00244     }
00249     void push_back(const T& value) {
00250         if (capacity_ == 0) {
00251             capacity_ = 10;
00252             data_ = new T[capacity_];
00253         }
00254         if (size_ == 0) {
00255             data_[0] = value;
00256             begin_ = &data_[0];
00257             end_ = &data_[0] + 1;
00258             size_ = 1;
00259         }
00260         else {
00261             if (size_ >= capacity_) {
00262                 capacity_ = capacity_ * 2;
00263                 T* temp = new T[capacity_];
00264                 for (int i = 0; i < size_; i++) {
00265                     temp[i] = data_[i];
00266                 }
00267                 delete[] data_;
00268                 data_ = temp;
00269                 begin_ = &data_[0];
00270                 end_ = &data_[size_];
00271                 temp = nullptr;
00272             }
00273             data_[size_] = value;
00274             end_ = &data_[size_] + 1;
00275             size_++;
00276         }
00277     }
00278 }
00282     void pop_back() {
00283         if (size_ == 0) return;
00284         ~data_[size_ - 1];
00285         size_ = size_ - 1;
00286         end_ = end_ - 1;
00287     }
00292     bool empty() const {
00293         if (begin_ == end_) return true;
00294         return false;
00295     }
00300     void reserve(const int& num) {
00301         if (num <= capacity_) return;
00302         capacity_ = num;
00303         T* temp = new T[capacity_];

```

```

00304         for (int i = 0; i < size_; i++) {
00305             temp[i] = data_[i];
00306         }
00307         delete[] data_;
00308         data_ = temp;
00309         begin_ = &data_[0];
00310         end_ = &data_[size_];
00311         temp = nullptr;
00312     }
00316 void shrink_to_fit() {
00317     if (capacity_ <= size_) return;
00318     capacity_ = size_;
00319     T* temp = new T[capacity_];
00320     for (int i = 0; i < size_; i++) {
00321         temp[i] = data_[i];
00322     }
00323     delete[] data_;
00324     data_ = temp;
00325     begin_ = &data_[0];
00326     end_ = &data_[size_];
00327     temp = nullptr;
00328 }
00332 void clear() {
00333     if (size_ == 0) return;
00334     delete[] data_;
00335     data_ = new T[capacity_];
00336     size_ = 0;
00337     begin_ = data_;
00338     end_ = begin_;
00339 }
00344 std::wstring output() const {
00345     std::wstringstream out;
00346     out << size_;
00347     if (size_ != 0) {
00348         out << *begin_ << *(end_ - 1);
00349         for (int i = 0; i < size_; i++) {
00350             out << data_[i];
00351         }
00352     }
00353     std::wstring str;
00354     str = out.str();
00355     return str;
00356 }
00361 bool operator==(const myVector& second) const {
00362     if (size_ != second.size_) return false;
00363     for (int i = 0; i < size_; i++) {
00364         if (data_[i] != second.data_[i]) return false;
00365     }
00366     return true;
00367 }
00373 bool operator!=(const myVector& second) const {
00374     if (size_ != second.size_) return true;
00375     for (int i = 0; i < size_; i++) {
00376         if (data_[i] != second.data_[i]) return true;
00377     }
00378     return false;
00379 }
00385 void assign(const int& num, const T& val) {
00386     if (num <= capacity_) {
00387         if (num > size_) {
00388             size_ = num;
00389             end_ = &data_[size_];
00390         }
00391         for (int i = 0; i < num; i++) {
00392             ~data_[i];
00393             data_[i] = val;
00394         }
00395     }
00396     else {
00397         this->~myVector();
00398         capacity_ = num;
00399         size_ = num;
00400         data_ = new T[capacity_];
00401         for (int i = 0; i < size_; i++) {
00402             data_[i] = val;
00403         }
00404         begin_ = &data_[0];
00405         end_ = &data_[size_];
00406     }
00407 }
00408
00414 void assign(const T* start, const T* end) {
00415     int num = end - start;
00416     if (num <= capacity_) {
00417         if (num > size_) {
00418             size_ = num;
00419             end_ = &data_[size_];

```



