Universidade Tecnológica Federal do Paraná
UTFPR – Campus Curitiba

Orientação a Objetos
-
Programação em C++

Grupos de Slides 14: As classes *List* e *Vector* da *STL* (*Standard Template Library*).

Prof. Jean Marcelo SIMÃO

Classe *List*

Uma classe Predefinida na STL.

Exemplos adaptados do livro dos Deitels:

- Deitel H. M., Deitel, P. J. "C++ Como Programar". 3a Edição Bookman, 2001.

stdafx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once


#define WIN32_LEAN_AND_MEAN
// Exclude rarely-used stuff from Windows headers
#include <stdio.h>
#include <tchar.h>

// TODO: reference additional headers your program requires here

// Aqui adicionamos os includes necessários ao projeto
// Atualmente as 'implementações'de C++ usam o formato
// de include sem o uso do .h no final.
#include <iostream>

// Quando se usa include sem .h faz-se necessário
// explicitar que parte da biblioteca utilizaremos por meio
// de 'usings'

using std::cout;
using std::endl;


#include <list>

#include <algorithm>

using namespace std;
```

```
#include "stdafx.h"

// Cabeçalho para a função 'genérica' capaz de imprimir uma lista.
template <class T>
void imprimeLista ( const list<T> &listaRef );

// Função main pré-criada na criação de
// projeto Win32 console no Visual Studio (Express Edition).

int _tmain (int argc, _TCHAR* argv[])
{
  // Define uma lista a partir da classe 'list' do STL
  // A 'list' do STL é um template e aqui este template é
  // parametrizado com a classe 'int'
  list < int >  valores;

  // ************************************************
  // Adiciona elementos a lista a partir de seu começo.
  valores.push_front ( 1 );
  valores.push_front ( 2 );
  // Adiciona elementos a lista a partir de seu fim.
  valores.push_back ( 4 );
  valores.push_back ( 3 );

  // Imprime os elementos da lista via a função printList().
  cout << "Values contém: ";
  imprimeLista ( valores );
  cout << endl << endl;

  // ************************************************
  // Ordena os elementos da lista.
  valores.sort();

  // Imprime os elementos da lista via a função printList().
  cout << "Values após o 'sort' contém: ";
  imprimeLista ( valores );
  cout << endl << endl;
  return 0;
}
```

```
// Implementação da função genérica capaz de imprimir uma lista

template <class T>
void imprimeLista( const list<T> &listaRef )
{

   // Testa se a lista está vazia.
   if ( listaRef.empty() )
   {

        cout << " Lista está vazia. " << endl;

   }
   else
   {

        // Cria um iterador, isto é, um objeto que é capaz de receber
        // um 'ponteiro inicial' e um 'ponteiro final' de uma lista e,
        // a partir daí, imprimir todos os elementos da lista.

        // O interador exige um objeto de saída... Neste caso
        // usamos o cout que, na verdade, é um objeto
        // denido em <iostream>
        // Ainda, pode-se adicionar um texto entre
        // cada elemento impresso.

        ostream_iterator< T > Saida ( cout, " " );

        // Neste comando, o interador Saida receber o 'ponteiro inicial'
        // bem como o 'ponteiro final'.

        copy ( listaRef.begin(), listaRef.end(), Saida );

