Classe vector

Artigo • 02/09/2023

A classe de vetor da biblioteca padrão C++ é um modelo de classe para contêineres de sequência. Um vetor armazena elementos de um determinado tipo de maneira linear e permite o acesso aleatório rápido a qualquer elemento. Um vetor é o contêiner preferencial para uma sequência quando o desempenho de acesso aleatório é reduzido.

Sintaxe

C++

```
template <class Type, class Allocator = allocator<Type>>
class vector
```

Parâmetros

Type

O tipo de dados do elemento a ser armazenado no vetor

Allocator

O tipo que representa o objeto allocator armazenado que encapsula detalhes sobre a alocação e a desalocação de memória do vetor. Esse argumento é opcional e o valor padrão é `allocator<Type>`.

Comentários

Os vetores permitem inserções e exclusões em tempo constante no final da sequência. Inserir ou excluir elementos no meio de um vetor exige tempo linear. O contêiner da classe deque é mais rápido nas inserções e exclusões no início e no final de uma sequência. O contêiner da classe list é mais rápido nas inserções e exclusões em qualquer parte de uma sequência.

A realocação do vetor ocorre quando uma função membro deve aumentar a sequência contida no objeto vector além da sua capacidade de armazenamento atual. Outras inserções e exclusões podem alterar vários endereços de armazenamento na sequência. Em todos esses casos, iteradores ou referências que apontam para partes alteradas da sequência tornam-se inválidos. Se nenhuma realocação ocorrer, somente os iteradores e as referências antes do ponto de inserção/exclusão permanecerão válidos.

A classe vector<bool> é uma especialização parcial de vector para elementos do tipo `bool`. Ela tem um alocador para o tipo subjacente usado pela especialização.

A classe de referência vector<bool> é uma classe aninhada cujos objetos podem fornecer referências aos elementos (bits únicos) em um objeto `vector<bool>`.

Membros

Construtores

Nome	Descrição
vector	Constrói um vetor de tamanho específico com elementos de um valor específico, com um `allocator` específico ou como uma cópia de algum outro vetor.

Typedefs

Nome	Descrição
`[allocator_type]` (#allocator_type)	Um tipo que representa a classe `allocator` do objeto vector.
const_iterator	Um tipo que fornece um iterador de acesso aleatório que pode ler um elemento `const` em um vetor.
const_pointer	Um tipo que fornece um ponteiro para um elemento `const` em um vetor.
const_reference	Um tipo que fornece uma referência a um elemento `const` armazenado em um vetor. Ele é usado para ler e fazer operações `const`.
const_reverse_iterator	Um tipo que fornece um iterador de acesso aleatório que pode ler qualquer elemento `const` no vetor.
difference_type	Um tipo que fornece a diferença entre os endereços de dois elementos em um vetor.
iterator	Um tipo que fornece um iterador de acesso aleatório que pode ler ou modificar qualquer elemento em um vetor.
pointer	Um tipo que fornece um ponteiro para um elemento em um vetor.
reference	Um tipo que fornece uma referência a um elemento armazenado em um vetor.

Nome	Descrição
reverse_iterator	Um tipo que fornece um iterador de acesso aleatório que pode ler ou modificar qualquer elemento em um vetor invertido.
size_type	Um tipo que conta o número de elementos em um vetor.
value_type	Um tipo que representa o tipo de dados armazenado em um vetor.

Funções

Nome	Descrição
assign	Apaga um vetor e copia os elementos especificados para o vetor vazio.
at	Retorna uma referência ao elemento em um local especificado no vetor.
back	Retorna uma referência ao último elemento do vetor.
begin	Retorna um iterador de acesso aleatório para o primeiro elemento no vetor.
capacity	Retorna o número de elementos que o vetor pode conter sem alocar mais armazenamento.
cbegin	Retorna um iterador de acesso aleatório const para o primeiro elemento no vetor.
cend	Retorna um iterador de acesso aleatório const que aponta para imediatamente após o fim do vetor.
crbegin	Retorna um iterador const para o primeiro elemento em um vetor invertido.
crend	Retorna um iterador const para o final de um vetor invertido.
clear	Apaga os elementos do vetor.
data	Retorna um ponteiro para o primeiro elemento no vetor.
emplace	Insere um elemento construído no local no vetor em uma posição especificada.
emplace_back	Adiciona um elemento construído no local ao final do vetor.
empty	Testa se o contêiner do vetor está vazio.
end	Retorna um iterador de acesso aleatório que aponta para o final do vetor.
erase	Remove um elemento ou um intervalo de elementos em um vetor das posições especificadas.
front	Retorna uma referência ao primeiro elemento em um vetor.

Nome	Descrição
get_allocator	Retorna um objeto para a classe `allocator` usada por um vetor.
insert	Insere um elemento ou uma série de elementos no vetor em uma posição especificada.
max_size	Retorna o tamanho máximo do vetor.
pop_back	Exclui o elemento no final do vetor.
push_back	Adiciona um elemento ao final do vetor.
rbegin	Retorna um iterador para o primeiro elemento em um vetor invertido.
rend	Retorna um iterador para o final de um vetor invertido.
reserve	Reserva um tamanho mínimo de armazenamento para um objeto vector.
resize	Especifica um novo tamanho para um vetor.
shrink_to_fit	Descarta o excesso de capacidade.
size	Retorna o número de elementos no vetor.
swap	Troca os elementos de dois vetores.

Operadores

Nome	Descrição
operator[]	Retorna uma referência para o elemento de vetor em uma posição especificada.
operator=	Substitui os elementos do vetor por uma cópia de outro vetor.

`allocator_type`

Um tipo que representa a classe allocator do objeto vector.

C++

```
typedef Allocator allocator_type;
```

Comentários

`allocator_type` é um sinônimo do parâmetro de modelo `Allocator`.

Exemplo

Confira get_allocator, para ver um exemplo de uso de `allocator_type`.

`assign`

Apaga um vetor e copia os elementos especificados para o vetor vazio.

C++

```
void assign(size_type count, const Type& value);
void assign(initializer_list<Type> init_list);

template <class InputIterator>
void assign(InputIterator first, InputIterator last);
```

Parâmetros

`first`

A posição do primeiro elemento no intervalo de elementos a ser copiado.

`last`

A posição do primeiro elemento após o intervalo de elementos a ser copiado.

`count`

O número de cópias de um elemento sendo inserido no vetor.

`value`

O valor do elemento sendo inserido no vetor.

