Aula 4:
C++ Standard Template Library

***Disciplina*: Maratona de Programação 1**

Profs. Edmilson Marmo e Luiz Olmes

edmarmo@unifei.edu.br, olmes@unifei.edu.br

Nas aulas anteriores...

- **O QUE JÁ ESTUDAMOS?**
  - Apresentação da Disciplina.
  - Introdução à Maratona.
- **OBJETIVOS:**
  - C++ STL
  - Containers:
    - vector
    - stack
    - queue
    - set
    - map
  - Iteradores
  - Warm up

Standard Template Library

- A Standard Template Library (STL) é uma biblioteca C++ que define componentes reutilizáveis, baseados em modelos (*templates* ou gabaritos).

- Estes componentes implementam muitas estruturas de dados e algoritmos comuns usados para processá-las.

- A STL possui três componentes chaves:
  - Containers: estruturas de dados capazes de armazenar elementos.
  - Iteradores: usados para manipular os elementos nos containers.
  - Algoritmos: funções que manipulam os dados (busca, comparação, ordenação...).

Containers

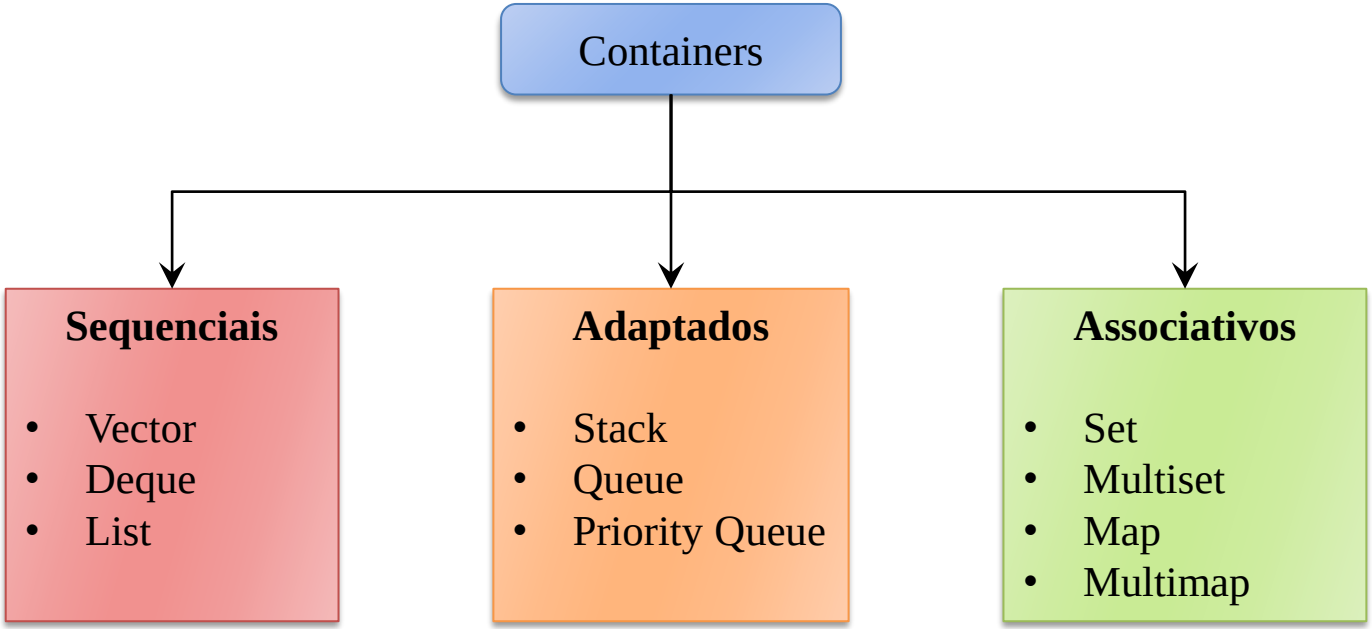
- Os containers da STL são classificados em:

- Sequenciais: usados para implementar estruturas de dados sequenciais, como listas, vetores, etc.

- Associativos: tipos de containers especiais onde cada elemento tem um valor associado a uma chave de busca.

- Adaptados: containers que especificam uma interface, usada para adicionar funcionalidades a containers pré-existentes.

