

# IMD0030

# LINGUAGEM DE PROGRAMAÇÃO I

*Aula 22 – Standard Template Library (STL)*  
*Containers e Iterators*

---

# Objetivos desta aula

- Introduzir a *Standard Template Library* (STL) e seus elementos
- Para isso, estudaremos:
  - *Containers*
  - Iteradores (*iterators*)
- Ao final da aula, espera-se que o aluno seja capaz de
  - ter uma visão geral acerca da STL e dos diferentes *containers* genéricos que ela oferece
  - fazer uso de iteradores sobre *containers*

---

# Standard Template Library (STL)

*“The Standard Template Library provides a set of well structured generic C++ components that work together in a seamless way.”*

Alexander Stepanov & Meng Lee, The Standard Template Library

---

# Standard Template Library (STL)

- A *Standard Template Library* (STL) é um subconjunto da biblioteca padrão da linguagem de programação C++ que define **containers genéricos e funções para manipula-los**
  - Os *containers* genéricos implementam essencialmente **tipos abstratos de dados** (TADs) utilizando *templates*, permitindo utilizar **qualquer tipo de dado/objeto**
- Principais vantagens do uso da STL
  - Redução do tempo de desenvolvimento, principalmente devido ao **reuso dos *containers* existentes**
    - Por exemplo, não é necessário implementar uma pilha: a própria STL provê uma pilha genérica e funções para inserir, remover, contabilizar elementos, etc.
  - **Facilidade de uso**
  - **Gerenciamento automático de memória**
    - Não é mais necessário preocupar-se com alocação dinâmica e liberação explícita de memória

# *Containers*

Classes que implementam estruturas de dados que servem para armazenar dados/objetos de qualquer tipo

---

# Containers

- Cada *container* implementa **regras** específicas que determinam como ele se comporta
  - Uma **lista** (*list*) permite inserir e remover elementos em qualquer posição
  - Uma **fila** (*queue*) permite apenas inserir elementos no seu fim e remover elementos de seu início
  - Um **deque** (*deque*) permite inserir e remover elementos tanto no início quanto no final
- Cada classe de *container* contém um conjunto de métodos que são utilizados para manipular os elementos que são nele armazenados
  - Principais métodos: **inserção, remoção, acesso, contagem, esvaziamento**
- *Containers* da STL são declarados em cabeçalho específicos, dentro do *namespace* **std**

# Containers

Containers	Descrição
<b>vector</b>	Vetor com número variável de elementos, que são armazenados sequencialmente em memória e podem ser acessados através da sua posição no vetor
<b>list</b>	Lista duplamente encadeada cujos elementos podem ser inseridos ou removidos em qualquer posição e pode ser percorrida tanto do início para o final quanto vice-versa
<b>queue</b>	TAD que implementa uma política FIFO ( <i>first-in, first-out</i> ): o primeiro elemento a ser inserido é o primeiro a ser removido
<b>deque</b>	TAD em que é possível inserir e remover elementos tanto no início quanto no final
<b>stack</b>	TAD que implementa uma política LIFO ( <i>last-in, first-out</i> ): o último elemento a ser inserido é o primeiro a ser removido
<b>set</b>	Conjunto ordenado de elementos que pode ou não conter duplicatas
<b>map</b>	<i>Container</i> que associa chaves a valores de forma ordenada: as chaves são utilizadas para recuperar os valores armazenados

---

# Containers

Novos *containers* introduzidos no C++11

Containers	Descrição
<b>array</b>	Vetor com número fixo de elementos, que são armazenados sequencialmente em memória e podem ser acessados através da sua posição no vetor
<b>forward_list</b>	Lista simplesmente encadeada que só pode ser percorrida do início para o final
<b>unordered_set</b>	Conjunto não-ordenado de elementos que pode ou não conter duplicatas
<b>unordered_map</b>	<i>Container</i> que associa chaves a valores de forma não-ordenada: as chaves são utilizadas para recuperar os valores armazenados