`init_list`

A initializer_list que contém os elementos a serem inseridos.

Comentários

Primeiro, `assign` apaga todos os elementos existentes em um vetor. Depois, `assign` insere um intervalo especificado de elementos do vetor original em um vetor ou insere cópias de um novo elemento de um valor especificado em um vetor.

Exemplo

C++

```
/ vector_assign.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main()
{
    using namespace std;
    vector<int> v1, v2, v3;

    v1.push_back(10);
    v1.push_back(20);
    v1.push_back(30);
    v1.push_back(40);
    v1.push_back(50);

    cout << "v1 = ";
    for (auto& v : v1){
        cout << v << " ";
    }
    cout << endl;

    v2.assign(v1.begin(), v1.end());
    cout << "v2 = ";
    for (auto& v : v2){
        cout << v << " ";
    }
    cout << endl;

    v3.assign(7, 4);
    cout << "v3 = ";
    for (auto& v : v3){
        cout << v << " ";
    }
    cout << endl;

    v3.assign({ 5, 6, 7 });
    for (auto& v : v3){
        cout << v << " ";
    }
    cout << endl;
}
```

at

Retorna uma referência ao elemento em um local especificado no vetor.

C++

```
reference at(size_type position);
```

```
const_reference at(size_type position) const;
```

Parâmetros

position
O número da posição ou subscrito do elemento para referência no vetor.

Retornar valor

Uma referência ao elemento subscrito no argumento. Se *position* for maior que o tamanho do vetor, `at` gerará uma exceção.

Comentários

Se o valor retornado de `at` for atribuído a `const_reference`, o objeto vector não poderá ser modificado. Se o valor retornado de `at` for atribuído a `reference`, o objeto vector poderá ser modificado.

Exemplo

C++

```
// vector_at.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;

   v1.push_back( 10 );
   v1.push_back( 20 );

   const int &i = v1.at( 0 );
   int &j = v1.at( 1 );
   cout << "The first element is " << i << endl;
   cout << "The second element is " << j << endl;
}
```

Output

```
The first element is 10
The second element is 20
```

back

Retorna uma referência ao último elemento do vetor.

C++

```
reference back();

const_reference back() const;
```

Retornar valor

O último elemento do vetor. Se o vetor estiver vazio, o valor retornado será indefinido.

Comentários

Se o valor retornado de `back` for atribuído a `const_reference`, o objeto vector não poderá ser modificado. Se o valor retornado de `back` for atribuído a `reference`, o objeto vector poderá ser modificado.

Quando compilado usando _ITERATOR_DEBUG_LEVEL definido como 1 ou 2, um erro de runtime ocorrerá se você tentar acessar um elemento em um vetor vazio. Para obter mais informações, confira Iteradores verificados.

Exemplo

C++

```
// vector_back.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main() {
   using namespace std;
   vector <int> v1;

   v1.push_back( 10 );
   v1.push_back( 11 );
```

```
    int& i = v1.back( );
    const int& ii = v1.front( );

    cout << "The last integer of v1 is " << i << endl;
    i--;
    cout << "The next-to-last integer of v1 is "<< ii << endl;
}
```

begin

Retorna um iterador de acesso aleatório para o primeiro elemento no vetor.

C++

```
const_iterator begin() const;

iterator begin();
```

Retornar valor

Um iterador de acesso aleatório que endereça o primeiro elemento em `vector` ou ao local que vem após um `vector` vazio. Sempre compare o valor retornado com vector::end para garantir que ele seja válido.

Comentários

Se o valor retornado de `begin` for atribuído a um vector::const_iterator, o objeto `vector` não poderá ser modificado. Se o valor retornado de `begin` for atribuído a um vector::iterator, o objeto `vector` pode ser modificado.

Exemplo

C++

```
// vector_begin.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main()
{
    using namespace std;
    vector<int> c1;
    vector<int>::iterator c1_Iter;
```

```
    vector<int>::const_iterator c1_cIter;

    c1.push_back(1);
    c1.push_back(2);

    cout << "The vector c1 contains elements:";
    c1_Iter = c1.begin();
    for (; c1_Iter != c1.end(); c1_Iter++)
    {
        cout << " " << *c1_Iter;
    }
    cout << endl;

    cout << "The vector c1 now contains elements:";
    c1_Iter = c1.begin();
    *c1_Iter = 20;
    for (; c1_Iter != c1.end(); c1_Iter++)
    {
        cout << " " << *c1_Iter;
    }
    cout << endl;

    // The following line would be an error because iterator is const
    // *c1_cIter = 200;
}
```

Output

```
The vector c1 contains elements: 1 2
The vector c1 now contains elements: 20 2
```

`capacity`

Retorna o número de elementos que o vetor pode conter sem alocar mais armazenamento.

C++

```
size_type capacity() const;
```

Retornar valor

O tamanho atual do armazenamento alocado para o vetor.

Comentários

A função membro resize será mais eficiente se houver alocação de memória suficiente para acomodá-la. Use a função membro reserve para especificar a quantidade de memória alocada.

Exemplo

C++

```
// vector_capacity.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;

   v1.push_back( 1 );
   cout << "The length of storage allocated is "
        << v1.capacity( ) << "." << endl;

   v1.push_back( 2 );
   cout << "The length of storage allocated is now "
        << v1.capacity( ) << "." << endl;
}
```

Output

```
The length of storage allocated is 1.
The length of storage allocated is now 2.
```

cbegin

Retorna um iterador `const` que trata o primeiro elemento no intervalo.

C++

```
const_iterator cbegin() const;
```

Retornar valor

Um iterador de acesso aleatório `const` que aponta o primeiro elemento do intervalo ou o local logo após o fim de um intervalo vazio (para um intervalo vazio, `cbegin() ==`

`cend()`).

Comentários

Com o valor retornado de `cbegin`, os elementos no intervalo não podem ser modificados.

Você pode usar essa função membro no lugar da função membro `begin()`, de modo a garantir que o valor de retorno seja `const_iterator`. Normalmente, ela é usada com a palavra-chave de dedução de tipo auto, conforme mostrado no exemplo a seguir. No exemplo, considere `Container` como um contêiner modificável (não **`const`**) de qualquer tipo, que dá suporte para `begin()` e `cbegin()`.

C++

```
auto i1 = Container.begin();
// i1 is Container<T>::iterator
auto i2 = Container.cbegin();

// i2 is Container<T>::const_iterator
```

cend

Retorna um iterador **`const`** que ultrapassa o fim que aponta para o elemento após o último elemento do vetor.

