

# Linguagens de Programação

Prof. Miguel Elias Mitre Campista

<http://www.gta.ufrj.br/~miguel>

# Parte IV

Introdução à Programação em C++  
(Continuação)

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Relembrando da Última Aula...

- Tratamento de exceção
- Mais exemplos de programação orientada a objetos...

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## STL (Standard Template Library)

- Componentes baseados em templates poderosos
  - Contêineres: estrutura de dados template
  - Iteradores: como ponteiros, acessam elementos dos contêineres
  - Algoritmos: manipulação de dados, busca, ordenação etc.
- Programação orientada a objetos
  - Reuso!
    - Estruturas de dados podem também ser feitas através de construções com ponteiros → listas, árvores, filas etc.
    - Dificuldade de programação

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## STL (Standard Template Library)

- Uso de estruturas em alto nível
  - Uma vez definido o conteúdo das estruturas...
    - Acesso e armazenamento podem ser realizados através de classes da biblioteca STL
    - Programação genérica
- Enorme biblioteca de classes

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêineres

- Classificados em três tipos:
  - Contêineres sequenciais
    - Estrutura de dados linear (vetores, listas encadeadas)
    - Contêineres de primeira classe
  - Contêineres associativos
    - Não lineares, podem encontrar elementos rapidamente
    - Armazenamento por pares de chave-valor
    - Contêineres de primeira classe
  - Contêineres adaptados
    - Versões limitadas (adaptadas) de contêineres de primeira classe

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêineres

- Contêineres próximos
  - Similares aos contêineres de primeira classe, mas com funcionalidade reduzida
    - Ex.: Arrays baseados em ponteiros, strings, bitsets para armazenamento de flags etc.
- Contêineres têm algumas funções comuns
  - Subconjuntos de contêineres parecidos possuem funções comuns
    - *Convite à extensão*

## Classes STL Contêineres

- Contêineres sequenciais
  - vector
  - deque
  - list
- Contêineres associativos
  - set
  - multiset
  - map
  - multimap

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Classes STL Contêineres

- Contêineres adaptados
  - stack
  - queue
  - priority\_queue

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Funções Membro Comuns STL

- Funções membro para todos os contêineres
  - Construtor padrão, construtor de cópia, destrutor
  - empty
  - size
  - = < <= > >= == !=
  - swap
- Funções para contêineres de primeira classe
  - begin, end
  - rbegin, rend
  - erase, clear
  - max\_size

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Typedefs Comuns STL

- typedefs para contêineres de primeira classe
  - Simplificam a chamada a nomes de estruturas mais o tipo para o qual elas estão sendo utilizadas
    - Ex.: value\_type é um typedef do tipo de elemento armazenado em um contêiner
  - Ainda tem...
    - reference, const\_reference
    - pointer
    - iterator, const\_iterator
    - reverse\_iterator, const\_reverse\_iterator
    - difference\_type, size\_type

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Iteradores

- Funcionalidade similar a dos ponteiros
  - Apontam para elementos em contêiner de primeira classe
  - Certas operações com iteradores são as mesmas para todos os contêineres
    - \* desreferencia
    - ++ aponta para o próximo elemento
    - begin() retorna o iterador do primeiro elemento
    - end() retorna iterador do elemento depois do último

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Iteradores

- Funcionalidade similar a dos ponteiros
  - Objetos do tipo:
    - **iterator**
      - Se refere ao elemento que pode ser modificado
    - **const\_iterator**
      - Se refere ao elemento que não pode ser modificado
  - Podem ser usados para acessar seqüências (intervalos)
    - Em contêineres
    - Em seqüências de entrada: **istream\_iterator**
    - Em seqüências de saída: **ostream\_iterator**

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Uso dos Iteradores

- `std::istream_iterator<int> inputInt( cin )`
  - Pode ler entrada de `cin`
  - `*inputInt`
    - Desreferencia para ler o primeiro `int` de `cin`
  - `++inputInt`
    - Vai para o próximo `int` na seqüência de entrada

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Uso dos Iteradores

- `std::ostream_iterator<int> outputInt( cout )`
  - Pode retornar `ints` para o `cout`
  - `*outputInt = 7`
    - Retorna 7 para o `cout`
  - `++outputInt`
    - Avança iterador para retornar o próximo `int` da seqüência

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Primeiro Exemplo Usando STL em C++

```
/*
 * Aula 21 - Exemplo 1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iterator> // ostream_iterator e istream_iterator

using namespace std;

int main() {
    cout << "Enter two integers: ";

    // cria istream_iterator para leitura de valores int de cin
    istream_iterator< int > inputInt( cin );

    int number1 = *inputInt; // lê int da entrada padrão
    ++inputInt; // move iterador para o próximo valor de entrada
    int number2 = *inputInt; // lê int de entrada padrão

    // cria ostream_iterator para escrita de valores int em cout
    ostream_iterator< int > outputInt( cout );

    cout << "The sum is: ";
    *outputInt = number1 + number2; // retorna resultado para cout
    cout << endl;

    return 0;
}
```

## Primeiro Exemplo Usando STL em C++

```
/*
 * Aula 21 - Exemplo 1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iterator> // ostream_iterator e istream_iterator

using namespace std;
```

```
// cria ostream_iterator para escrita de valores int em cout
ostream_iterator< int > outputInt( cout );

cout << "The sum is: ";
*outputInt = number1 + number2; // retorna resultado para cout
cout << endl;

return 0;
}
```

## Primeiro Exemplo Usando STL em C++

```
/*
 * Aula 21 - Exemplo 1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iterator> // ostream_iterator e istream_iterator

using namespace std;
```

```
int main() {
    cout << "Enter two integers: ";

    // cria istream_iterator para leitura de valores int de cin
    istream_iterator<int> inputInt( cin );

    int number1 = *inputInt; // lê int da entrada padrão
    ++inputInt;
    int number2 = *inputInt; // lê int da entrada padrão

    // cria ostream_iterator para escrita de valores int em cout
    ostream_iterator<int> outputInt( cout );

    cout << "The sum is: ";
    *outputInt = number1 + number2; // retorna resultado para cout
    cout << endl;

    return 0;
}
```

## Primeiro Exemplo Usando STL em C++

```
/* Aula 21 - Exemplo 1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iterator> // ostream_iterator e istream_iterator

using namespace std;

int main() {
    cout << "Enter two integers: ";

    // Cria istream_iterator para leitura de valores int de cin
    istream_iterator<int> inputInt (cin);

    int number1 = *inputInt; // Lê int da entrada padrão
    ++inputInt;
    int number2 = *inputInt; // Lê int da entrada padrão

    E se comentar o ++inputInt? cout

    cout << "The sum is: ";
    *outputInt = number1 + number2; // retorna resultado para cout
    cout << endl;

    return 0;
}
```