Principais Containers



Containers Sequenciais

- Vector: podem ser vistos como arrays dinâmicos, ou arrays com propriedades especiais.
  - Sintaxe: `vector<tipo> v;`
- Deque: também chamados de *double-ended queue*, que permite operações de inserção e remoção em suas duas extremidades. Um deque é mais eficiente que um vector em operações de inserção e remoção.
  - Sintaxe: `deque<tipo> d;`
- List: é um container que permite alocação não contígua. Além disso, inserções e remoções podem ser feitas em qualquer posição.
  - Sintaxe: `list<tipo> l;`

Containers Adaptados

- Stack: container com política de inserção/remoção bem definida: LIFO (last-in, first out).
  - Sintaxe: `stack<tipo> st;`
- Queue: container com política de inserção/remoção bem definida: FIFO (first-in, first out).
  - Sintaxe: `queue<tipo> q;`
- Priority Queue: container com política de inserção/remoção bem definida: FIFO (first-in, first out), porém, nas remoções, o maior elemento é o primeiro a ser removido.
  - Sintaxe: `priority_queue<tipo> pq;`

Containers Associativos

- Set: usado para armazenar elementos que são únicos (sem repetição).
  - Sintaxe: `set<tipo> s;`
- Multiset: similar ao Set, porém, permite repetição de elementos.
  - Sintaxe: `multiset<tipo> ms;`
- Map: contém dados na forma chave-valor, sem repetição de chaves.
  - Sintaxe: `map<tipo, tipo> m;`
- Multimap: similar ao Map, porém, vários elementos podem estar associados a mesma chave.
  - Sintaxe: `multimap<tipo, tipo> mm;`

Vector

- Um vector representa um array dinâmico, isto é, um array que não precisa ter um tamanho fixo, pré-definido.

- O arquivo de cabeçalho `vector` é necessário para se utilizar a estrutura vector.
  - `#include <vector>` (ou `#include <bits/stdc++.h>`)

- Além disso, o tipo de dado que o container armazena é declarado entre os símbolos de < e >.
  - `vector<int> v1; // um vector de inteiros`
  - `vector<double> v2; // um vector de números reais de precisão dupla`

Vector: principais comandos

- `push_back(e)`: insere o elemento `e` no final do vector.
- `size()`: devolve a quantidade de elementos presentes no vector.
- `clear()`: remove todos os elementos presentes no vector.
- `begin()` e `end()`: devolve iteradores para o início e fim do vector.
- Operador `[]`: permite acessar elementos, da mesma forma que em um array.
- `erase(it)`: apaga o elemento referenciado pelo iterador `it`.

Operações úteis

- Ordenando um vector de inteiros (ordem crescente):

```
1.   vector<int> vec = {1, 3, 5, 7, 9, 2, 4, 6, 8};
2.   sort( vec.begin(), vec.end() );
```

- Ordenando um vector de inteiros (ordem decrescente):

```
1.   vector<int> vec = {1, 3, 5, 7, 9, 2, 4, 6, 8};
2.   sort( vec.rbegin(), vec.rend() );
```

- Eliminando duplicatas de um vector de inteiros:

```
1.   vector<int> vec = {1, 3, 5, 3, 2, 2, 1, 4, 8, 9, 3, 7};
2.   vector<int>::iterator it;
3.   sort( vec.begin(), vec.end() );
4.   it = unique( vec.begin(), vec.end() );
5.   vec.resize( distance(vec.begin(), it) );
```

Queue

- A estrutura de dados queue (ou fila) é outra estrutura da STL e simula uma fila. Nela, podemos inserir elementos no fim da fila e remover o elemento mais à frente.

- Na Ciência da Computação, uma fila é uma estrutura de dados empregada para armazenar e controlar o fluxo de dados em um computador:
  - Fila da impressora.
  - Fila de processos da CPU.
  - Transferência de dados em redes de comunicação.
  - Filas de prioridade.

Queue

- O arquivo de cabeçalho `queue` é necessário para se utilizar a estrutura queue.
  - `#include <queue>`

- Além disso, o tipo de dado que o container armazena é declarado entre os símbolos de < e >.
  - `queue<int> q1; // um queue de inteiros`
  - `queue<float> q2; // um queue de números reais`

- `priority_queue`: fila de prioridade. A remoção remove o maior elemento.

- Ambas as estruturas não possuem `iterator`.

Queue: principais comandos

- `push(e)`: insere o elemento `e` no final do queue.
- `pop()`: remove o primeiro elemento do queue.
- `front()`: acessa o elemento que está na primeira posição do queue.
- `back()`: acessa o elemento que está na última posição do queue.
- `empty()`: verifica se o queue está vazio ou não (booleano).
- `size()`: devolve a quantidade de elementos presentes no queue.