    }

}
```


C:\WINDOWS\system32\cmd.exe
Values contÚm: 2 1 4 3
Values ap¾s o 'sort' contÚm: 1 2 3 4
Press any key to continue . . .

stdafx.h

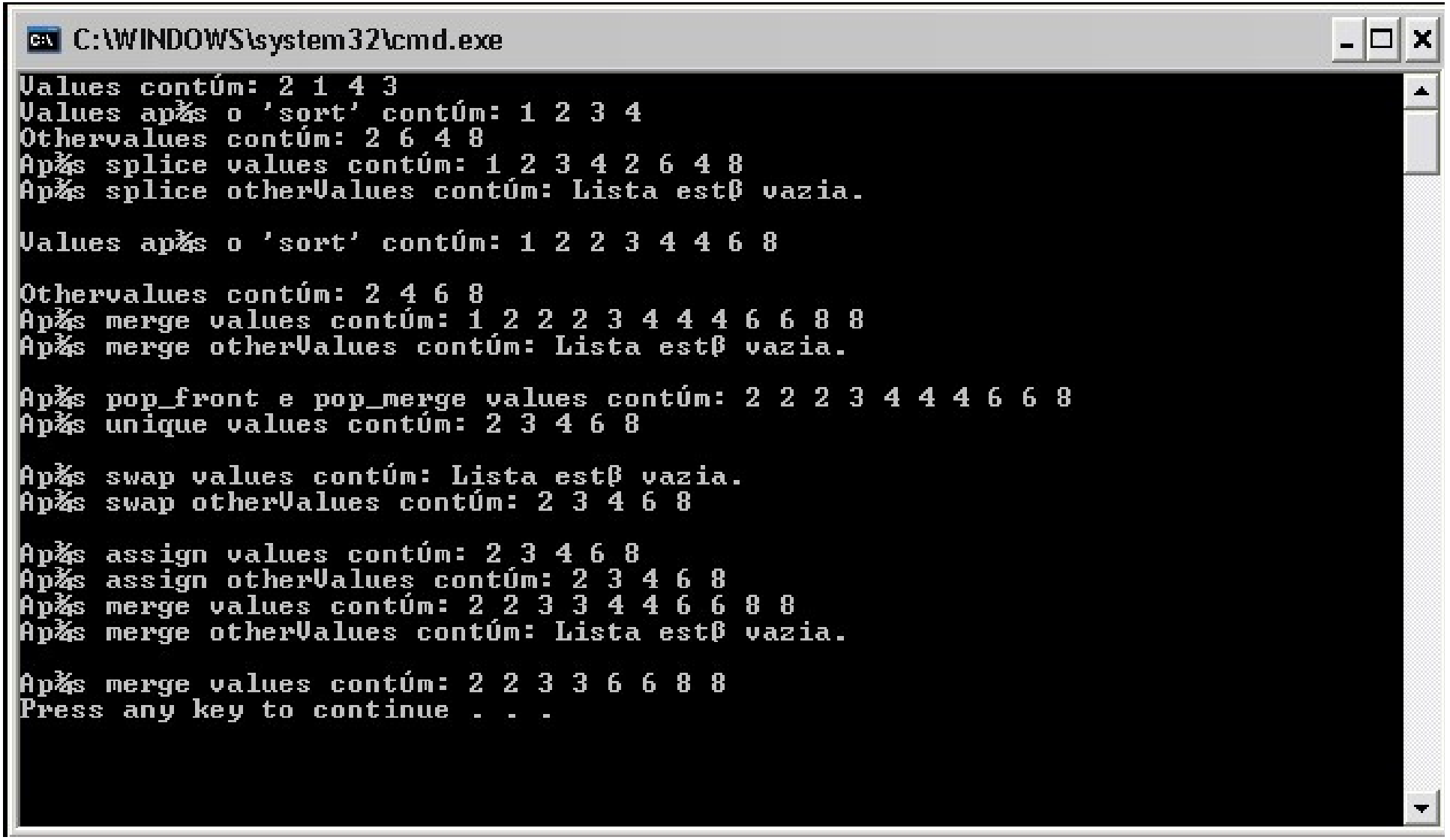
```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently

#pragma once

#define WIN32_LEAN_AND_MEAN
// Exclude rarely-used stuff from Windows headers
#include <stdio.h>
#include <tchar.h>

// TODO: reference additional headers your program requires here

// Aqui adicionamos os includes necessários ao projeto
// Atualmente várias 'implementações'de C++ usam o formato
// de include sem o uso do .h no final.
#include <iostream>

// Quando se usa include sem .h, faz- se necessário
// explicitar que parte da biblioteca utilizaremos por meio
// de 'usings'

using std::cout;
using std::endl;