## Primeiro Exemplo Usando STL em C++

```
/* Aula 21 - Exemplo 1
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iterator> // ostream_iterator e istream_iterator

using namespace std;

int main() {
    cout << "Enter two integers: ";

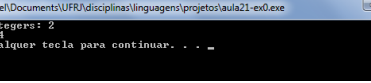
    // Cria istream_iterator para leitura de valores int de cin
    istream_iterator<int> inputInt (cin);

    int number1 = *inputInt; // Lê int da entrada padrão
    ++inputInt;
    int number2 = *inputInt; // Lê int da entrada padrão

    // Cria ostream_iterator para escrita de valores int em cout
    ostream_iterator<int> outputInt (cout);

    cout << "The sum is: ";
    *outputInt = number1 + number2; // retorna resultado para cout
    cout << endl;

    return 0;
}
```



## Categorias de Iteradores

- Entrada (*Input*)
  - Lê elementos do contêiner
  - Move somente na direção direta (do início para o fim)
- Saída (*Output*)
  - Escreve elementos no contêiner
  - Move somente na direção direta (do início para o fim)
- Encaminhamento (*Forward*)
  - Combina entrada com saída
  - Retém posição no contêiner (pode informar estado)

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Categorias de Iteradores

- Bidirecional (*Bidirectional*)
  - Como o de encaminhamento, mas pode também retroceder
- Acesso aleatório (*Random access*)
  - Como bidirecional, mas pode também saltar para qualquer elemento

**Iteradores de entrada são os mais simples enquanto os de acesso aleatório são os mais poderosos**

**Iteradores são usados somente em contêineres de primeira classe (sequenciais e associativos)**

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Tipos de Iteradores Suportados

- Contêineres sequenciais
  - `vector`: acesso aleatório
  - `deque`: acesso aleatório
  - `list`: bidirecional
- Contêineres associativos (todos bidirecionais)
  - `set`
  - `multiset`
  - `map`
  - `multimap`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Tipos de Iteradores Suportados

- Contêineres adaptados (não há suporte a iteradores)
  - `stack`
  - `queue`
  - `priority_queue`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Operações de Iteradores

- Todos
  - ++p, p++
  - p = p1
- Iteradores de entrada
  - \*p (pode ser desreferenciado como um *rvalue*)
  - p == p1, p != p1
- Iteradores de saída
  - \*p = t (pode ser desreferenciado como um *lvalue*)
- Iteradores de encaminhamento
  - Têm funcionalidade de iteradores de entrada e saída

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Operações de Iteradores

- Bidirecional
  - Têm funcionalidade de iteradores de encaminhamento
  - --p, p--
- Acesso aleatório
  - Têm funcionalidade de iteradores bidirecional
  - p + i, p += i
  - p - i, p -= i
  - p[i]
  - p < p1, p <= p1
  - p > p1, p >= p1

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Introdução aos Algoritmos

- STL tem algoritmos genéricos usados com contêineres
  - Opera indiretamente em elementos via iteradores
  - Frequentemente opera em sequência de elementos
    - Definido por pares de iteradores (primeiro e último elemento)
  - Algoritmos frequentemente retornam iteradores
    - `find()`
      - Retorna iterador para elemento ou `end()` se não encontrado
  - Algoritmos pré-construídos economizam tempo e esforço dos programadores

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêineres Sequenciais

- Três contêineres sequenciais
  - `vector` - baseado em arrays
  - `deque` - baseado em arrays
  - `list` - lista encadeada robusta

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Sequencial `vector`

- `vector`
  - `<vector>`
  - Estrutura de dados com alocação de memória sequencial
    - Acessa elementos com `[]`
  - Usado quando os dados devem ser ordenados e facilmente acessível
  - Mais eficientes se inserções forem feitas apenas no final

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Sequencial `vector`

- `vector`
  - Quando memória estiver esgotada...
    - Aloca maior área sequencial de memória
    - Copia ele mesmo lá
    - Desaloca memória antiga
  - Tem iteradores de acesso aleatório

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Sequencial vector

- Declaração
  - `std::vector<type> v;`
    - `type`: `int`, `float` etc.
- Iteradores
  - `std::vector<type>::const_iterator iterVar;`
    - `const_iterator` não pode modificar elementos
  - `std::vector<type>::reverse_iterator iterVar;`
    - Visita elementos na ordem reversa (fim para o início)
    - Usa `rbegin` para receber ponto de início na ordem reversa
    - Usa `rend` para receber ponto final na ordem reversa

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Sequencial vector

- Funções vector
  - `v.push_back(value)`
    - Adiciona elemento ao final (encontrado em todos os contêineres sequenciais)
  - `v.size()`
    - Número de elementos no vector
  - `v.capacity()`
    - Quanto o vector pode inserir antes de realocar memória
      - Realocação dobra tamanho atual

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Sequencial vector

- Funções vector
  - `vector<type> v(a, a + SIZE)`
    - Cria vector `v` com elementos do array `a` até o (não incluindo) `a + SIZE`
  - `v.insert(iterator, value)`
    - Insere `value` antes do posicionamento do `iterator`
  - `v.insert(iterator, array, array + SIZE)`
    - Insere elementos do array (até, mas não incluindo `array + SIZE`) antes do posicionamento do `iterator`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Sequencial vector

- Funções vector
  - `v.erase(iterator)`
    - Remove elemento do contêiner
  - `v.erase(iterator1, iterator2)`
    - Remove elementos começando do `iterator1` e até (não incluindo) `iterator2`
  - `v.clear()`
    - Apaga todo o contêiner

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Sequencial vector

- Operações de funções vector
  - `v.front()`, `v.back()`
    - Retorna uma referência para o primeiro e último elemento no contêiner, respectivamente
  - `v[elementNumber] = value;`
    - Atribui `value` a um elemento
  - `v.at(elementNumber) = value;`
    - Como acima, com checagem de intervalo
    - Exceção `out_of_bounds`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Sequencial vector