C++

```
const_iterator cend() const;
```

Retornar valor

O iterador `const` que ultrapassa o fim para o vetor. Ele aponta para o elemento que segue o último elemento do vetor. Esse elemento é um espaço reservado e não deve ser desreferenciado. Use-o apenas para comparações. Se o vetor estiver vazio, `vector::cend() == vector::cbegin()`.

Comentários

`cend` é usado para testar se um iterador passou do fim de seu intervalo.

Você pode usar essa função membro no lugar da função membro `end()`, de modo a garantir que o valor de retorno seja `const_iterator`. Normalmente, ela é usada com a palavra-chave de dedução de tipo auto, conforme mostrado no exemplo a seguir. No exemplo, considere `Container` como um contêiner modificável (não **`const`**) de qualquer tipo, que dá suporte para `end()` e `cend()`.

C++

```
auto i1 = Container.end();
// i1 is Container<T>::iterator
auto i2 = Container.cend();

// i2 is Container<T>::const_iterator
```

O valor retornado por `cend` não deve ser desreferenciado. Use-o apenas para comparações.

`clear`

Apaga os elementos do vetor.

C++

```
void clear();
```

Exemplo

C++

```
// vector_clear.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;

   v1.push_back( 10 );
   v1.push_back( 20 );
   v1.push_back( 30 );

   cout << "The size of v1 is " << v1.size( ) << endl;
   v1.clear( );
```

```
    cout << "The size of v1 after clearing is " << v1.size( ) << endl;
}
```

Output

```
The size of v1 is 3
The size of v1 after clearing is 0
```

const_iterator

Um tipo que fornece um iterador de acesso aleatório que pode ler um elemento **const** em um vetor.

C++

```
typedef implementation-defined const_iterator;
```

Comentários

Um tipo `const_iterator` não pode ser usado para modificar o valor de um elemento.

Exemplo

Veja o exemplo de back para obter um exemplo que usa `const_iterator`.

const_pointer

Um tipo que fornece um ponteiro para um elemento **const** em um vetor.

C++

```
typedef typename Allocator::const_pointer const_pointer;
```

Comentários

Um tipo `const_pointer` não pode ser usado para modificar o valor de um elemento.

Um iterator é usado com mais frequência para acessar um elemento de vetor.

const_reference

Um tipo que fornece uma referência a um elemento **const** armazenado em um vetor. Ele é usado para ler e fazer operações **const**.

C++

```
typedef typename Allocator::const_reference const_reference;
```

Comentários

Um tipo `const_reference` não pode ser usado para modificar o valor de um elemento.

Exemplo

C++

```
// vector_const_ref.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;

   v1.push_back( 10 );
   v1.push_back( 20 );

   const vector <int> v2 = v1;
   const int &i = v2.front( );
   const int &j = v2.back( );
   cout << "The first element is " << i << endl;
   cout << "The second element is " << j << endl;

   // The following line would cause an error as v2 is const
   // v2.push_back( 30 );
}
```

Output

```
The first element is 10
The second element is 20
```

const_reverse_iterator

Um tipo que fornece um iterador de acesso aleatório que pode ler qualquer elemento `const` no vetor.

C++

```
typedef std::reverse_iterator<const_iterator> const_reverse_iterator;
```

Comentários

Um tipo `const_reverse_iterator` não pode modificar o valor de um elemento e é usado para iterar no vetor em ordem inversa.

Exemplo

Consulte rbegin para obter um exemplo de como declarar e usar um iterador.

crbegin

Retorna um iterador const para o primeiro elemento em um vetor invertido.

C++

```
const_reverse_iterator crbegin() const;
```

Retornar valor

Um iterador de acesso aleatório reverso const que trata o primeiro elemento em um vector inverso ou que trata aquele que foi o último elemento no `vector` não reverso.

Comentários

Com o valor retornado `crbegin`, o objeto `vector` não pode ser modificado.

Exemplo

C++

```
// vector_crbegin.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;
   vector <int>::iterator v1_Iter;
   vector <int>::const_reverse_iterator v1_rIter;

   v1.push_back( 1 );
   v1.push_back( 2 );

   v1_Iter = v1.begin( );
   cout << "The first element of vector is "
        << *v1_Iter << "." << endl;

   v1_rIter = v1.crbegin( );
   cout << "The first element of the reversed vector is "
        << *v1_rIter << "." << endl;
}
```

Output

```
The first element of vector is 1.
The first element of the reversed vector is 2.
```

`crend`

Retorna um iterador reverso `const` que ultrapassa o fim e aponta para o elemento após o último elemento do vetor invertido.

C++

```
const_reverse_iterator crend() const;
```

Retornar valor

Um iterador reverso `const` que ultrapassa o fim para o vetor invertido. Ele aponta para o elemento que segue o último elemento do vetor invertido, que é o mesmo que o elemento antes do primeiro elemento do vetor não invertido. Esse elemento é um espaço reservado e não deve ser desreferenciado. Use-o apenas para comparações.

Comentários

`crend` é usado com um `vector` invertido, assim como vector::cend é usado com um `vector`.

Com o valor retornado de `crend` (adequadamente diminuído), o objeto `vector` não poderá ser modificado.

`crend` pode ser usado para testar se um iterador inverso alcançou o final de sua `vector`.

O valor retornado por `crend` não deve ser desreferenciado. Use-o apenas para comparações.

Exemplo

C++

```
// vector_crend.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;
   vector <int>::const_reverse_iterator v1_rIter;

   v1.push_back( 1 );
   v1.push_back( 2 );

   for ( v1_rIter = v1.rbegin( ) ; v1_rIter != v1.rend( ) ; v1_rIter++ )
      cout << *v1_rIter << endl;
}
```

Output

```
2
1
```

data

Retorna um ponteiro para o primeiro elemento no vetor.

C++

```
const_pointer data() const;

pointer data();
```

Retornar valor

Um ponteiro para o primeiro elemento no vector ou para o local após um `vector` vazio.