---

# Que *container* usar?

A escolha por um determinado *container* deve levar em consideração

- **como o *container* funciona**
  - Uma `queue` só permite inserir elementos no final e remover elementos do início
  - Um `set` não permite inserir elementos repetidos
  - Os elementos de uma `list` não podem ser acessados diretamente com o operador `[]`
- **a eficiência (complexidade computacional) das operações que podem ser realizadas**
  - O acesso a um elemento ou a inserção e remoção de elementos no fim de um `vector` são feitos em tempo constante ( $O(1)$ ), porém a inserção e remoção de elementos em posições intermediárias são feitas em tempo linear ( $O(n)$ )
  - A busca e o acesso a elementos em uma `list` são feitos em tempo linear ( $O(n)$ ), porém a inserção e remoção de elementos é feita em tempo constante ( $O(1)$ )

---

# Vector

- Disponível através da inclusão da biblioteca `<vector>`
  - Referência: <http://www.cplusplus.com/reference/vector/vector/>
- Como trata-se de um *template*, é necessário indicar o tipo de dados/objetos que o vetor irá armazenar
- Construtores disponíveis para a instanciação de um vetor
  - Construtor padrão: cria um vetor vazio
  - Construtor parametrizado: cria um vetor com um determinado número de elementos
  - Construtor cópia: cria um vetor a partir de outro já existente

---

# Vector

Principais métodos:

Método	Descrição
<b>push_back</b>	Insere um elemento no final do vetor
<b>pop_back</b>	Remove o último elemento do vetor
<b>size</b>	Retorna o número de elementos do vetor
<b>empty</b>	Verifica se o vetor está vazio
<b>operator[]</b>	Implementa a sobrecarga do operador [ ] para acessar um elemento em uma determinada posição
<b>operator=</b>	Implementa a sobrecarga do operador = para atribuir os elementos de um vetor a outro
<b>clear</b>	Remove todos os elementos do vetor

Lista completa disponível em <http://www.cplusplus.com/reference/vector/vector/>

---

# Vector

Complexidade computacional das operações sobre um vetor

Operação	Complexidade computacional (pior caso)
Acesso a elementos	constante – $O(1)$
Inserção e remoção de elementos no final	constante – $O(1)$
Inserção e remoção de elementos no meio	linear – $O(n)$
Busca por elementos	linear – $O(n)$

---

# Vector

## Exemplo

```
#include <iostream>
using std::cout;

#include <string>
using std::string;

#include <vector>
using std::vector;

int main() {
    vector<int> v;
    for (int i = 1; i <= 10; i++) v.push_back(i); // Vetor de inteiros vazio
                                                // Insercao de elementos no vetor

    for (int i = 0; i < (int)v.size(); i++) { // Iteracao sobre os elementos do vetor
        std::cout << v[i] << " "; // Acesso ao elemento indicado
    }

    return 0;
}
```

---

# List

- Disponível através da inclusão da biblioteca `<list>`
  - Referência: <http://www.cplusplus.com/reference/list/list/>
- Como trata-se de um *template*, é necessário indicar o tipo de dados/objetos que a lista irá armazenar
- Construtores disponíveis para a instanciação de uma lista
  - Construtor padrão: cria uma lista vazia
  - Construtor parametrizado: cria uma lista com um determinado número de elementos
  - Construtor cópia: cria uma lista a partir de outra já existente

---

# List

Principais métodos:

Método	Descrição
<b>push_front</b>	Insere um elemento no início da lista
<b>push_back</b>	Insere um elemento no final da lista
<b>pop_front</b>	Remove o primeiro elemento da lista
<b>pop_back</b>	Remove o último elemento da lista
<b>insert</b>	Insere um elemento em uma determinada posição da lista, utilizando um iterador
<b>erase</b>	Remove um elemento de uma determinada posição da lista, utilizando um iterador
<b>empty</b>	Verifica se a lista está vazia
<b>size</b>	Retorna o número de elementos da lista