- `ostream_iterator`
  - `std::ostream_iterator< type > Name(outputStream, separator);`
    - `type`: retorna valores de um certo tipo
    - `outputStream`: localização do iterador de saída
    - `separator`: caractere separador da saída
- Ex.:
  - `std::ostream_iterator< int > output( cout, " " );`
  - `std::copy( iterator1, iterator2, output );`
    - Copia elementos do `iterator1` até (não incluindo) o `iterator2` na saída, um `ostream_iterator`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Segundo Exemplo Usando STL em C++

```
#include <iostream>
#include <vector>

using namespace std;

void print (vector<int> &vec) {
    for (int x : vec)
        cout << x << " ";
    cout << endl;
}

int main () {
    vector<int> v (1);
    v.push_back (1);
    v.push_back (2);

    cout << endl << v.size () << " " << v.capacity () << endl;
    print (v);

    vector<int> vv (5);
    vv.push_back (1);
    vv.push_back (2);
    vv.push_back (3);
    vv.push_back (4);
    vv.push_back (5);
    vv.push_back (6);
    vv.push_back (7);
    vv.push_back (8);
    vv.push_back (9);
    vv.push_back (10);
    vv.push_back (11);

    cout << endl << vv.size () << " " << vv.capacity () << endl;
    print (vv);

    return 0;
}
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Segundo Exemplo Usando STL em C++

```
#include <iostream>
#include <vector>

using namespace std;

void print (vector<int> &vec) {
    for (int x : vec)
        cout << x << " ";
    cout << endl;
}

int main () {
    vector<int> v (1);
    v.push_back (1);
    v.push_back (2);

    cout << endl << v.size () << " " << v.capacity () << endl;
    print (v);

    vector<int> vv (5);
    vv.push_back (1);
    vv.push_back (2);
    vv.push_back (3);
    vv.push_back (4);
    vv.push_back (5);
    vv.push_back (6);
    vv.push_back (7);
    vv.push_back (8);
    vv.push_back (9);
    vv.push_back (10);
    vv.push_back (11);

    cout << endl << vv.size () << " " << vv.capacity () << endl;
    print (vv);

    return 0;
}
```

Tipo auto assume o tipo relativo aos elementos inseridos no contêiner

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Segundo Exemplo Usando STL em C++

Capacidade é dobrada toda vez que a memória acaba e um novo elemento é inserido

Tipo auto requer uso do compilador para versão 11

```
int main () {
    vector<int> v (1);
    v.push_back (1);
    v.push_back (2);

    cout << endl << v.size () << " " << v.capacity () << endl;
    print (v);

    vector<int> vv (5);
    vv.push_back (1);
    vv.push_back (2);
    vv.push_back (3);
    vv.push_back (4);
    vv.push_back (5);
    vv.push_back (6);
    vv.push_back (7);
    vv.push_back (8);
    vv.push_back (9);
    vv.push_back (10);
    vv.push_back (11);

    cout << endl << vv.size () << " " << vv.capacity () << endl;
    print (vv);

    return 0;
}
```

POO para Redes de Computadores - COPPE-PEE/UFRJ

Prof. Miguel Campista

## Terceiro Exemplo Usando STL em C++

```
/*
 * Aula 21 - Exemplo 2
 * Arquivo Principais
 * Autor: Miguel Campista
 */
#include <iostream>
#include <vector> // definição de classe template vector

using namespace std;

// protótipo para função template printVector
template < class T >
void printVector (const vector< T > &integers2);

int main () {
    const int SIZE = 6;
    int array [ SIZE ] = { 1, 2, 3, 4, 5, 6 };

    vector< int > integers;

    cout << "The initial size of integers is: "
         << integers.size ()
         << "\nThe initial capacity of integers is: "
         << integers.capacity ();

    // função push_back será em todas as sequências
    integers.push_back ( 2 );
    integers.push_back ( 3 );
    integers.push_back ( 4 );

    cout << "\nThe size of integers is: " << integers.size ()
         << "\nThe capacity of integers is: "
         << integers.capacity ();
}
```

## Terceiro Exemplo Usando STL em C++

```
cout << "\n\nOutput array using pointer notation: ";
for ( int *ptr = array; ptr != array + SIZE; ++ptr )
    cout << *ptr << " ";

cout << "\n\nOutput vector using iterator notation: ";
printVector (integers);

cout << "\n\nReversed contents of vector integers: ";
vector< int > rvec (integers.rbegin(), integers.rend());

for ( reverse_iterator = integers.rbegin();
      reverse_iterator != integers.rend();
      ++reverse_iterator )
    cout << *reverse_iterator << " ";

cout << endl;

return 0;

// função template para exibir elementos do vector
template < class T >
void printVector (const vector< T > &integers2) {
    // prefixo class # necessário pq o compilador não pode definir
    // os membros T > const_iterator # uma especialização ou não
    class vector< T >::const_iterator const_iterator;

    for ( const_iterator = integers2.begin();
          const_iterator != integers2.end();
          ++const_iterator )
        cout << *const_iterator << " ";
}
```

## Segundo Exemplo Usando STL em C++

```
cout << "\n\nOutput array using pointer notation: ";
for ( int *ptr = array; ptr != array + SIZE; ++ptr )
    cout << *ptr << " ";

cout << "\n\nOutput vector using iterator notation: ";
printVector (integers);

cout << "\n\nReversed contents of vector integers: ";

// C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projetos\aula21-ex2.exe
The initial size of integers is: 0
The initial capacity of integers is: 0
The size of integers is: 3
The capacity of integers is: 4

Output array using pointer notation: 1 2 3 4 5 6
Reversed contents of vector integers: 4 3 2
Pressione qualquer tecla para continuar. . .

template < class T >
void printVector (const vector< T > &integers2) {
    // prefixo class # necessário pq o compilador não pode definir
    // os membros T > const_iterator # uma especialização ou não
    class vector< T >::const_iterator const_iterator;

    for ( const_iterator = integers2.begin();
          const_iterator != integers2.end();
          ++const_iterator )
        cout << *const_iterator << " ";
}
```

## Quarto Exemplo Usando STL em C++

```

/*
 * Aula 22 - Exemplo 3
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <vector> // definição da classe template vector
#include <algorithm> // algoritmo de copia
#include <iterator>
#include <stdexcept>

using namespace std;

int main() {
    const int SIZE = 6;
    int array[ SIZE ] = { 1, 2, 3, 4, 5, 6 };

    vector< int > integers( array, array + SIZE );
    ostream_iterator< int > output( cout, " " );

    cout << "Vector integers contains: ";
    copy( integers.begin(), integers.end(), output );

    cout << "\nFirst element of integers: " << integers.front()
        << "\nLast element of integers: " << integers.back();

    integers[ 0 ] = 7; // atribui ao primeiro elemento o 7
    integers.at( 2 ) = 10; // atribui ao elemento na posição 2 o 10

```

## Quarto Exemplo Usando STL em C++