Stack

- A Stack é uma estrutura de dados da STL que simula uma pilha, de forma que só é possível inserir e remover elementos do seu topo.

- Na Ciência da Computação, uma pilha é uma estrutura de dados empregada em diversas situações:
  - Recursão.
  - Avaliação de expressões.
  - Parsing, em compiladores.
  - Busca em profundidade em grafos.
  - Operações de Desfazer e Refazer.
  - Chamada de funções.

Stack

- O arquivo de cabeçalho `stack` é necessário para se utilizar a estrutura stack.
  - `#include <stack>`

- Além disso, o tipo de dado que o container armazena é declarado entre os símbolos de < e >.
  - `stack<int> s1; // um stack de inteiros`
  - `stack<long long int> s2; // um stack de inteiros longos`

- A estrutura stack não possui `iterator`.

Stack: principais comandos

- `push(e)`: insere o elemento `e` no início da stack.
- `pop()`: remove o primeiro elemento da stack.
- `top()`: acessa o elemento que está na primeira posição (topo) da stack.
- `empty()`: verifica se a stack está vazia ou não (booleano).
- `size()`: devolve a quantidade de elementos presentes na stack.

Containers Associativos

- Um container associativo fornece acesso direto aos elementos nele armazenados através de uma chave (chave de busca).

- Quatro containers associativos são definidos pela C++ STL:
  - `set`, `multiset`
  - `map`, `multimap`

- Todos eles mantém suas chaves ordenadas.

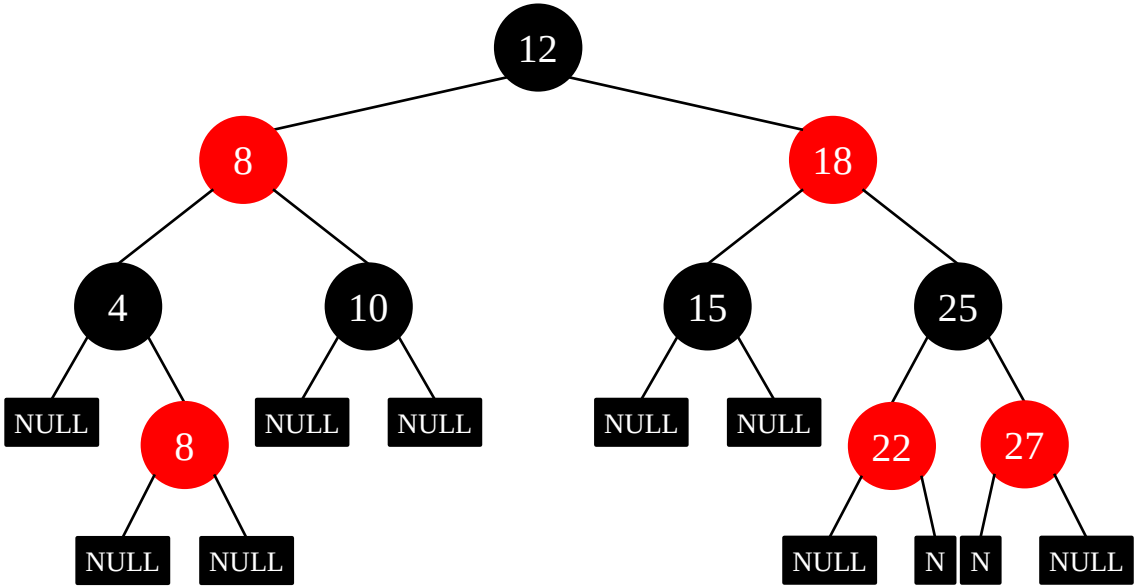
- Além destes, quatro outro containers equivalentes também são definidos, porém, estes não mantém a ordenação de chaves:
  - `unordered_set`, `unordered_multiset`, `unordered_map`, `unordered_multimap`.

Containers Associativos: `set` e `multiset`

- As classes `set` e `multiset` são usadas para manipular conjuntos de valores em que estes valores são as chaves de busca.
- Enquanto que um `multiset` permite valores duplicados, um `set` não permite.
- Ambas as estruturas são definidas no arquivo de cabeçalho `set`, sendo normalmente implementadas através de Árvores Red-Black (*próximo slide*).
  - `#include <set>` para as duas estruturas (`set` e `multiset`).
- Sintaxe:
  - `set<int> st;`
  - `multiset<int> ms;`

Árvore Red-Black

- Exemplo:



Containers Associativos: `set` e `multiset`

- A iteração (com iterators) sobre ambas as estruturas ocorre de forma ordenada sobre as chaves de busca, ainda que a inserção seja desordenada.
  - Iteradores bidirecionais podem ser usados para navegação.