Exemplo

C++

```
// vector_data.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main()
{
    using namespace std;
    vector<int> c1;
    vector<int>::pointer c1_ptr;
    vector<int>::const_pointer c1_cPtr;

    c1.push_back(1);
    c1.push_back(2);

    cout << "The vector c1 contains elements:";
    c1_cPtr = c1.data();
    for (size_t n = c1.size(); 0 < n; --n, c1_cPtr++)
    {
        cout << " " << *c1_cPtr;
    }
    cout << endl;

    cout << "The vector c1 now contains elements:";
    c1_ptr = c1.data();
    *c1_ptr = 20;
    for (size_t n = c1.size(); 0 < n; --n, c1_ptr++)
    {
        cout << " " << *c1_ptr;
    }
    cout << endl;
}
```

Output

```
The vector c1 contains elements: 1 2
The vector c1 now contains elements: 20 2
```

difference_type

Um tipo que fornece a diferença entre dois iteradores que se referem a elementos no mesmo vetor.

C++

```
typedef typename Allocator::difference_type difference_type;
```

Comentários

Um `difference_type` também pode ser descrito como o número de elementos entre dois ponteiros, uma vez que um ponteiro para um elemento contém o seu endereço.

Um iterator é usado com mais frequência para acessar um elemento de vetor.

Exemplo

C++

```
// vector_diff_type.cpp
// compile with: /EHsc
#include <iostream>
#include <vector>
#include <algorithm>

int main( )
{
   using namespace std;

   vector <int> c1;
   vector <int>::iterator c1_Iter, c2_Iter;

   c1.push_back( 30 );
   c1.push_back( 20 );
   c1.push_back( 30 );
   c1.push_back( 10 );
   c1.push_back( 30 );
   c1.push_back( 20 );

   c1_Iter = c1.begin( );
   c2_Iter = c1.end( );
```

```
    vector <int>::difference_type   df_typ1, df_typ2, df_typ3;

    df_typ1 = count( c1_Iter, c2_Iter, 10 );
    df_typ2 = count( c1_Iter, c2_Iter, 20 );
    df_typ3 = count( c1_Iter, c2_Iter, 30 );
    cout << "The number '10' is in c1 collection " << df_typ1 << " times.\n";
    cout << "The number '20' is in c1 collection " << df_typ2 << " times.\n";
    cout << "The number '30' is in c1 collection " << df_typ3 << " times.\n";
}
```

Output

```
The number '10' is in c1 collection 1 times.
The number '20' is in c1 collection 2 times.
The number '30' is in c1 collection 3 times.
```

emplace

Insere um elemento construído no local no vetor em uma posição especificada.

C++

```
template <class... Types>
iterator emplace(
    const_iterator position,
    Types&&... args);
```

Parâmetros

position

A posição no vector em que o primeiro elemento é inserido.

args

Argumentos de construtor. A função infere qual sobrecarga de construtor deve ser invocada com base nos argumentos fornecidos.

Retornar valor

A função retorna um iterador que aponta para a posição em que o novo elemento foi inserido no `vector`.

Comentários

Uma operação de inserção pode ser cara; confira classe vector para ver uma discussão sobre o desempenho de `vector`.

Exemplo

C++

```
// vector_emplace.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;
   vector <int>::iterator Iter;

   v1.push_back( 10 );
   v1.push_back( 20 );
   v1.push_back( 30 );

   cout << "v1 =" ;
   for ( Iter = v1.begin( ) ; Iter != v1.end( ) ; Iter++ )
      cout << " " << *Iter;
   cout << endl;

// initialize a vector of vectors by moving v1
   vector < vector <int> > vv1;

   vv1.emplace( vv1.begin(), move( v1 ) );
   if ( vv1.size( ) != 0 && vv1[0].size( ) != 0 )
      {
      cout << "vv1[0] =";
      for (Iter = vv1[0].begin( ); Iter != vv1[0].end( ); Iter++ )
         cout << " " << *Iter;
      cout << endl;
      }
}
```

Output

```
v1 = 10 20 30
vv1[0] = 10 20 30
```

emplace_back

Adiciona um elemento construído no local ao final do vetor.

C++

```
template <class... Types>
void emplace_back(Types&&... args);
```

Parâmetros

args

Argumentos de construtor. A função infere qual sobrecarga de construtor deve ser invocada com base nos argumentos fornecidos.

Exemplo

C++

```
#include <vector>
struct obj
{
   obj(int, double) {}
};

int main()
{
   std::vector<obj> v;
   v.emplace_back(1, 3.14); // obj in created in place in the vector
}
```

`empty`

Testa se o vetor está vazio.

C++

```
bool empty() const;
```

Retornar valor

`true` se o vetor estiver vazio; **`false`** se o vetor não estiver vazio.

Exemplo

C++

```cpp
// vector_empty.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;

   v1.push_back( 10 );

   if ( v1.empty( ) )
      cout << "The vector is empty." << endl;
   else
      cout << "The vector is not empty." << endl;
}
```

Output

```
The vector is not empty.
```

end

Retorna um iterador que ultrapassa o fim que aponta para o elemento após o último elemento do vetor.

C++

```cpp
iterator end();

const_iterator end() const;
```

Retornar valor

Um iterador que ultrapassa o fim para o vetor. Ele aponta para o elemento que segue o último elemento do vetor. Esse elemento é um espaço reservado e não deve ser desreferenciado. Use-o apenas para comparações. Se o vetor estiver vazio, `vector::end() == vector::begin()`.

Comentários

Se o valor retornado de `end` for atribuído a uma variável de tipo `const_iterator`, o objeto vector não poderá ser modificado. Se o valor retornado de `end` for atribuído a uma variável de tipo `iterator`, o objeto vector poderá ser modificado.

Exemplo

C++

```
// vector_end.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>
int main( )
{
   using namespace std;
   vector <int> v1;
   vector <int>::iterator v1_Iter;

   v1.push_back( 1 );
   v1.push_back( 2 );

   for ( v1_Iter = v1.begin( ) ; v1_Iter != v1.end( ) ; v1_Iter++ )
      cout << *v1_Iter << endl;
}
```

Output

```
1
2
```

erase

Remove um elemento ou um intervalo de elementos em um vetor das posições especificadas.

C++

```
iterator erase(
    const_iterator position);

iterator erase(
    const_iterator first,
    const_iterator last);
```

Parâmetros

position

Posição do elemento a ser removido do vetor.

first

Posição do primeiro elemento removido do vetor.

last

Posição após o último elemento removido do vetor.

Retornar valor

Um iterador que designará o primeiro elemento restante além de todos os elementos removidos ou um ponteiro para o final do vetor se esse elemento não existir.