```

// insere 22 como 2o elemento
integers.insert( integers.begin() + 1, 22 );

cout << "\nContents of vector integers after changes: ";
copy( integers.begin(), integers.end(), output );

// acessa elemento fora do intervalo
try {
    integers.at( 100 ) = 777;
}

// pega exceção out_of_range
catch ( out_of_range outOfRange ) {
    cout << "\nException: " << outOfRange.what();
}

// apaga primeiro elemento
integers.erase( integers.begin() );
cout << "\nVector integers after erasing first element: ";
copy( integers.begin(), integers.end(), output );

// apaga elementos restantes
integers.erase( integers.begin(), integers.end() );
cout << "\nAfter erasing all elements, vector integers = "
    << ( integers.empty() ? "Is" : "is not" ) << " empty";

```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Quarto Exemplo Usando STL em C++

```

// insere elementos do array
integers.insert( integers.begin(), array, array + SIZE );
cout << "\nContents of vector integers before clear: ";
copy( integers.begin(), integers.end(), output );

// esvazia inteiros; chamada clear apaga para esvaziar todos os elementos
integers.clear();
cout << "\nAfter clear, vector integers = "
    << ( integers.empty() ? "is" : "is not" ) << " empty";
cout << endl;

return 0;
}

```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Quarto Exemplo Usando STL em C++

```

// insere elementos do array
integers.insert( integers.begin(), array, array + SIZE );
cout << "\nContents of vector integers before clear: ";
copy( integers.begin(), integers.end(), output );

// esvazia inteiros; chamada clear apaga para esvaziar todos os elementos
integers.clear();
cout << "\nAfter clear, vector integers = "
    << ( integers.empty() ? "is" : "is not" ) << " empty";
cout << endl;

return 0;
}

```

```

C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projeto\aula21-ex3.exe
Vector integers contains: 1 2 3 4 5 6
First element of integers: 1
Last element of integers: 6
Contents of vector integers after changes: 7 22 2 10 4 5 6
Exception: vector::_M_range_check
Vector integers after erasing first element: 22 2 10 4 5 6
After erasing all elements, vector integers is empty
Contents of vector integers before clear: 1 2 3 4 5 6
After clear, vector integers is empty
Pressione qualquer tecla para continuar. . .

```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Sequencial list

- Contêiner list
  - Cabeçalho <list>
  - Inserção/remoção eficiente em qualquer lugar no contêiner
  - Lista duplamente encadeada (dois ponteiros por nó)
    - Um para elemento anterior outro para elemento posterior
  - Iteradores bidirecionais

Uso: `std::list< type > name;`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Sequencial list

- Funções list para o objeto l
  - l.sort()
    - Ordena em ordem crescente
  - l.splice( iterator, otherObject );
    - Insere valores de otherObject em l antes do iterador e remove os objetos de otherObject
  - l.merge( otherObject )
    - Remove otherObject e o insere em l, ordenado
  - l.unique()
    - Remove elementos duplicados

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista



## Contêiner Sequencial list

### Funções list

- l.swap(otherObject);
  - Troca conteúdo de l com o de otherObject
- l.assign(iterator1, iterator2);
  - Substitui conteúdo com elementos no intervalo dos iteradores
- l.remove(value);
  - Apaga todas as instâncias de value

## Quinto Exemplo Usando STL em C++

```

/*
 * Aula 21 - Exemplo 4
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <list> // definição da classe template list
#include <algorithm> // algoritmo de cópia
#include <iterator>

using namespace std;

// protótipo da função template printList
template < class T >
void printList( const std::list< T > &listRef );

int main() {
    const int SIZE = 4;
    int array[ SIZE ] = { 2, 6, 4, 8 };

    list< int > values;
    list< int > otherValues;

    // insere itens em values
    values.push_front( 1 );
    values.push_front( 2 );
    values.push_back( 4 );
    values.push_back( 3 );

    cout << "values contains: ";
    printList( values );
}

```

## Quinto Exemplo Usando STL em C++

```

values.sort(); // ordena values

cout << "\nvalues after sorting contains: ";
printList( values );

// insere elementos do array em otherValues
otherValues.insert( otherValues.begin(),
    array, array + SIZE );

cout << "\nAfter insert, otherValues contains: ";
printList( otherValues );

// remove elementos de otherValues e insere no final de values
values.splice( values.end(), otherValues );

cout << "\nAfter splice, values contains: ";
printList( values );

values.sort(); // ordena values

cout << "\nAfter sort, values contains: ";
printList( values );

// insere elementos do array em otherValues
otherValues.insert( otherValues.begin(),
    array, array + SIZE );
otherValues.sort();

cout << "\nAfter insert, otherValues contains: ";
printList( otherValues );
}

```

## Quinto Exemplo Usando STL em C++

```

// remove elementos de otherValues e insere ordenado em values
values.merge( otherValues );

cout << "\nAfter merge\n values contains: ";
printList( values );
cout << "\n otherValues contains: ";
printList( otherValues );

values.pop_front(); // remove elemento da frente
values.pop_back(); // remove elemento de trás

cout << "\nAfter pop_front and pop_back:"
    << "\n values contains: ";
printList( values );

values.unique(); // remove elementos duplicados

cout << "\nAfter unique, values contains: ";
printList( values );

// troca elementos de values e otherValues
values.swap( otherValues );

cout << "\nAfter swap\n values contains: ";
printList( values );
cout << "\n otherValues contains: ";
printList( otherValues );

// substitui conteúdo de values com elementos de otherValues
values.assign( otherValues.begin(), otherValues.end() );

cout << "\nAfter assign, values contains: ";
printList( values );
}

```

## Quinto Exemplo Usando STL em C++

```

// remove elementos de otherValues e insere ordenado em values
values.merge( otherValues );

cout << "\nAfter merge, values contains: ";
printList( values );

values.remove( 4 ); // remove todos os quatros

cout << "\nAfter remove( 4 ), values contains: ";
printList( values );

cout << endl;
return 0;
}

// definição da função template printList: usa
// ostream_iterator e algoritmo de cópia exibir elementos da lista
template < class T >
void printList( const std::list< T > &listRef ) {
    if ( listRef.empty() )
        cout << "List is empty";
    else
        ostream_iterator< T > output( cout, " " );
        copy( listRef.begin(), listRef.end(), output );
}
}

```

## Quinto Exemplo Usando STL em C++

```

// remove elementos de otherValues e insere ordenado em values
values.merge( otherValues );

```