Lista completa disponível em <http://www.cplusplus.com/reference/list/list/>

---

# List

Complexidade computacional das operações sobre uma lista

Operação	Complexidade computacional (pior caso)
Inserção e remoção de elementos no início ou do final	constante – $O(1)$
Inserção e remoção de elementos no meio	constante – $O(1)$
Busca por elementos	linear – $O(n)$



---

# List

## Exemplo

```
#include <iostream>
using std::cout;

#include <list>
using std::list;

int main() {
    list<int> l; // Criacao de uma lista vazia de inteiros
    l.push_front(1); // Insercao de elemento no inicio da lista
    l.push_back(2); // Insercao de elemento no final da lista

    list<int>::iterator it = l.begin(); // Iterador aponta para o inicio da lista
    l.insert(it, 0); // Insercao de elemento na posicao do iterador

    for (it = l.begin(); it != l.end(); ++it) // Iteracao sobre a lista usando o iterador
        std::cout << *it << " "; // Impressao do elemento apontado por iterador

    return 0;
}
```

---

---

# Stack

- Disponível através da inclusão da biblioteca `<stack>`
  - Referência: <http://www.cplusplus.com/reference/stack/stack/>
- Na STL, uma pilha é um **adaptador** para os *containers* sequenciais vetor, deque e lista, ou seja, pode ser implementada a partir desses *containers*
  - Por padrão, uma pilha é implementada a partir de um deque
- Como trata-se de um *template*, é necessário indicar o tipo de dados/objetos que a pilha irá armazenar
- **Não é possível** usar iteradores para percorrer uma pilha: apenas o elemento que está no topo da pilha pode ser acessado

---

# Stack

Principais métodos:

Método	Descrição
<b>push</b>	Insere um elemento no topo da lista
<b>pop</b>	Remove o elemento que está no topo da lista
<b>top</b>	Retorna o elemento que está no topo da lista
<b>size</b>	Retorna o número de elementos presentes na pilha
<b>empty</b>	Verifica se a pilha está vazia

Lista completa disponível em <http://www.cplusplus.com/reference/stack/stack/>

---

# Stack

## Exemplo

```
#include <iostream>
using std::cout;

#include <stack>
using std::stack;

int main() {
    stack<int> s;                                     // Criacao de uma pilha vazia de inteiros
    for (int i = 1; i <= 5; i++) {                    // Insercao de elementos na pilha
        s.push(i);
    }

    cout << "Elementos da pilha: ";
    while (!s.empty()) {
        cout << s.top() << " ";                      // Impressao do elemento que esta no topo da pilha
        s.pop();                                       // Remocao do elemento que esta no topo da pilha
    }

    return 0;
}
```

---

# Queue

- Disponível através da inclusão da biblioteca `<queue>`
  - Referência: <http://www.cplusplus.com/reference/queue/queue/>
- Na STL, uma fila é um **adaptador** para os *containers* sequenciais deque e lista, ou seja, pode ser implementada a partir desses *containers*
  - Por padrão, uma fila é implementada a partir de um deque
- Como trata-se de um *template*, é necessário indicar o tipo de dados/objetos que a pilha irá armazenar
- **Não é possível** usar iteradores para percorrer uma fila: apenas o primeiro e o último elemento da fila podem ser acessados

---

# Queue

Principais métodos:

Método	Descrição
<b>push</b>	Insere um elemento no fim da fila
<b>pop</b>	Remove o primeiro elemento da fila
<b>front</b>	Retorna o primeiro elemento da fila
<b>back</b>	Retorna o último elemento da fila
<b>size</b>	Retorna o número de elementos na fila
<b>empty</b>	Verifica se a fila está vazia