Exemplo

C++

```
// vector_erase.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;
   vector <int>::iterator Iter;

   v1.push_back( 10 );
   v1.push_back( 20 );
   v1.push_back( 30 );
   v1.push_back( 40 );
   v1.push_back( 50 );

   cout << "v1 =" ;
   for ( Iter = v1.begin( ) ; Iter != v1.end( ) ; Iter++ )
      cout << " " << *Iter;
   cout << endl;

   v1.erase( v1.begin( ) );
   cout << "v1 =";
   for ( Iter = v1.begin( ) ; Iter != v1.end( ) ; Iter++ )
      cout << " " << *Iter;
   cout << endl;
```

```
    v1.erase( v1.begin( ) + 1, v1.begin( ) + 3 );
    cout << "v1 =";
    for ( Iter = v1.begin( ) ; Iter != v1.end( ) ; Iter++ )
       cout << " " << *Iter;
    cout << endl;
}
```

Output

```
v1 = 10 20 30 40 50
v1 = 20 30 40 50
v1 = 20 50
```

`front`

Retorna uma referência ao primeiro elemento em um vetor.

C++

```
reference front();

const_reference front() const;
```

Retornar valor

Uma referência ao primeiro elemento no objeto vector. Se o vetor estiver vazio, o valor retornado será indefinido.

Comentários

Se o valor retornado de `front` for atribuído a `const_reference`, o objeto vector não poderá ser modificado. Se o valor retornado de `front` for atribuído a **`reference`**, o objeto vector poderá ser modificado.

Quando compilado usando _ITERATOR_DEBUG_LEVEL definido como 1 ou 2, um erro de runtime ocorrerá se você tentar acessar um elemento em um vetor vazio. Para obter mais informações, confira Iteradores verificados.

Exemplo

C++

```
// vector_front.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;

   v1.push_back( 10 );
   v1.push_back( 11 );

   int& i = v1.front( );
   const int& ii = v1.front( );

   cout << "The first integer of v1 is "<< i << endl;
   // by incrementing i, we move the front reference to the second element
   i++;
   cout << "Now, the first integer of v1 is "<< i << endl;
}
```

get_allocator

Retorna uma cópia do objeto allocator usado para construir o vetor.

C++

```
Allocator get_allocator() const;
```

Retornar valor

O alocador usado pelo vetor.

Comentários

Os alocadores da classe vector especificam como a classe gerencia o armazenamento. Os alocadores padrão fornecidos com as classes de contêiner da Biblioteca Padrão C++ são suficientes para a maioria das necessidades de programação. Gravando e usando sua própria classe de alocador é um recurso avançado do C++.

Exemplo

C++

```
// vector_get_allocator.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   // The following lines declare objects that use the default allocator.
   vector<int> v1;
   vector<int, allocator<int> > v2 = vector<int, allocator<int> >
(allocator<int>( )) ;

   // v3 will use the same allocator class as v1
   vector <int> v3( v1.get_allocator( ) );

   vector<int>::allocator_type xvec = v3.get_allocator( );
   // You can now call functions on the allocator class used by vec
}
```

`insert`

Insere um elemento, vários elementos ou um intervalo de elementos no vetor em uma posição especificada.

C++

```
iterator insert(
    const_iterator position,
    const Type& value);

iterator insert(
    const_iterator position,
    Type&& value);

void insert(
    const_iterator position,
    size_type count,
    const Type& value);

template <class InputIterator>
void insert(
    const_iterator position,
    InputIterator first,
    InputIterator last);
```

Parâmetros

position

A posição no vetor em que o primeiro elemento é inserido.

value

O valor do elemento sendo inserido no vetor.

count

O número de elementos sendo inseridos no vetor.

first

A posição do primeiro elemento no intervalo de elementos a serem copiados.

last

A posição do primeiro elemento além do intervalo de elementos a serem copiados.

Retornar valor

As duas primeiras funções `insert` retornam um iterador que aponta para a posição na qual o novo elemento foi inserido no vetor.

Comentários

Como precondição, *first* e *last* não deverão ser iteradores no vetor ou o comportamento será indefinido. Uma operação de inserção pode ser cara; confira classe vector para ver uma discussão sobre o desempenho de `vector`.

Exemplo

C++

```
// vector_insert.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;
   vector <int>::iterator Iter;

   v1.push_back( 10 );
   v1.push_back( 20 );
   v1.push_back( 30 );
```

```
    cout << "v1 =" ;
    for ( Iter = v1.begin( ) ; Iter != v1.end( ) ; Iter++ )
       cout << " " << *Iter;
    cout << endl;

    v1.insert( v1.begin( ) + 1, 40 );
    cout << "v1 =";
    for ( Iter = v1.begin( ) ; Iter != v1.end( ) ; Iter++ )
       cout << " " << *Iter;
    cout << endl;
    v1.insert( v1.begin( ) + 2, 4, 50 );

    cout << "v1 =";
    for ( Iter = v1.begin( ) ; Iter != v1.end( ) ; Iter++ )
       cout << " " << *Iter;
    cout << endl;

    const auto v2 = v1;
    v1.insert( v1.begin( )+1, v2.begin( )+2, v2.begin( )+4 );
    cout << "v1 =";
    for (Iter = v1.begin( ); Iter != v1.end( ); Iter++ )
       cout << " " << *Iter;
    cout << endl;

// initialize a vector of vectors by moving v1
    vector < vector <int> > vv1;

    vv1.insert( vv1.begin(), move( v1 ) );
    if ( vv1.size( ) != 0 && vv1[0].size( ) != 0 )
       {
       cout << "vv1[0] =";
       for (Iter = vv1[0].begin( ); Iter != vv1[0].end( ); Iter++ )
          cout << " " << *Iter;
       cout << endl;
       }
}
```

Output

```
v1 = 10 20 30
v1 = 10 40 20 30
v1 = 10 40 50 50 50 50 20 30
v1 = 10 50 50 40 50 50 50 50 20 30
vv1[0] = 10 50 50 40 50 50 50 50 20 30
```

iterator

Um tipo que fornece um iterador de acesso aleatório que pode ler ou modificar qualquer elemento em um vetor.

C++

```
typedef implementation-defined iterator;
```

Comentários

Um tipo de **`iterator`** pode ser usado para modificar o valor de um elemento.

Exemplo

Confira o exemplo de begin.

`max_size`

Retorna o tamanho máximo do vetor.

C++

```
size_type max_size() const;
```

Retornar valor

O tamanho máximo possível do vetor.