```

C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projeto\aula21-ex4.exe
values contains: 2 1 4 3
values after sorting contains: 1 2 3 4
After insert, otherValues contains: 2 6 4 8
After splice, values contains: 1 2 3 4 2 6 4 8
After sort, values contains: 1 2 3 4 4 6 8
After unique, otherValues contains: 2 4 6 8
After merge:
values contains: 1 2 2 3 4 4 4 6 6 8 8
otherValues contains: List is empty
After pop_front and pop_back:
values contains: 2 2 3 4 4 4 6 6 8
After unique, values contains: 2 3 4 6 8
After swap:
values contains: List is empty
otherValues contains: 2 3 4 6 8
After assign, values contains: 2 3 4 6 8
After merge, values contains: 2 2 3 4 4 6 6 8 8
After remove( 4 ), values contains: 2 2 3 6 6 8 8
Pressione qualquer tecla para continuar. - -

```

## Contêiner Sequencial deque

- **deque** ("deek"): fila com final duplo (*double-ended queue*)
  - Header <deque>
  - Acesso indexado usando []
  - Inserção/remoção eficiente na frente e no final
- Mesmas operações básicas como **vector**
  - Entretanto, também possui como nas listas
    - **push\_front** (insere na frente do deque)
    - **pop\_front** (remove da frente)

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Sexto Exemplo Usando STL em C++

```
/* Aula 21 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <deque> // definição de classe template deque
#include <iterator>
#include <algorithm> // algoritmo copy

using namespace std;

int main() {
    deque<double> values;
    ostream_iterator<double> output( cout, " " );

    // insere valores como elementos
    values.push_front( 2.2 );
    values.push_front( 3.5 );
    values.push_back( 1.1 );

    cout << "values contains: ";

    // usa operador sub-escrito para obter valores dos elementos
    for ( int i = 0; i < values.size(); ++i )
        cout << values[ i ] << " ";

    values.pop_front(); // remove primeiro elemento

    cout << "\nAfter pop_front, values contains: ";
    copy( values.begin(), values.end(), output );
}
```

## Sexto Exemplo Usando STL em C++

```
// usa operador sub-escrito para modificar elemento na posição 1
values[ 1 ] = 5.4;

cout << "\nAfter values[ 1 ] = 5.4, values contains: ";
copy( values.begin(), values.end(), output );

cout << endl;

return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Sexto Exemplo Usando STL em C++

```
// usa operador sub-escrito para modificar elemento na posição 1
values[ 1 ] = 5.4;

cout << "\nAfter values[ 1 ] = 5.4, values contains: ";

values contains: 2.2 2.2 1.1
After pop_front, values contains: 2.2 1.1
After values[ 1 ] = 5.4, values contains: 2.2 5.4
Pressione qualquer tecla para continuar. . .
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêineres Associativos

- Acesso direto para armazenar/recuperar elementos
- Usa chaves
  - Realiza busca por chaves
- Quatro tipos: **multiset**, **set**, **multimap** e **map**
  - Ordenados por chaves

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêineres Associativos

- **multiset** e **set** manipulam conjunto de valores
  - Valores são as próprias chaves
- **multimap** e **map** manipulam valores associados com chaves
  - Possuem chaves e valores
- **multiset** e **multimap** permitem chaves duplicadas enquanto **set** e **map** não permitem

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Associativo multiset

- **multiset**
  - Header <set>
  - Armazenamento rápido, recuperação de chaves (sem valores)
  - Permite duplicatas
  - Iteradores bidirecionais

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Associativo multiset

- **multiset**
  - Ordenação de elementos
    - Feito por objeto com função comparadora (**operator<**)
      - Usado na criação do multiset
    - Para multiset inteiro
      - **less<int>** objeto com função comparadora
      - **multiset< int, std::less<int> > myObject;**
      - Elementos são armazenados em ordem crescente

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Associativo multiset

- Funções multiset
  - **ms.insert(value)**
    - **Insere valor no multiset**
  - **ms.count(value)**
    - **Retorna número de ocorrências do value**
  - **ms.find(value)**
    - **Retorna iterador para primeira ocorrência do value**
  - **ms.lower\_bound(value)**
    - **Retorna iterador para a primeira ocorrência do value**
  - **ms.upper\_bound(value)**
    - **Retorna iterador da primeira posição depois da última ocorrência do value**

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Associativo multiset

- Classe **pair**
  - Manipula pares de valores
  - Objeto pair contém first e second
    - **const\_iterators**
  - Para um objeto pair q
    - q = ms.equal\_range(value)**
    - **Ajusta first e second para lower\_bound e upper\_bound para um dado value**

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Sétimo Exemplo Usando STL em C++

```
/*
 * Aula 21 - Exemplo 6
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iterator>
#include <set> // definição de classes template multiset
// define apelido para tipo multiset usado neste programa
typedef std::multiset< int, std::less< int > > msz;
#include <algorithm> // algoritmo de cópia
using namespace std;

int main() {
    const int SIZE = 10;
    int ai[SIZE] = { 7, 22, 9, 1, 18, 30, 100, 22, 85, 13 };
    msz multiset; // msz é o apelido para "multiset inteiro"
    ostream_iterator< int > outout( cout, " " );

    cout << "There are currently " << multiset.count( 15 )
         << " values of 15 in the multiset!\n";

    multiset.insert( 15 ); // insere 15 no multiset
    multiset.insert( 15 ); // insere 15 no multiset

    cout << "After inserts, there are "
         << multiset.count( 15 )
         << " values of 15 in the multiset!\n";

    // iterador que não pode ser usado para mudar valores de elementos
    msz::const_iterator result;
```

## Sétimo Exemplo Usando STL em C++

```
// encontre (find) 15 em multiset; find retorna um iterador
result = multiset.find( 15 );

if ( result != multiset.end() ) // se iterador não estiver no fim
    cout << "Found value 15!\n"; // busca encontrou valor 15

// encontre 20 em multiset; find retorna iterador
result = multiset.find( 20 );

if ( result == multiset.end() ) // será true então
    cout << "Did not find value 20!\n"; // não encontrou 20

// insere elementos do array a no multiset
multiset.insert( a, a + SIZE );

cout << "\nAfter insert, multiset contains:\n";
copy( multiset.begin(), multiset.end(), outout );

// determina limiar inferior e superior dos 22 no multiset
cout << "\nLower bound of 22: "
     << * ( multiset.lower_bound( 22 ) );
cout << "\nUpper bound of 22: "
     << * ( multiset.upper_bound( 22 ) );

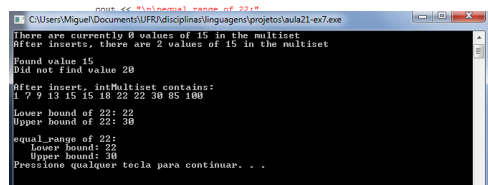
// p representa par de const_iterators
pair< msz::const_iterator, msz::const_iterator > p;
```

## Sétimo Exemplo Usando STL em C++