Lista completa disponível em <http://www.cplusplus.com/reference/queue/queue/>

---

# Queue

Complexidade computacional das operações sobre uma fila

Operação	Complexidade computacional (pior caso)
Acesso ao primeiro elemento	constante – $O(1)$
Inserção de elementos no final	constante – $O(1)$
Remoção de elementos do início	constante – $O(1)$
Busca por elementos	linear – $O(n)$

---

# Queue

## Exemplo

```
#include <iostream>
using std::cout;

#include <string>
using std::string;

#include <queue>
using std::queue;

int main() {
    queue<string> q;                                // Criacao de uma fila vazia de strings
    q.push("Roberto");                             // Insercao de elementos na fila
    q.push("Antonio");
    cout << "Elementos da fila: ";
    while (!q.empty()) {
        cout << q.front() << " ";                // Impressao do primeiro elemento da fila
        q.pop();                                   // Remocao do primeiro elemento da fila
    }

    return 0;
}
```



---

# Deque

- Disponível através da inclusão da biblioteca `<deque>`
  - Referência: <http://www.cplusplus.com/reference/deque/deque/>
- Como trata-se de um *template*, é necessário indicar o tipo de dados/objetos que o deque irá armazenar
- Construtores disponíveis para a instanciação de um deque
  - Construtor default: cria um deque vazio
  - Construtor parametrizado: cria um deque com um determinado número de elementos
  - Construtor cópia: cria um deque a partir de outro já existente

---

# Deque

Principais métodos:

Método	Descrição
<b>push_back</b>	Insere elemento no final do deque
<b>push_front</b>	Insere elemento no início do deque
<b>pop_back</b>	Remove o último elemento do deque
<b>pop_front</b>	Remove o primeiro elemento do deque
<b>size</b>	Retorna o número de elementos presentes no deque
<b>empty</b>	Verifica se o deque está vazio
<b>clear</b>	Esvazia o deque

Lista completa disponível em <http://www.cplusplus.com/reference/deque/deque/>

---

# Deque

Complexidade computacional das operações sobre um deque

Operação	Complexidade computacional (pior caso)
Acesso ao primeiro e ao último elemento	constante – $O(1)$
Inserção de elementos no início ou no final	constante – $O(1)$
Remoção de elementos no início ou no final	constante – $O(1)$
Busca por elementos	linear – $O(n)$
Inserção de elementos no meio	linear – $O(n)$
Remoção de elementos no meio	linear – $O(n)$

---

# Deque

## Exemplo

```
#include <iostream>
using std::cout;

#include <deque>
using std::deque;

int main() {
    deque<int> d; // Criacao de uma fila vazia de strings
    for (int i = 5; i >= 0; i--) d.push_front(i); // Insercao de elementos no inicio (0-5)
    for (int i = 6; i <= 11; i++) d.push_back(i); // Insercao de elementos no final (6-11)

    d.pop_back(); // Remocao do ultimo elemento do deque
    d.pop_front(); // Remocao do primeiro elemento do deque

    deque<int>::iterator it;
    for (it = d.begin(); it != d.end(); ++it) { // Iterador sobre o deque
        cout << *it << " "; // Impressao dos elementos do deque
    }

    return 0;
}
```

---

---

# Set

- Disponível através da inclusão da biblioteca `<set>`
  - Referência: <http://www.cplusplus.com/reference/set/set/>
- Como trata-se de um *template*, é necessário indicar o tipo de dados/objetos que o conjunto irá armazenar
- Variações:
  - `set`: não permite armazenar elementos repetidos
  - `multiset`: permite armazenar elementos repetidos

---

# Set

Principais métodos:

Método	Descrição
<b>insert</b>	Insere um elemento no conjunto
<b>erase</b>	Remove um elemento do conjunto
<b>find</b>	Busca por um elemento no conjunto, retornando um iterador
<b>count</b>	Conta o número de ocorrências de um determinado elemento no conjunto
<b>size</b>	Retorna o número de elementos do conjunto
<b>empty</b>	Verifica se o conjunto está vazio
<b>clear</b>	Remove todos os elementos do conjunto