Exemplo

C++

```
// vector_max_size.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;
   vector <int>::size_type i;

   i = v1.max_size( );
   cout << "The maximum possible length of the vector is " << i << "." <<
endl;
}
```

operator[]

Retorna uma referência para o elemento de vetor em uma posição especificada.

C++

```
reference operator[](size_type position);

const_reference operator[](size_type position) const;
```

Parâmetros

position
A posição do elemento de vetor.

Retornar valor

Se a posição especificada for maior ou igual ao tamanho do contêiner, o resultado será indefinido.

Comentários

Se o valor retornado de `operator[]` for atribuído a `const_reference`, o objeto vector não poderá ser modificado. Se o valor de retorno de `operator[]` for atribuído a uma referência, o objeto de vetor poderá ser modificado.

Quando compilado usando _ITERATOR_DEBUG_LEVEL definido como 1 ou 2, um erro de runtime ocorrerá se você tentar acessar um elemento fora dos limites do vetor. Para obter mais informações, confira Iteradores verificados.

Exemplo

C++

```
// vector_op_ref.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;
```

```
    v1.push_back( 10 );
    v1.push_back( 20 );

    int& i = v1[1];
    cout << "The second integer of v1 is " << i << endl;
}
```

operator=

Substitui os elementos do vetor por uma cópia de outro vetor.

C++

```
vector& operator=(const vector& right);

vector& operator=(vector&& right);
```

Parâmetros

right

O vector que está sendo copiado no `vector`.

Comentários

Após apagar os elementos existentes em um `vector`, `operator=` copiará ou moverá o conteúdo de *right* para `vector`.

Exemplo

C++

```
// vector_operator_as.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector<int> v1, v2, v3;
   vector<int>::iterator iter;

   v1.push_back(10);
   v1.push_back(20);
```

```
    v1.push_back(30);
    v1.push_back(40);
    v1.push_back(50);

    cout << "v1 = " ;
    for (iter = v1.begin(); iter != v1.end(); iter++)
       cout << *iter << " ";
    cout << endl;

    v2 = v1;
    cout << "v2 = ";
    for (iter = v2.begin(); iter != v2.end(); iter++)
       cout << *iter << " ";
    cout << endl;

// move v1 into v2
    v2.clear();
    v2 = move(v1);
    cout << "v2 = ";
    for (iter = v2.begin(); iter != v2.end(); iter++)
       cout << *iter << " ";
    cout << endl;
}
```

`pointer`

Um tipo que fornece um ponteiro para um elemento em um vetor.

C++

```
typedef typename Allocator::pointer pointer;
```

Comentários

Um tipo de **`pointer`** pode ser usado para modificar o valor de um elemento.

Exemplo

C++

```
// vector_pointer.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
```

```
    using namespace std;
    vector<int> v;
    v.push_back( 11 );
    v.push_back( 22 );

    vector<int>::pointer ptr = &v[0];
    cout << *ptr << endl;
    ptr++;
    cout << *ptr << endl;
    *ptr = 44;
    cout << *ptr << endl;
}
```

Output

```
11
22
44
```

pop_back

Exclui o elemento no final do vetor.

C++

```
void pop_back();
```

Comentários

Para obter um exemplo de código, consulte vector::push_back().

push_back

Adiciona um elemento ao final do vetor.

C++

```
void push_back(const T& value);

void push_back(T&& value);
```

Parâmetros

value

O valor a ser atribuído ao elemento adicionado ao final do vetor.

Exemplo

C++

```
// compile with: /EHsc /W4
#include <vector>
#include <iostream>

using namespace std;

template <typename T> void print_elem(const T& t) {
    cout << "(" << t << ") ";
}

template <typename T> void print_collection(const T& t) {
    cout << "  " << t.size() << " elements: ";

    for (const auto& p : t) {
        print_elem(p);
    }
    cout << endl;
}

int main()
{
    vector<int> v;
    for (int i = 0; i < 10; ++i) {
        v.push_back(10 + i);
    }

    cout << "vector data: " << endl;
    print_collection(v);

    // pop_back() until it's empty, printing the last element as we go
    while (v.begin() != v.end()) {
        cout << "v.back(): "; print_elem(v.back()); cout << endl;
        v.pop_back();
    }
}
```

rbegin

Retorna um iterador para o primeiro elemento em um vetor invertido.

C++

```
reverse_iterator rbegin();
const_reverse_iterator rbegin() const;
```

Retornar valor

Um iterador de acesso aleatório inverso que endereça o primeiro elemento em um vetor invertido ou que endereça qual foi o último elemento no vetor não invertido.

Comentários

Se o valor retornado de `rbegin` for atribuído a `const_reverse_iterator`, o objeto vector não poderá ser modificado. Se o valor retornado de `rbegin` for atribuído a `reverse_iterator`, o objeto vector poderá ser modificado.

Exemplo

C++

```
// vector_rbegin.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;
   vector <int>::iterator v1_Iter;
   vector <int>::reverse_iterator v1_rIter;

   v1.push_back( 1 );
   v1.push_back( 2 );

   v1_Iter = v1.begin( );
   cout << "The first element of vector is "
        << *v1_Iter << "." << endl;

   v1_rIter = v1.rbegin( );
   cout << "The first element of the reversed vector is "
        << *v1_rIter << "." << endl;
}
```

Output

```
The first element of vector is 1.
The first element of the reversed vector is 2.
```

reference

Um tipo que fornece uma referência a um elemento armazenado em um vetor.

C++

```
typedef typename Allocator::reference reference;
```

Exemplo

Consulte at para obter um exemplo de como usar **reference** na classe vector.

rend

Retorna um iterador reverso que ultrapassa o fim e aponta para o elemento após o último elemento do vetor invertido.

C++

```
const_reverse_iterator rend() const;
reverse_iterator rend();
```

Retornar valor

Um iterador reverso que ultrapassa o fim para o vetor invertido. Ele aponta para o elemento que segue o último elemento do vetor invertido, que é o mesmo que o elemento antes do primeiro elemento do vetor não invertido. Esse elemento é um espaço reservado e não deve ser desreferenciado. Use-o apenas para comparações.

Comentários

`rend` é usado com um vetor invertido, assim como end é usado com um vetor.

Se o valor retornado de `rend` for atribuído a `const_reverse_iterator`, o objeto vector não poderá ser modificado. Se o valor retornado de `rend` for atribuído a `reverse_iterator`, o objeto vector poderá ser modificado.

`rend` pode ser usado para testar se um iterador invertido alcançou o final de seu vetor.