```
// usa equal_range para determinar limiar inferior e superior
// dos 22 no multiset
p = insert(multiset, equal_range( 22 ));
cout << "\nInequal_range of 22:"
    << "\n Lower bound: " << *( p.first )
    << "\n Upper bound: " << *( p.second );
cout << endl;
return 0;
}
```

## Sétimo Exemplo Usando STL em C++

```
// usa equal_range para determinar limiar inferior e superior
// dos 22 no multiset
p = insert(multiset, equal_range( 22 ));
```



## Contêiner Associativo set

### • set

- Header <set>
- Implementação idêntica de multiset
- Chaves únicas
  - Duplicatas ignoradas e não inseridas
- Suporta iteradores bidirecionais
  - Mas não acesso aleatório
- std::set< type, std::less<type> > name;

```
/*
 * Aula 21 - Exemplo 7
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iterator>
#include <set> // definição de classe template multiset

// define apelido para tipo multiset usado neste programa
typedef std::set< double, std::less< double > > double_set;

#include <algorithm> // algoritmo de cópia

using namespace std;

int main() {
    const int SIZE = 5;
    double a[ SIZE ] = { 2.1, 4.2, 9.5, 2.1, 3.7 };

    double_set doubleSet( a, a + SIZE );
    ostream_iterator< double > output( cout, " " );

    cout << "doubleSet contains: ";
    copy( doubleSet.begin(), doubleSet.end(), output );

    // p representa pair contendo const_iterator e bool
    pair< double_set::const_iterator, bool > p;

    // insere 13.8 em doubleSet; inserção retorna pair no qual
    // p.first representa posição de 13.8 no doubleSet e
    // p.second representa se 13.8 foi inserido
    p = doubleSet.insert( 13.8 ); // value que não está em set

    cout << "\n\n" << *( p.first )
        << ( p.second ? " was" : " was not" ) << " inserted";
}
```

## Oitavo Exemplo Usando STL em C++

```
cout << "\ndoubleSet contains: ";
copy( doubleSet.begin(), doubleSet.end(), output );

// insere 9.5 no doubleSet
p = doubleSet.insert( 9.5 ); // value já no set
cout << "\n\n" << *( p.first )
    << ( p.second ? " was" : " was not" ) << " inserted";

cout << "\ndoubleSet contains: ";
copy( doubleSet.begin(), doubleSet.end(), output );

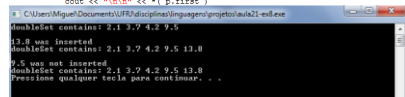
cout << endl;
return 0;
}
```

## Oitavo Exemplo Usando STL em C++

```
cout << "\ndoubleSet contains: ";
copy( doubleSet.begin(), doubleSet.end(), output );

// insere 9.5 no doubleSet
p = doubleSet.insert( 9.5 ); // value já no set

cout << "\n\n" << *( p.first )
```



## Contêiner Associativo multimap

- **multimap**
  - Header <map>
  - Armazenamento rápido e recuperação de chaves e valores associados
    - **Tem pares chave/valor**
  - Chaves duplicadas são permitidas (múltiplos valores para uma única chave)
    - **Relação um-para-muitos**
      - Ou seja, um estudante pode fazer múltiplos cursos
  - Insere objetos `pair` (com uma chave e valor)
  - Iteradores bidirecionais

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Associativo multimap

- Ex.

```
std::multimap< int, double, std::less< int > >
mmapObject;
```

  - Tipo de chave `int` e tipo de valor `double`
  - Ordenados em ordem crescente das chaves
    - Usa `typedef` para simplificar o código

```
typedef std::multimap<int, double, std::less<int>>
mmid;
```

```
mmid mmapObject;
```

```
mmapObject.insert(mmid::value_type(1, 3.4));
```

  - Insere chave 1 com valor 3.4
  - `mmid::value_type` cria um objeto `pair`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Nono Exemplo Usando STL em C++

```
/*
 * Aula 21 - Exemplo 8
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <string>
#include <map> // definição da classe template map

using namespace std;

// define apelido para tipo multimap usado neste programa
typedef multimap< int, double, less< int > > mmid;

int main() {
    mmid pairs;

    cout << "There are currently " << pairs.count( 15 )
         << " pairs with key 15 in the multimap\n";

    // insere dois objetos valor_tipo em pairs
    pairs.insert( mmid::value_type( 15, 2.7 ) );
    pairs.insert( mmid::value_type( 15, 99.3 ) );

    cout << "After inserts, there are "
         << pairs.count( 15 )
         << " pairs with key 15\n";
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Nono Exemplo Usando STL em C++

```
// insere cinco objetos valor_tipo em pairs
pairs.insert( mmid::value_type( 30, 111.11 ) );
pairs.insert( mmid::value_type( 10, 22.22 ) );
pairs.insert( mmid::value_type( 25, 33.333 ) );
pairs.insert( mmid::value_type( 20, 9.345 ) );
pairs.insert( mmid::value_type( 5, 77.84 ) );

cout << "Multimap pairs contains:\nKey\tValue\n";

// usa const_iterator para percorrer os elementos de pairs
for ( mmid::const_iterator iter = pairs.begin();
      iter != pairs.end(); ++iter )
    cout << iter->first << '\t'
         << iter->second << '\n';
cout << endl;

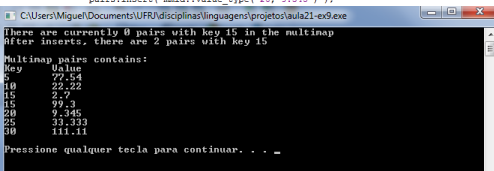
return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Nono Exemplo Usando STL em C++

```
// insere cinco objetos valor_tipo em pairs
pairs.insert( mmid::value_type( 30, 111.11 ) );
pairs.insert( mmid::value_type( 10, 22.22 ) );
pairs.insert( mmid::value_type( 25, 33.333 ) );
pairs.insert( mmid::value_type( 20, 9.345 ) );
pairs.insert( mmid::value_type( 5, 77.84 ) );
```



```
There are currently 0 pairs with key 15 in the multimap
After inserts, there are 2 pairs with key 15

Multimap pairs contains:
Key      Value
5        77.84
10       22.22
15       2.7
15       99.3
20       9.345
25       33.333
30       111.11

Pressione qualquer tecla para continuar. . .
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Associativo map