Lista completa disponível em <http://www.cplusplus.com/reference/set/set/>

---

# Set

## Exemplo

```
#include <iostream>
using std::cout;
using std::endl;

#include <string>
using std::string;

#include <set>
using std::set;

int main() {
    set<string> frutas;           // Criacao de um conjunto de strings vazio
    frutas.insert("banana");     // Insercao de elementos no conjunto
    frutas.insert("morango");
    frutas.insert("abacaxi");
}
```

---

# Set

## Exemplo (cont.)

```
set<string>::iterator it = frutas.find("morango");    // Busca por elemento no conjunto
if (it != frutas.end()) {
    cout << "Elemento encontrado" << endl;
} else {
    cout << "Elemento nao encontrado" << endl;
}

return 0;
}
```



---

# Map

- Disponível através da inclusão da biblioteca `<map>`
  - Referência: <http://www.cplusplus.com/reference/map/map/>
- Como trata-se de um *template*, é necessário indicar
  - o tipo das chaves do mapa
  - o tipo de dados/objetos aos quais as chaves serão associadas
- Variações:
  - `map`: permite associar apenas um valor para cada chave
  - `multimap`: permite associar uma mesma chave a mais de um valor

---

# Map

Principais métodos:

Método	Descrição
<b>insert</b>	Insere um par <i>chave-valor</i> no mapa
<b>erase</b>	Remove um valor do mapa por meio de sua respectiva chave
<b>operator[]</b>	Acessa o valor associado a uma chave
<b>find</b>	Busca por um valor por meio de uma chave, retornando um iterador como um par <i>chave-valor</i>
<b>size</b>	Retorna o número de elementos do mapa
<b>empty</b>	Verifica se o mapa está vazio
<b>clear</b>	Remove todos os elementos do mapa

Lista completa disponível em <http://www.cplusplus.com/reference/map/map/>

---

# Map

## Exemplo

```
#include <iostream>
using std::cout;
using std::endl;

#include <map>
using std::map;
using std::find;
using std::insert;

#include <string>
using std::string;

#include <utility>
using std::pair;

int main() {
    map<int, string> alunos;
    alunos.insert(pair<int, string>(1, "Maria"));
    alunos.insert(pair<int, string>(2, "Joao"));
    alunos[3] = "Ana";
    // Cria mapa vazio com chaves int e valores string
    // Insercao de pares chave-valor
    // Insercao de valor com chave 3
```

---

# Map

## Exemplo

```
alunos.erase(2); // Remocao de valor com chave 2
map<int, string>::iterator it;
for (it = alunos.begin(); it != alunos.end(); ++it) // Impressao dos valores no mapa
    cout << it->first << " - " << it->second << endl;

map<int, string>::iterator busca = alunos.find(1);
cout << "Codigo: " << busca->first << " - Nome: " << busca->second << endl;

return 0;
}
```

---

# Outros *containers*

<i>Containers</i>	Referência
<b>array</b>	<a href="http://www.cplusplus.com/reference/array/array/">http://www.cplusplus.com/reference/array/array/</a>
<b>forward_list</b>	<a href="http://www.cplusplus.com/reference/forward_list/forward_list/">http://www.cplusplus.com/reference/forward_list/forward_list/</a>
<b>map</b>	<a href="http://www.cplusplus.com/reference/map/map/">http://www.cplusplus.com/reference/map/map/</a>
<b>multimap</b>	<a href="http://www.cplusplus.com/reference/map/multimap/">http://www.cplusplus.com/reference/map/multimap/</a>
<b>set</b>	<a href="http://www.cplusplus.com/reference/set/set/">http://www.cplusplus.com/reference/set/set/</a>
<b>multiset</b>	<a href="http://www.cplusplus.com/reference/set/multiset/">http://www.cplusplus.com/reference/set/multiset/</a>
<b>unordered_map</b>	<a href="http://www.cplusplus.com/reference/unordered_map/unordered_map/">http://www.cplusplus.com/reference/unordered_map/unordered_map/</a>
<b>unordered_multimap</b>	<a href="http://www.cplusplus.com/reference/unordered_map/unordered_multimap/">http://www.cplusplus.com/reference/unordered_map/unordered_multimap/</a>
<b>unordered_set</b>	<a href="http://www.cplusplus.com/reference/unordered_set/unordered_set/">http://www.cplusplus.com/reference/unordered_set/unordered_set/</a>
<b>unordered_multiset</b>	<a href="http://www.cplusplus.com/reference/unordered_set/unordered_multiset/">http://www.cplusplus.com/reference/unordered_set/unordered_multiset/</a>