O valor retornado por `rend` não deve ser desreferenciado. Use-o apenas para comparações.

Exemplo

C++

```
// vector_rend.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;
   vector <int>::reverse_iterator v1_rIter;

   v1.push_back( 1 );
   v1.push_back( 2 );

   for ( v1_rIter = v1.rbegin( ) ; v1_rIter != v1.rend( ) ; v1_rIter++ )
      cout << *v1_rIter << endl;
}
```

Output

```
2
1
```

reserve

Reserva um tamanho mínimo de armazenamento para um objeto vector alocando espaço se necessário.

C++

```
void reserve(size_type count);
```

Parâmetros

count

O tamanho mínimo de armazenamento a ser alocado para o vetor.

Exemplo

C++

```
// vector_reserve.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;
   //vector <int>::iterator Iter;

   v1.push_back( 1 );
   cout << "Current capacity of v1 = "
      << v1.capacity( ) << endl;
   v1.reserve( 20 );
   cout << "Current capacity of v1 = "
      << v1.capacity( ) << endl;
}
```

Output

```
Current capacity of v1 = 1
Current capacity of v1 = 20
```

`resize`

Especifica um novo tamanho para um vetor.

C++

```
void resize(size_type new_size);
void resize(size_type new_size, Type value);
```

Parâmetros

new_size

O novo tamanho do vetor.

value

O valor de inicialização de novos elementos adicionados ao vetor, caso o novo tamanho seja maior que o tamanho original. Se o valor for omitido, os novos objetos usarão o construtor padrão.

Comentários

Se o tamanho do contêiner for menor que o tamanho solicitado, *new_size*, `resize` adicionará elementos ao vetor até ele atingir o tamanho solicitado. Se o tamanho do contêiner for maior que o tamanho solicitado, `resize` excluirá os elementos mais próximos do final do contêiner até o contêiner chegar ao tamanho *new_size*. Se o tamanho atual do contêiner for igual ao tamanho solicitado, nenhuma ação será realizada.

size reflete o tamanho atual do vetor.

Exemplo

C++

```
// vectorsizing.cpp
// compile with: /EHsc /W4
// Illustrates vector::reserve, vector::max_size,
// vector::resize, vector::resize, and vector::capacity.
//
// Functions:
//
//    vector::max_size - Returns maximum number of elements vector could
//                       hold.
//
//    vector::capacity - Returns number of elements for which memory has
//                       been allocated.
//
//    vector::size - Returns number of elements in the vector.
//
//    vector::resize - Reallocates memory for vector, preserves its
//                     contents if new size is larger than existing size.
//
//    vector::reserve - Allocates elements for vector to ensure a minimum
//                      size, preserving its contents if the new size is
//                      larger than existing size.
//
//    vector::push_back - Appends (inserts) an element to the end of a
//                        vector, allocating memory for it if necessary.
//
////////////////////////////////////////////////////////////////////////

// The debugger cannot handle symbols more than 255 characters long.
```

```
// The C++ Standard Library often creates symbols longer than that.
// The warning can be disabled:
//#pragma warning(disable:4786)

#include <iostream>
#include <vector>
#include <string>

using namespace std;

template <typename C> void print(const string& s, const C& c) {
    cout << s;

    for (const auto& e : c) {
        cout << e << " ";
    }
    cout << endl;
}

void printvstats(const vector<int>& v) {
    cout << "   the vector's size is: " << v.size() << endl;
    cout << "   the vector's capacity is: " << v.capacity() << endl;
    cout << "   the vector's maximum size is: " << v.max_size() << endl;
}

int main()
{
    // declare a vector that begins with 0 elements.
    vector<int> v;

    // Show statistics about vector.
    cout << endl << "After declaring an empty vector:" << endl;
    printvstats(v);
    print("   the vector's contents: ", v);

    // Add one element to the end of the vector.
    v.push_back(-1);
    cout << endl << "After adding an element:" << endl;
    printvstats(v);
    print("   the vector's contents: ", v);

    for (int i = 1; i < 10; ++i) {
        v.push_back(i);
    }
    cout << endl << "After adding 10 elements:" << endl;
    printvstats(v);
    print("   the vector's contents: ", v);

    v.resize(6);
    cout << endl << "After resizing to 6 elements without an initialization
value:" << endl;
    printvstats(v);
    print("   the vector's contents: ", v);

    v.resize(9, 999);
```

```
    cout << endl << "After resizing to 9 elements with an initialization va-
lue of 999:" << endl;
    printvstats(v);
    print("   the vector's contents: ", v);

    v.resize(12);
    cout << endl << "After resizing to 12 elements without an initialization
value:" << endl;
    printvstats(v);
    print("   the vector's contents: ", v);

    // Ensure there's room for at least 1000 elements.
    v.reserve(1000);
    cout << endl << "After vector::reserve(1000):" << endl;
    printvstats(v);

    // Ensure there's room for at least 2000 elements.
    v.resize(2000);
    cout << endl << "After vector::resize(2000):" << endl;
    printvstats(v);
}
```

`reverse_iterator`

Um tipo que fornece um iterador de acesso aleatório que pode ler ou modificar qualquer elemento em um vetor invertido.

C++

```
typedef std::reverse_iterator<iterator> reverse_iterator;
```

Comentários

Um tipo `reverse_iterator` é usado para iterar no vetor em ordem inversa.

Exemplo

Confira o exemplo de rbegin.

`shrink_to_fit`

Descarta o excesso de capacidade.

C++

```
void shrink_to_fit();
```

Exemplo

C++

```
// vector_shrink_to_fit.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;
   //vector <int>::iterator Iter;

   v1.push_back( 1 );
   cout << "Current capacity of v1 = "
      << v1.capacity( ) << endl;
   v1.reserve( 20 );
   cout << "Current capacity of v1 = "
      << v1.capacity( ) << endl;
   v1.shrink_to_fit();
   cout << "Current capacity of v1 = "
      << v1.capacity( ) << endl;
}
```

Output

```
Current capacity of v1 = 1
Current capacity of v1 = 20
Current capacity of v1 = 1
```

`size`

Retorna o número de elementos no vetor.

C++

```
size_type size() const;
```

Retornar valor

O tamanho atual do vetor.

Exemplo

C++

```
// vector_size.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1;
   vector <int>::size_type i;

   v1.push_back( 1 );
   i = v1.size( );
   cout << "Vector length is " << i << "." << endl;

   v1.push_back( 2 );
   i = v1.size( );
   cout << "Vector length is now " << i << "." << endl;
}
```

Output

```
Vector length is 1.
Vector length is now 2.
```

`size_type`

Um tipo que conta o número de elementos em um vetor.