- As operações de busca, remoção e inserção de elementos são realizadas em tempo logarítmico, para ambas as estruturas.

- No caso de set, tentativas de inserir valores duplicados são ignoradas.
  - Isto é um comportamento comum de conjuntos, na matemática. Por isso, não é considerado um erro de programação.

`set` e `multiset`: principais comandos

- `insert(e)`: insere o elemento `e` na estrutura.
- `size()`: devolve a quantidade de elementos presentes na estrutura.
- `clear()`: remove todos os elementos da estrutura.
- `begin()` e `end()`: devolve iteradores para o início e fim da estrutura.
  - `rbegin()` e `rend()`: idem, mas para `reverse_iterator`
- `find(e)`: busca pelo elemento `e`. Se o elemento estiver presente, devolve um iterador para ele. Caso contrário, devolve `end()`.
  - `count(e)`: conta quantas vezes o elemento `e` aparece na estrutura.
- `erase(it)`: apaga o elemento referenciado pelo iterador `it`.

Containers Associativos: `map` e `multimap`

- As classes `map` e `multimap` são usadas para manipular elementos compostos por uma chave e um valor.
  - Por exemplo: um dicionário de idiomas. A chave é a palavra em uma língua e o valor é a tradução para a segunda língua.
  - Sintaxe:
    - `map<string, string> mp;`
    - `multimap<string, string> mm;`

- Em ambas as estruturas, os elementos inseridos são pares de valores, ao invés de elementos individuais.

- Dessa forma, para o processo de inserção, é necessário usar a função `make_pair(c, v)`, para criar o par chave / valor (de acordo com os tipos).
  - Ou, a partir de C++17, inserir o par entre chaves: `mymap.insert( {2, Ana} );`

Containers Associativos: `map` e `multimap`

- Ambas as estruturas são definidas no arquivo de cabeçalho `map`, sendo normalmente implementadas através de Árvores Red-Black.
  - `#include <map>` para as duas estruturas (`map` e `multimap`).
  - A ordenação das chaves se mantém.

- Uma vez que um `multimap` permite duplicidade, múltiplos valores podem ser associados a uma mesma chave.
  - Os valores seguem a ordem de inserção. A chave se mantém ordenada.

- Isto é chamado de relacionamento um-para-muitos:
  - Um cartão de crédito pode ter várias transações associadas a ele.
  - Um estudante pode se matricular em várias disciplinas.

`map` e `multimap`: principais comandos

- `insert( make_pair(c, v) )`: insere o par `<c, v>` na estrutura.
- `size()`: devolve a quantidade de elementos presentes na estrutura.
- `begin()` e `end()`: devolve iteradores para o início e fim da estrutura.
  - `rbegin()` e `rend()`: idem, mas para `reverse_iterator`
- `find(c)`: busca pela chave `c`. Se a chave estiver presente, devolve um iterador para seu par. Caso contrário, devolve `end()`.
  - `count(c)`: conta quantos elemento possuem `c` como chave.
- `erase(it)`: apaga o elemento referenciado pelo iterador `it`.
- Operador `[]` (apenas `map`): permite acessar o valor informando sua chave.

Iteradores

- Um iterador (*iterator*, em inglês) é um objeto que aponta para um elemento dentro de uma coleção ou de um container.

- Possui a capacidade de percorrer o container através de operações de incremento (++) ou dereferenciação (*).

- Normalmente está combinado com as seguintes operações:
  - `begin()`: retorna a primeira posição do container (primeiro elemento).
  - `end()`: retorna a posição após o último elemento do container.

Iteradores

- Exemplo: iterando sobre um vector de inteiros.

```
1.   vector<int> vec = {1, 3, 5, 3, 2, 2, 1, 4, 8, 9, 3, 7};
2.   vector<int>::iterator it;
3.
4.   for( it = vec.begin(); it != vec.end(); it++ )
5.   {
6.       cout << *it;
7.   }
8.
```

Dúvidas?

Aula 4:
C++ Standard Template Library

***Disciplina*: Maratona de Programação 1**

Profs. Edmilson Marmo e Luiz Olmes

edmarmo@unifei.edu.br, olmes@unifei.edu.br