#include <list>

#include <algorithm>

using namespace std;
```

stdafx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
// Lista2.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

// TODO: reference any additional headers you need in STDAFX.H
// and not in this file
```

```
// OBSERVAÇÃO, não deixe de olhar o stdafx.h
#include "stdafx.h"
// Cabeçalho para a função 'genérica' capaz de imprimir uma lista.
template <class T>
void imprimeLista( const std::list<T> &listRef );
// Função main pré-criada na criação de projeto Win32
//  console no Visual Studio (Express Edition).

int _tmain ( int argc, _TCHAR* argv[] )
{
   // Define uma constante;
   const int SIZE = 4;
   // Define um vetor, inicializando-o;
   int a [ SIZE ] = { 2, 6, 4, 8 };
   // Define duas lista a partir da classe 'list' do STL
   // A 'list' do STL é um template e aqui este template é
   // parametrizado com a classe 'int'.
   list<int> valores, outrosValores;

   // ************************************************
   // Adiciona elementos a lista a partir de seu começo.
   valores.push_front( 1 );
   valores.push_front( 2 );
   // Adiciona elementos a lista a partir de seu fim.
   valores.push_back ( 4 );
   valores.push_back ( 3 );
   // Imprime os elementos da lista via a função printList().
   cout << "Valores contém: "; imprimeLista ( valores ); cout << endl;
   // ************************************************
   // Ordena os elementos da lista.
   valores.sort();
   // Imprime os elementos da lista via a função printList().
   cout << "Values após o 'sort' contém: "; imprimeLista ( valores );
   cout << endl;

   // SEGUNDA VERSÃO A PARTIR DAQUI
   // ************************************************
   // Insere valores em outrosValores...
   outrosValores.insert ( outrosValores.begin(), a, a + SIZE);
   cout << "OutrosValores contém: "; imprimeLista ( outrosValores );
   cout << endl;
   valores.splice( valores.end(), outrosValores);
   cout << "Após splice valores contém: "; imprimeLista ( valores );
   cout << endl;
   cout << "Após splice outrosValores contém: "; imprimeLista ( outrosValores );
   cout << endl;
   // ************************************************
   valores.sort();
   cout << "Valores após o 'sort' contém: "; imprimeLista ( valores );
   cout << endl << endl;
   outrosValores.insert( outrosValores.begin(), a, a + SIZE );
   outrosValores.sort();
   cout << "OutrosValores contém: "; imprimeLista ( outrosValores );
   cout << endl;
   valores.merge( outrosValores );
   cout << "Após merge valores contém: "; imprimeLista ( valores );
   cout << endl;
   cout << "Após merge outrosValores contém: ";
   imprimeLista( outrosValores ); cout << endl;
   // ************************************************
   valores.pop_front();
   valores.pop_back(); // todos os conteineres sequenciais.
   cout << "Após pop_front e pop_baclk valores contém: ";
   imprimeLista( valores ); cout << endl;
   // ************************************************
   valores.unique();
   cout << "Após unique valores contém: ";
   imprimeLista( valores ); cout << endl << endl;
   // ************************************************
   valores.swap( outrosValores );
   cout << "Após swap valores  contém: "; imprimeLista ( valores );
   cout << "Após swap valores contém: ";
   imprimeLista( outrosValores ); cout << endl << endl;
   // ************************************************
   valores.assign( outrosValores.begin(), outrosValores.end() );
   cout << "Após assign valores contém: "; imprimeLista ( valores ); cout << endl;
   cout << "Após assign outrosValores contém: ";
   imprimeLista( outrosValores ); cout << endl;
   // ************************************************
   valores.merge( outrosValores );
   cout << "Após merge valores contém: "; imprimeLista ( valores ); cout << endl;
   cout << "Após merge outrosValores contém: ";
   imprimeLista( outrosValores ); cout << endl;
   // ************************************************
   valores.remove( 4 );
   cout << "Após merge valores contém: "; printList( valores ); cout << endl;
   return 0;
}
```

```
// Implementação da função genérica capaz de imprimir uma lista

template <class T>
void imprimeLista ( const std::list<T> &listRef )
{
   // Testa se a lista está vazia.
   if ( listRef.empty() )
   {
          cout << "Lista está vazia." << endl;
   }
   else
   {
          // Cria um iterado, isto é, um objeto que é capaz de receber
          // um 'ponteiro inicial' e um 'ponteiro final' de uma lista e,
          // a partir daí, imprimir todos os elementos da lista.

          // O interador exige um objeto de saída... Neste caso
          // usamos o cout que, na verdade, é um objeto denido em <stream>
          // Ainda, pode-se adicionar um texto entre cada elemento impresso.

          std::ostream_iterator< T > Saida ( cout, " " );
          // Este comando o interador Saida receber o 'ponteiro inicial'
          // bem como o 'ponteiro final'.
          std::copy( listRef.begin(), listRef.end(), Saida );
    }
}
```