C++

```
typedef typename Allocator::size_type size_type;
```

Exemplo

Confira o exemplo de capacity.

`swap`

Troca os elementos de dois vetores.

C++

```
void swap(
    vector<Type, Allocator>& right);

friend void swap(
    vector<Type, Allocator>& left,
    vector<Type, Allocator>& right);
```

Parâmetros

right

Um vetor que fornece os elementos a serem trocados. Alternativamente, um vetor cujos elementos deverão ser trocados por aqueles do vetor *left*.

left

Um vetor cujos elementos deverão ser trocados por aqueles do vetor *right*.

Exemplo

C++

```
// vector_swap.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector <int> v1, v2;

   v1.push_back( 1 );
   v1.push_back( 2 );
   v1.push_back( 3 );

   v2.push_back( 10 );
   v2.push_back( 20 );

   cout << "The number of elements in v1 = " << v1.size( ) << endl;
   cout << "The number of elements in v2 = " << v2.size( ) << endl;
   cout << endl;

   v1.swap( v2 );

   cout << "The number of elements in v1 = " << v1.size( ) << endl;
```

```
    cout << "The number of elements in v2 = " << v2.size( ) << endl;
}
```

Output

```
The number of elements in v1 = 3
The number of elements in v2 = 2

The number of elements in v1 = 2
The number of elements in v2 = 3
```

value_type

Um tipo que representa o tipo de dados armazenado em um vetor.

C++

```
typedef typename Allocator::value_type value_type;
```

Comentários

`value_type` é um sinônimo do parâmetro de modelo `Type`.

Exemplo

C++

```
// vector_value_type.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main( )
{
   using namespace std;
   vector<int>::value_type AnInt;
   AnInt = 44;
   cout << AnInt << endl;
}
```

Output

```
44
```

vector

Constrói um vetor. Sobrecargas constroem um vetor de um tamanho específico ou com elementos de um valor específico. Ou, como uma cópia de todos ou parte de algum outro vetor. Algumas sobrecargas também permitem que você especifique o alocador a ser usado.

C++

```
vector();
explicit vector(const Allocator& allocator);
explicit vector(size_type count);
vector(size_type count, const Type& value);
vector(size_type count, const Type& value, const Allocator& allocator);

vector(const vector& source);
vector(vector&& source);
vector(initializer_list<Type> init_list, const Allocator& allocator);

template <class InputIterator>
vector(InputIterator first, InputIterator last);
template <class InputIterator>
vector(InputIterator first, InputIterator last, const Allocator& allocator);
```

Parâmetros

allocator

A classe de alocador a ser usada com esse objeto. get_allocator retorna a classe de alocador para o objeto.

count

O número de elementos no vetor construído.

value

O valor dos elementos no vetor construído.

source

O vetor do qual o vetor construído deve ser uma cópia.

first

A posição do primeiro elemento no intervalo de elementos a ser copiado.

last

A posição do primeiro elemento após o intervalo de elementos a ser copiado.

init_list

O `initializer_list` contendo os elementos a serem copiados.

Comentários

Todos os construtores armazenam um objeto allocator (*allocator*) e iniciam o vetor.

Os primeiros dois construtores especificam um vetor inicial vazio. O segundo construtor especifica explicitamente o tipo allocator (*allocator*) a ser usado.

O terceiro construtor especifica uma repetição de um número especificado (*count*) de elementos do valor padrão para a classe `Type`.

O quarto e o quinto construtor especificam uma repetição de elementos *count* de valor *value*.

O sexto construtor especifica uma cópia do vetor *source*.

O sétimo construtor move o vetor *source*.

O oitavo construtor usa initializer_list para especificar os elementos.

O nono e o décimo construtor copiam o intervalo [`first`, `last`] de um vetor.

Exemplo

C++

```
// vector_ctor.cpp
// compile with: /EHsc
#include <vector>
#include <iostream>

int main()
{
    using namespace std;
    vector <int>::iterator v1_Iter, v2_Iter, v3_Iter, v4_Iter, v5_Iter,
v6_Iter;

    // Create an empty vector v0
    vector <int> v0;

    // Create a vector v1 with 3 elements of default value 0
    vector <int> v1(3);

    // Create a vector v2 with 5 elements of value 2
    vector <int> v2(5, 2);
```

```
    // Create a vector v3 with 3 elements of value 1 and with the allocator
    // of vector v2
    vector <int> v3(3, 1, v2.get_allocator());

    // Create a copy, vector v4, of vector v2
    vector <int> v4(v2);

    // Create a new temporary vector for demonstrating copying ranges
    vector <int> v5(5);
    for (auto i : v5) {
        v5[i] = i;
    }

    // Create a vector v6 by copying the range v5[ first,  last)
    vector <int> v6(v5.begin() + 1, v5.begin() + 3);

    cout << "v1 =";
    for (auto& v : v1){
        cout << " " << v;
    }
    cout << endl;

    cout << "v2 =";
    for (auto& v : v2){
        cout << " " << v;
    }
    cout << endl;

    cout << "v3 =";
    for (auto& v : v3){
        cout << " " << v;
    }
    cout << endl;
    cout << "v4 =";
    for (auto& v : v4){
        cout << " " << v;
    }
    cout << endl;

    cout << "v5 =";
    for (auto& v : v5){
        cout << " " << v;
    }
    cout << endl;

    cout << "v6 =";
    for (auto& v : v6){
        cout << " " << v;
    }
    cout << endl;

    // Move vector v2 to vector v7
    vector <int> v7(move(v2));
    vector <int>::iterator v7_Iter;
```

```
    cout << "v7 =";
    for (auto& v : v7){
        cout << " " << v;
    }
    cout << endl;

    cout << "v8 =";
    vector<int> v8{ { 1, 2, 3, 4 } };
    for (auto& v : v8){
        cout << " " << v ;
    }
    cout << endl;
}
```

Output

```
v1 = 0 0 0
v2 = 2 2 2 2 2
v3 = 1 1 1
v4 = 2 2 2 2 2
v5 = 0 0 0 0 0
v6 = 0 0
v7 = 2 2 2 2 2
v8 = 1 2 3 4
```

Confira também

Acesso Thread-Safe na Biblioteca Padrão C++

Referência da biblioteca padrão C++