```

00420         }
00421         for (int i = 0; i < num; i++) {
00422             ~data_[i];
00423             data_[i] = *(start+i);
00424         }
00425     }
00426     else {
00427         this->~myVector();
00428         capacity_ = num;
00429         size_ = num;
00430         data_ = new T[capacity_];
00431         for (int i = 0; i < size_; i++) {
00432             data_[i] = *(start + i);
00433         }
00434         begin_ = &data_[0];
00435         end_ = &data_[size_];
00436     }
00437 }
00442 T* data() const {
00443     return &data_[0];
00444 }
00450 void insert(const T* pos, const T& val) {
00451     int index = pos - begin_;
00452     if (size_ + 1 <= capacity_) {
00453         for (auto it = end_; it != pos; it--) {
00454             *it = *(it - 1);
00455         }
00456         data_[index] = val;
00457         size_++;
00458         end_ = &data_[size_];
00459     }
00460     else {
00461         capacity_ = capacity_ * 2;
00462         T* temp = new T[capacity_];
00463         for (int i = 0; i != index; i++) {
00464             temp[i] = data_[i];
00465         }
00466         temp[index] = val;
00467         for (int i = index; i < size_; i++) {
00468             temp[i+1] = data_[i];
00469         }
00470         delete[] data_;
00471         data_ = temp;
00472         begin_ = data_;
00473         size_++;
00474         end_ = &data_[size_];
00475     }
00476 }
00483 void insert(const T* pos, const int num, const T& val) {
00484     int index = pos - begin_;
00485     if (size_ + num <= capacity_) {
00486         for (auto it = end_ + num - 1; it != pos; it--) {
00487             *it = *(it - num);
00488         }
00489         for (int i = num; i > 0; i--) {
00490             data_[index + i - 1] = val;
00491         }
00492         size_ = size_ + num;
00493         end_ = &data_[size_];
00494     }
00495     else {
00496         capacity_ = capacity_ + num;
00497         T* temp = new T[capacity_];
00498         for (int i = 0; i != index; i++) {
00499             temp[i] = data_[i];
00500         }
00501         for (int i = index; i < index + num; i++) {
00502             temp[i] = val;
00503         }
00504         for (int i = index; i < size_; i++) {
00505             temp[i + num] = data_[i];
00506         }
00507         delete[] data_;
00508         data_ = temp;
00509         begin_ = data_;
00510         size_ = size_ + num;
00511         end_ = &data_[size_];
00512     }
00513 }
00514 };

```


Index

~myVector
 myVector< T >, 8

assign
 myVector< T >, 9

at
 myVector< T >, 9

back
 myVector< T >, 10

begin
 myVector< T >, 10

capacity
 myVector< T >, 10

cbegin
 myVector< T >, 10

cend
 myVector< T >, 11

clear
 myVector< T >, 11

data
 myVector< T >, 11

empty
 myVector< T >, 11

end
 myVector< T >, 11

front
 myVector< T >, 12

insert
 myVector< T >, 12

myVector
 myVector< T >, 7, 8

myVector< T >, 5
 ~myVector, 8

 assign, 9

 at, 9

 back, 10

 begin, 10

 capacity, 10

 cbegin, 10

 cend, 11

 clear, 11

 data, 11

 empty, 11

 end, 11

 front, 12

 insert, 12

 myVector, 7, 8

 operator!=, 13

 operator=, 13, 14

 operator==, 14

 operator[], 14

 output, 15

 pop_back, 15

 push_back, 15

 reserve, 15

 shrink_to_fit, 16

 size, 16

myvector.h, 17

operator!=
 myVector< T >, 13

operator=
 myVector< T >, 13, 14

operator==
 myVector< T >, 14

operator[]
 myVector< T >, 14

output
 myVector< T >, 15

pop_back
 myVector< T >, 15

push_back
 myVector< T >, 15

reserve
 myVector< T >, 15

shrink_to_fit
 myVector< T >, 16

size
 myVector< T >, 16