C:\WINDOWS\system32\cmd.exe
Values contÚm: 2 1 4 3
Values ap%s o 'sort' contÚm: 1 2 3 4
Othervalues contÚm: 2 6 4 8
Ap%s splice values contÚm: 1 2 3 4 2 6 4 8
Ap%s splice otherValues contÚm: Lista estß vazia.
Values ap%s o 'sort' contÚm: 1 2 2 3 4 4 6 8
Othervalues contÚm: 2 4 6 8
Ap%s merge values contÚm: 1 2 2 2 3 4 4 4 6 6 8 8
Ap%s merge otherValues contÚm: Lista estß vazia.
Ap%s pop_front e pop_merge values contÚm: 2 2 2 3 4 4 4 6 6 8
Ap%s unique values contÚm: 2 3 4 6 8
Ap%s swap values contÚm: Lista estß vazia.
Ap%s swap otherValues contÚm: 2 3 4 6 8
Ap%s assign values contÚm: 2 3 4 6 8
Ap%s assign otherValues contÚm: 2 3 4 6 8
Ap%s merge values contÚm: 2 2 3 3 4 4 6 6 8 8
Ap%s merge otherValues contÚm: Lista estß vazia.
Ap%s merge values contÚm: 2 2 3 3 6 6 8 8
Press any key to continue . . .

Classe *Vector*

Uma classe Predefinida na STL.

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once


#define WIN32_LEAN_AND_MEAN
// Exclude rarely-used stuff from Windows headers

#include <stdio.h>
#include <tchar.h>



// TODO: reference additional headers your program requires here

#include <iostream>

using std::cout;
using std::cin;
using std::endl;

#include <vector>
using namespace std;
```

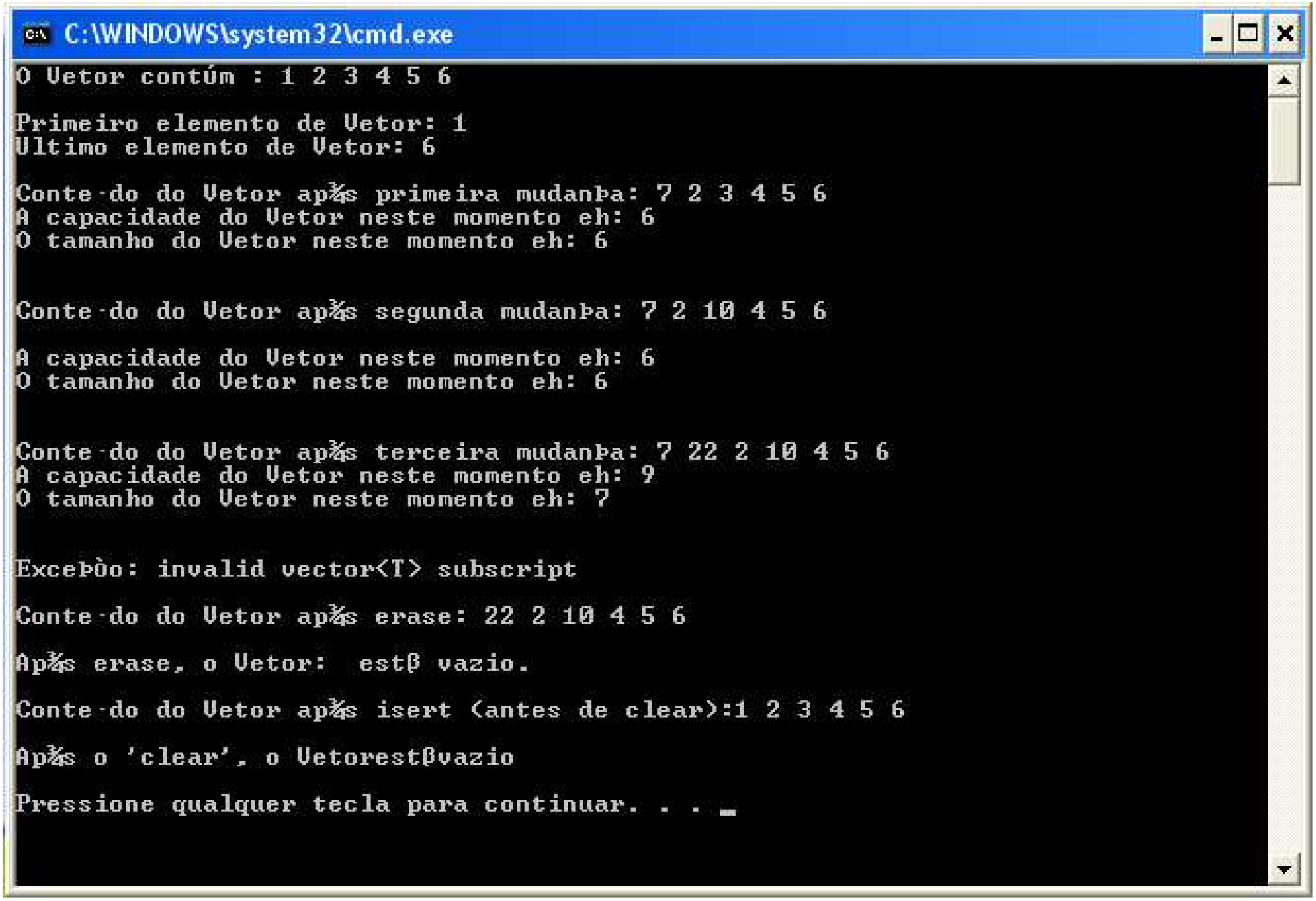
```
#include "stdafx.h“
// Detalhes na pág. 930 do Livro dos Deitels