- **map**
  - Header <map>
  - Como `multimap`, mas somente pares chave/valor únicos
    - **Mapeamento um-para-um (duplicatas ignoradas)**
  - Usa `[]` para acessar valores
  - Ex.: para objeto `map m`
    - `m[30] = 4000.21;`
    - **Ajusta o valor da chave 30 para 4000.21**
- Declaração de tipo
  - `std::map< int, double, std::less< int > >;`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Décimo Exemplo Usando STL em C++

```

/*
 * Aula 21 - Exemplo 9
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <iterator>
#include <map> // definição de classe template map
using namespace std;

// Define apelido para tipo multimap usado neste programa
typedef map< int, double, less< int > > mtd;

int main() {
    mtd pairs;

    // insere oito objetos valor_tipo em pairs
    pairs.insert( mtd::value_type( 15, 2.7 ) );
    pairs.insert( mtd::value_type( 30, 111.11 ) );
    pairs.insert( mtd::value_type( 5, 1010.1 ) );
    pairs.insert( mtd::value_type( 10, 22.22 ) );
    pairs.insert( mtd::value_type( 20, 33.333 ) );
    pairs.insert( mtd::value_type( 5, 77.84 ) ); // dup ignored
    pairs.insert( mtd::value_type( 20, 9.945 ) );
    pairs.insert( mtd::value_type( 15, 99.3 ) ); // dup ignored

    cout << "pairs contains:\nKey\Value\n";

    // usa const_iterator para percorrer elementos de pairs
    for ( mtd::const_iterator iter = pairs.begin();
          iter != pairs.end(); ++iter )
        cout << iter->first << '\t'
              << iter->second << '\n';
}

```

## Décimo Exemplo Usando STL em C++

```

// usa operador sub-escrito para mudar valor da chave 25
pairs[ 25 ] = 9999.99;

// usa operador sub-escrito para inserir valo para chave 40
pairs[ 40 ] = 8765.432;

cout << "\nAfter subscript operations, pairs contains:"
     << "\nKey\Value\n";

for ( mtd::const_iterator ite2 = pairs.begin();
      ite2 != pairs.end(); ++ite2 )
    cout << ite2->first << '\t'
          << ite2->second << '\n';
cout << endl;

return 0;
}

```

Linguagens de Programação – DEL-Poli/UFRJ

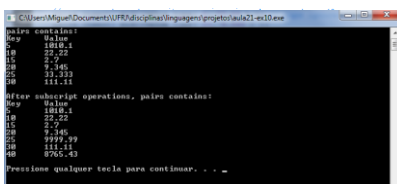
Prof. Miguel Campista

## Décimo Exemplo Usando STL em C++

```

// usa operador sub-escrito para mudar valor da chave 25
pairs[ 25 ] = 9999.99;

```



Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêiner Associativo map

### • map

- Se sub-escrito não estiver no map, um novo par chave/valor é criado

• Logo, uma sentença como  
`m [30] = 100;`  
 insere um par na estrutura

• Já uma chamada como  
`cout << m [1];`  
 insere o par chave = 1 e valor = 0 se m [1] não pertencer à estrutura

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Contêineres Adaptados

### • Contêineres adaptados

- stack, queue e priority\_queue
- Não são contêineres de primeira classe
  - Não suportam iteradores
  - Não provêm estrutura de dados atual
- Programador pode selecionar implementação
- Funções membro push e pop

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Adaptador stack

### • stack

- Header <stack>
- Inserções e remoções em uma extremidade
- Estrutura de dados last-in, first-out (LIFO)
- Pode usar vector, list, ou deque (padrão)
- Declarações

```

stack<type, vector<type> > myStack;
stack<type, list<type> > myOtherStack;
stack<type> anotherStack; // padrão deque

```

### • vector, list

- Implementação de stack (padrão deque)
- Não muda comportamento, apenas desempenho (deque e vector mais rápidos)

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Décimo Primeiro Exemplo Usando STL em C++

```

// Aula 21 - Exemplo 10
// Arquivo Principal
// Autor: Miguel Campista
//
#include <iostream>
#include <vector>
#include <list> // definição da classe template list
#include <vector> // definição da classe template vector
#include <stack> // definição adaptação stack

using namespace std;

// protótipo da função template popElements
template< class T >
void popElements( T &stackRef );

int main() {
    // stack com deque padrão
    stack< int > intDequeStack;

    // stack com vector
    stack< int, std::vector< int > > intVectorStack;

    // stack com list
    stack< int, std::list< int > > intListStack;

    // push os valores 0-9 na stack
    for ( int i = 0; i < 10; ++i ) {
        intDequeStack.push( i );
        intVectorStack.push( i );
        intListStack.push( i );
    }
}

```

## Décimo Primeiro Exemplo Usando STL em C++

```

// display and remove elements from each stack
cout << "Popping from intDequeStack: ";
popElements( intDequeStack );
cout << "\nPopping from intVectorStack: ";
popElements( intVectorStack );
cout << "\nPopping from intListStack: ";
popElements( intListStack );

cout << endl;

return 0;

// pop elementos do objeto stack para o qual stackRef se refere
template< class T >
void popElements( T &stackRef ) {
    while ( !stackRef.empty() ) {
        cout << stackRef.top() << " "; // vê elemento do topo
        stackRef.pop(); // remove elemento do topo
    }
}

```

Linguagens de Programação – DEL-Poli/UFRJ

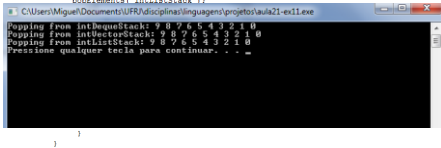
Prof. Miguel Campista

## Décimo Primeiro Exemplo Usando STL em C++

```

// display and remove elements from each stack
cout << "Popping from intDequeStack: ";
popElements( intDequeStack );
cout << "\nPopping from intVectorStack: ";
popElements( intVectorStack );
cout << "\nPopping from intListStack: ";
popElements( intListStack );
}

```



Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Adaptador queue

- **queue**
  - Header <queue>
  - Inserções no final, remoções na frente
  - Estrutura de dados first-in-first-out (FIFO)
  - Implementada com list ou deque (padrão)
    - **std::queue<double> values;**
- **Funções**
  - **push( elemento )**
    - Mesmo que **push\_back**, adicionar no final
  - **pop( element )**
    - Implementado com **pop\_front**, remove da frente
  - **empty() e size()**