# Iteradores (*iterators*)

Objetos usados para **acessar os elementos de um *container***, da mesma forma que um índice acessa os elementos de um arranjo

---

# Iteradores

- Motivação: permitir uma maneira **unificada** de percorrer os elementos de um *container*
- Cada classe *container* define um iterador apropriado
  - `vector<int>::iterator it`  
instancia um iterador (`it`) para percorrer um `vector` de inteiros
  - `map<int, string>::iterator elemento`  
instancia um iterador (`elemento`) para acessar um `map` com chaves do tipo inteiro e valores do tipo *string*
  - `list<Estudante>::iterator valor`  
instancia um iterador (`valor`) para percorrer uma `list` de objetos da classe `Estudante`
- Iteradores ocultam os detalhes de implementação (principalmente o uso de ponteiros) das aplicações, tornando possível trocar o tipo de *container* e ainda assim usar o mesmo código

---

# Iteradores

- Iteradores comuns: ***first*** e ***last***, que apontam respectivamente para o início e fim de uma sequência de elementos
- Iteradores podem ser incrementados ou decrementados com os operadores `++` e `--`, podendo ir para a frente ou para trás enquanto percorrem o *container*
- Iteradores são de-referenciados com o operador `*`  
(uma vez que iteradores são implementados como ponteiros)



---

# Iteradores

## Classificação dos iteradores

Iterador	Descrição	Direção	Acesso
<i>Entrada</i>	Lê valores de um <i>container</i>	Para a frente	Linear
<i>Output</i>	Escreve valores em um <i>container</i>	Para a frente	Linear
<i>Forward</i>	Lê/escreve valores em um <i>container</i>	Para a frente	Linear
<i>Bidirectional</i>	Lê/escreve valores em um <i>container</i>	Para a frente e para trás	Linear
<i>Random access</i>	Lê/escreve valores em um <i>container</i>	Para a frente e para trás	Aleatório

---

# Iteradores

Para facilitar o uso de iteradores, todos os *containers* definem os seguintes métodos:

- `iterator begin()` : retorna iterador que aponta para o **primeiro elemento** do *container*
- `iterator end()` : retorna iterador que aponta para **uma posição após o último elemento** do *container*

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

int main() {
    vector<int> codigos;           // Vetor de inteiros
    vector<int>::iterator it;     // Iterador sobre o vetor de inteiros

    for (it = codigos.begin(); it != codigos.end(); ++it) {
        // Iteracao sobre os elementos do vetor utilizando o iterador it
    }

    return 0;
}
```

---

# Iteradores

Para facilitar o uso de iteradores, todos os *containers* definem os seguintes métodos:

- `reverse_iterator rbegin()`: retorna **iterador reverso** para o **último elemento** do *container*
- `reverse_iterator rend()`: retorna um **iterador reverso** para **uma posição antes do primeiro elemento** do *container*

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

int main() {
    vector<int> codigos;                // Vetor de inteiros
    vector<int>::reverse_iterator itr;  // Iterador reverso sobre o vetor de inteiros
    for (itr = codigos.rbegin(); itr != codigos.rend(); ++itr) {
        // Iteracao sobre os elementos do vetor utilizando o iterador itr
    }

    return 0;
}
```

# Alguma Questão?