template < class T >
void imprimeVetor ( vector< T > &Vetor);

int _tmain(int argc, _TCHAR* argv[])
{
      const int TAMANHO = 6;
      int a [ TAMANHO ] = { 1, 2, 3, 4, 5, 6};

      cout  << "Conteúdo do array a usando notação de ponteiro:“
       << endl;

      for ( int i = 0; i < TAMANHO; i++)
      {
            cout << a [ i ] << ' ';
      }

      for ( int *ptr = a; ptr != a + TAMANHO; ++ptr )
      {
            cout << *ptr << ' ';
      }
     cout << endl << endl;

     // ------------------------------------------------
      vector<int> Vetor;

      cout  << "O tamanho inicial de Vetor é: "
            << Vetor.size()
            << endl

            << "A capacidade inicial de Vetor é: "
            << Vetor.capacity()
            << endl;

      Vetor.push_back ( 2 );
      Vetor.push_back ( 3 );
      Vetor.push_back ( 4 );

      cout   << "O tamanho de Vetor é: "
             << Vetor.size()
             << endl

             <<  "A capacidade de Vetor é: "
             << Vetor.capacity()
             << endl << endl;

      cout << "Conteúdo do Vetor usando a notação de iterador: ";
      imprimeVetor ( Vetor );
      cout << endl << endl;

      cout << "Conteúdo do Vetor invertido: ";
      vector<int>::reverse_iterator Iterador;


      for ( Iterador = Vetor.rbegin(); Iterador != Vetor.rend(); ++Iterador )
      {
          cout << *Iterador << ' ';
      }
      cout << endl << endl;

      return 0;
}


template < class T >
void imprimeVetor ( vector< T > &Vetor )
{

    vector< T >::const_iterator Iterador2;

    for ( Iterador2 = Vetor.begin(); Iterador2 != Vetor.end(); Iterador2++ )
    {
         cout << *Iterador2 << ' ';
    }