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Décimo Segundo Exemplo Usando STL em C++

```

//
// Aula 21 - Exemplo 11
// Arquivo Principal
// Autor: Miguel Campista
//
#include <iostream>
#include <queue> // definição adaptação queue

using namespace std;

int main() {
    queue< double > values;

    // push elementos em valores da queue
    values.push( 3.2 );
    values.push( 9.0 );
    values.push( 5.4 );

    cout << "Popping from values: ";

    while ( !values.empty() ) {
        cout << values.front() << " "; // vê elementos da frente
        values.pop(); // remove elemento
    }
    cout << endl;

    return 0;
}

```

## Décimo Segundo Exemplo Usando STL em C++

```

//
// Aula 21 - Exemplo 11
// Arquivo Principal
// Autor: Miguel Campista
//
#include <iostream>
#include <queue> // definição adaptação queue

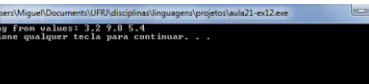
using namespace std;

cout << "Popping from values: ";

while ( !values.empty() ) {
    cout << values.front() << " "; // vê elementos da frente
    values.pop(); // remove elemento
}
cout << endl;

return 0;
}

```



## Adaptador priority\_queue

- `priority_queue`
  - Header `<queue>`
  - Inserções acontecem ordenadas, remoções da frente
  - Implementada com `vector` (padrão) ou `deque`
  - Elemento de **prioridade mais alta** é sempre removido primeiro
    - Algoritmo `heapsort` coloca elementos maiores na frente
    - `less<T>` padrão, programador especifica outro comparador
  - Funções
    - `push(value)`, `pop(value)`
    - `top()`
      - Vê elemento do topo
    - `size()` e `empty()`

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Adaptador priority\_queue

- `priority_queue`
  - Diferente dos anteriores, a classe template `priority_queue` possui três parâmetros:
    - **Tipo dos elementos**
    - **Contêiner**
    - **Classe de comparação:** Pode ser uma classe implementando uma função ou um ponteiro para uma função

```
template < class T, class Container =
vector<T>, class Compare = less<typename
Container::value_type> > class
priority_queue;
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Décimo Terceiro Exemplo Usando STL em C++

```
/*
 * Aula 21 - Exemplo 12
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <queue> // definição adaptação priority_queue

using namespace std;

int main() {
    priority_queue<double > priorities;

    // push elementos em valores da priorities
    priorities.push( 3.2 );
    priorities.push( 9.8 );
    priorities.push( 5.4 );

    cout << "Popping from priorities: ";

    while ( !priorities.empty() ) {
        cout << priorities.top() << " "; // vê elementos da frente
        priorities.pop(); // remove elemento
    }
    cout << endl;

    return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Décimo Terceiro Exemplo Usando STL em C++

```
/*
 * Aula 21 - Exemplo 12
 * Arquivo Principal
 * Autor: Miguel Campista
 */
#include <iostream>
#include <queue> // definição adaptação priority_queue

using namespace std;

int main() {
    priority_queue<double > priorities;

    // push elementos em valores da priorities
    priorities.push( 3.2 );
    priorities.push( 9.8 );
    priorities.push( 5.4 );

    cout << "Popping from priorities: 9.8 5.4 3.2
    Pressione qualquer tecla para continuar. . . .

    while ( !priorities.empty() ) {
        cout << priorities.top() << " "; // vê elementos da frente
        priorities.pop(); // remove elemento
    }
    cout << endl;

    return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Exemplo 1

- Escreva um programa que implemente a classe `Sistema` para armazenamento de cadastros em uma estrutura STL do tipo `map`. A classe `Sistema` ainda deve sobrecarregar o operador `<<` para exibir todos os dados relacionados com os cadastros armazenados. A chave deve ser o nome e o valor deve ser um ponteiro para um objeto da classe `Cadastro`. A classe `Cadastro` deve possuir dois atributos privados (`nome` e `cargo`) e métodos para obter tais atributos.

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista

## Exemplo 1

```
#include <iostream>
#include "sistemaex1ex14.h"

using namespace std;

int main() {
    Sistema s;

    // Inserção fora de ordem para posterior ordenação
    s.inserer ("Miguel", "Professor");
    s.inserer ("Luis", "Professor");

    cout << s;

    return 0;
}
```

Linguagens de Programação – DEL-Poli/UFRJ

Prof. Miguel Campista



## Exemplo 1

```
#include <iostream>
#include <string>
#include <map>
#include <omanip>
#include "CadastroCap21ex14.h"

#ifdef SISTEMA_H
#define SISTEMA_H
#endif

using namespace std;

typedef map<string, Cadastro *, less<string> > myMap;

class Sistema {
    friend ostream &operator<<(ostream &, Sistema &);
public:
    void insere (string, string);
private:
    myMap m;
};

#endif
```

## Exemplo 1

```
#include "sistemaCap21ex14.h"

ostream &operator<<(ostream &output, Sistema &s) {
    for (myMap::const_iterator it = (s.m).begin ();
         it != (s.m).end ();
         it++) {
        output << setw (8) << "Nome: "
                << (it->second->getNome() << "\n"
                << setw (8) << "Cargo: "
                << (it->second->getCargo() << endl << endl;
    }
    return output;
}

void Sistema::insere (string n, string c) {
    m.insert (myMap::value_type(n, new Cadastro (n, c)));
}
```

## Exemplo 1

```
#include <iostream>
#include <string>

#ifdef CADASTRO_H
#define CADASTRO_H
#endif

using namespace std;

class Cadastro {
public:
    Cadastro (string, string);
    string getNome () const;
    string getCargo () const;
private:
    string nome, cargo;
};

#endif
```

## Exemplo 1

```
#include "CadastroCap21ex14.h"

Cadastro::Cadastro (string n, string c) : nome (n), cargo (c) {}

string Cadastro::getNome () const {
    return nome;
}

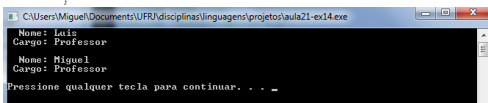
string Cadastro::getCargo () const {
    return cargo;
}
```

## Exemplo 1

```
#include "CadastroCap21ex14.h"

Cadastro::Cadastro (string n, string c) : nome (n), cargo (c) {}

string Cadastro::getNome () const {
    return nome;
}
```



```
C:\Users\Miguel\Documents\UFRJ\disciplinas\linguagens\projeto\laura21-ex14.exe
Nome: Luis
Cargo: Professor
Nome: Miguel
Cargo: Professor
Pressione qualquer tecla para continuar. . . _
```

## Leitura Recomendada

- Capítulos 21 do livro  
- Deitel, "C++ How to Program", 5th edition, Editora Prentice Hall, 2005