}
```


C:\WINDOWS\system32\cmd.exe
Conte·do do array a usando notaÞÒo de ponteiro:
1 2 3 4 5 6
O tamanho inicial de Vetor Ú: 0
A capacidade inicial de Vetor Ú: 0
O tamanho de Vetor Ú: 3
A capacidade de Vetor Ú: 3
Conte·do do Vetor usando a notaÞÒo de iterador: 2 3 4
Conte·do do Vetor invertido: 4 3 2
Pressione qualquer tecla para continuar. . .

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once


#define WIN32_LEAN_AND_MEAN
// Exclude rarely-used stuff from Windows headers
#include <stdio.h>
#include <tchar.h>

// TODO: reference additional headers your program requires here

#include <iostream>

using std::cout;
using std::endl;

#include <vector>

#include <algorithm>

using namespace std;
```

```
#include "stdafx.h“

int _tmain ( int argc, _TCHAR* argv[] )
{
    const int TAMANHO = 6;
    int a [TAMANHO]= { 1, 2, 3, 4, 5, 6 };

    vector<int> Vetor ( a, a + TAMANHO );

    ostream_iterator<int> Saida( cout, " " );

    cout << "O Vetor contém : ";
    copy ( Vetor.begin(), Vetor.end(), Saida);

    cout << endl << endl;
    cout << "Primeiro elemento de Vetor: "
         << Vetor.front() << endl

         << "Ultimo elemento de Vetor: "
         << Vetor.back()
         << endl << endl;

    // Atribui 7 ao primeiro elemento
    Vetor [ 0 ] = 7;
    cout << "Conteúdo do Vetor após primeira mudança: “ << endl;
    copy ( Vetor.begin(), Vetor.end(), Saida);

    cout  << "A capacidade do Vetor neste momento eh: "
          << Vetor.capacity() << endl;
    cout  << "O tamanho do Vetor neste momento eh: "
          << Vetor.size() << endl << endl << endl;

    // Atribui 10 ao elemento na posição 2
    Vetor.at (2) = 10;

    cout  << "Conteúdo do Vetor após segunda mudança: ";
    copy ( Vetor.begin(), Vetor.end(), Saida);
    cout  << endl << endl;

    cout  << "A capacidade do Vetor neste momento eh: "
          << Vetor.capacity() << endl;
    cout  << "O tamanho do Vetor neste momento eh: "
          << Vetor.size() << endl << endl << endl;
```

```
    // insere 22 como segundo elemento.
    Vetor.insert( Vetor.begin() + 1, 22);
    cout << "Conteúdo do Vetor após terceira mudança: “  << endl;
    copy( Vetor.begin(), Vetor.end(), Saida);
    cout          << "A capacidade do Vetor neste momento eh: "
                  << Vetor.capacity() << endl;
     cout         << "O tamanho do Vetor neste momento eh: "
                  << Vetor.size() << endl << endl << endl;
     try
     {    // acessa elemento fora do intervalo válido
          Vetor.at(100) = 777;
     }
     catch ( std::out_of_range e )
     {
          cout << "Exceção: " << e.what() << endl << endl;
     }

     Vetor.erase ( Vetor.begin() );
     cout << "Conteúdo do Vetor após erase: “  << endl << endl;
     std::copy( Vetor.begin(), Vetor.end(), Saida);

     // cout       << "Conteúdo do Vetor após Segundo erase: ";
     // Vetor.erase( Vetor.begin()+2, Vetor.end() );
     // std::copy( Vetor.begin(), Vetor.end(), Saida);
     // cout       << endl << endl ;

     Vetor.erase( Vetor.begin(), Vetor.end() );
     cout << "Após erase, o Vetor: "
          << (Vetor.empty() ? " está " : " não está ")
           << "vazio." << endl << endl;

     Vetor.insert( Vetor.begin(), a, a + TAMANHO );
     cout  << "Conteúdo do Vetor após isert (antes de clear):";
     std::copy( Vetor.begin(), Vetor.end(), Saida);
     cout  << endl << endl;

     // clear chama erase para esvaziar uma coleção
     Vetor.clear();
     cout          << "Após o 'clear', o Vetor"
                   << ( Vetor.empty() ? "está" : "não está“ )
                   << "vazio" << endl << endl;
     return 0;
}
```


C:\WINDOWS\system32\cmd.exe
O Vetor contÚm : 1 2 3 4 5 6
Primeiro elemento de Vetor: 1
Ultimo elemento de Vetor: 6
Conte·do do Vetor ap¾s primeira mudanÞa: 7 2 3 4 5 6
A capacidade do Vetor neste momento eh: 6
O tamanho do Vetor neste momento eh: 6
Conte·do do Vetor ap¾s segunda mudanÞa: 7 2 10 4 5 6
A capacidade do Vetor neste momento eh: 6
O tamanho do Vetor neste momento eh: 6
Conte·do do Vetor ap¾s terceira mudanÞa: 7 22 2 10 4 5 6
A capacidade do Vetor neste momento eh: 9
O tamanho do Vetor neste momento eh: 7
ExceÞÒo: invalid vector<T> subscript
Conte·do do Vetor ap¾s erase: 22 2 10 4 5 6
Ap¾s erase, o Vetor: estß vazio.
Conte·do do Vetor ap¾s isert (antes de clear):1 2 3 4 5 6
Ap¾s o 'clear', o VetorestßvazioPressione qualquer tecla para continuar. . .