

Programação



UFOP



INSTITUTO FEDERAL
MINAS GERAIS
Campus Ouro Preto



Standard Templates Library

STL

- A *Standard Templates Library* (ou STL) é uma biblioteca que faz parte da biblioteca padrão da linguagem C++
- Composta por contêineres (ou estruturas de dados), iteradores, operadores e implementação de diversos algoritmos
 - Alto desempenho.

STL

- **Contêineres** são estruturas de dados genéricas
 - Possuem métodos associados a eles.
- **Iteradores** são semelhantes a ponteiros, utilizados para percorrer e manipular os elementos de um contêiner;
- **Algoritmos** são as funções que realizam operações tais como buscar, ordenar e comparar elementos ou contêineres inteiros
 - Existem aproximadamente 85 algoritmos implementados na STL;
 - A maioria utiliza iteradores para acessar os elementos de contêineres.

STL

- Crítica comum:
 - As mensagens de erro envolvendo a STL não são claras;
 - Dificultam a depuração do código.

Contêineres

Contêineres

Contêineres Sequenciais	Descrição
<i>vector</i>	Inserções e remoções no final, acesso direto a qualquer elemento.
<i>deque</i>	Fila duplamente ligada, inserções e remoções no início ou no final, acesso direto a qualquer elemento.
<i>list</i>	Lista duplamente ligada, inserção e remoção em qualquer ponto.

Contêineres

Contêineres Associativos	Descrição
<i>set</i>	Busca rápida, não permite elementos duplicados.
<i>multiset</i>	Busca rápida, permite elementos duplicados.
<i>map</i>	Mapeamento um-para-um, não permite elementos duplicados, busca rápida.
<i>multimap</i>	Mapeamento um-para-um, permite elementos duplicados, busca rápida.

Contêineres

Adaptadores de Contêineres	Descrição
<i>stack</i>	Last-in, first out (LIFO)
<i>queue</i>	First –in, first out (FIFO)
<i>priority_queue</i>	O elemento de maior prioridade é sempre o primeiro elemento a sair.

Funções Comuns a Todos Contêineres

Funcionalidade	Descrição
<i>empty</i>	Retorna <i>true</i> se não houver elementos no contêiner e <i>false</i> caso contrário.
<i>size</i>	Retorna o número de elementos no contêiner.
<i>operator=</i>	Atribui um contêiner a outro.
<i>operator<</i>	Retorna <i>true</i> se o primeiro contêiner for menor que o segundo e <i>false</i> caso contrário.

Funções Comuns a Todos Contêineres

Funcionalidade	Descrição
<code>operator<=</code>	Retorna <i>true</i> se o primeiro contêiner for menor ou igual ao segundo e <i>false</i> caso contrário.
<code>operator></code>	Retorna <i>true</i> se o primeiro contêiner for maior que o segundo e <i>false</i> caso contrário.
<code>operator>=</code>	Retorna <i>true</i> se o primeiro contêiner for maior ou igual ao segundo e <i>false</i> caso contrário.
<code>operator==</code>	Retorna <i>true</i> se o primeiro contêiner for igual ao segundo e <i>false</i> caso contrário.
<code>operator!=</code>	Retorna <i>true</i> se o primeiro contêiner for diferente do segundo e <i>false</i> caso contrário.

Funções Comuns a Todos Contêineres

■ Atenção!

- Os operadores `<`, `<=`, `>`, `>=`, `==` e `!=` não são fornecidos para o contêiner ***priority_queue***.

Funcionalidade	Descrição
<i>max_size</i>	Retorna o número máximo de elementos de um contêiner.
<i>begin</i>	As duas versões deste método retornam um <i>iterator</i> ou um <i>const_iterator</i> para o primeiro elemento do contêiner.
<i>end</i>	As duas versões deste método retornam um <i>iterator</i> ou um <i>const_iterator</i> para a posição após o final do contêiner.
<i>rbegin</i>	As duas versões deste método retornam um <i>reverse_iterator</i> ou um <i>const_reverse_iterator</i> para o primeiro elemento do contêiner invertido.
<i>rend</i>	As duas versões deste método retornam um <i>reverse_iterator</i> ou um <i>const_reverse_iterator</i> para a posição após o final do contêiner invertido.
<i>erase</i>	Apaga um ou mais elementos do contêiner.
<i>clear</i>	Apaga todos os elementos do contêiner.

Bibliotecas de Contêineres

Biblioteca	Observação
<vector>	Vetor
<list>	Lista
<deque>	Fila duplamente ligada
<queue>	Contém <i>queue</i> e <i>priority_queue</i>
<stack>	Pilha
<map>	Contém <i>map</i> e <i>multimap</i>
<set>	Contém <i>set</i> e <i>multiset</i>

Iteradores

Iteradores

- **Iteradores** são utilizados para apontar elementos de contêineres sequenciais e associativos
 - Entre outras coisas;
 - Algumas funcionalidades como *begin* e *end* retornam iteradores.
- Se um iterador *i* aponta para um elemento:
 - ***++i*** aponta para o próximo elemento;
 - ****i*** se refere ao conteúdo do elemento apontado por *i*.

Iteradores

- Os iteradores são objetos declarados na biblioteca `<iterator>`;
- Existem basicamente dois tipos de objetos iteradores:
 - *iterator*: aponta para um elemento que pode ser modificado;
 - *const_iterator*: aponta para um elemento que não pode ser modificado.

Operações em Iteradores

<i>Input</i>	<i>Descrição</i>
*p	Referencia o conteúdo apontado.
p = p1	Atribui um iterador a outro.
p == p1	Compara dois iteradores quanto a igualdade.
p != p1	Compara dois iteradores quanto a desigualdade.
++p	Incremento prefixado.
p++	Incremento pós-fixado.

Algoritmos

Algoritmos

- A STL inclui mais de 85 algoritmos
 - Podem ser utilizados genericamente, em vários tipos de contêineres.
- Os algoritmos operam indiretamente sobre os elementos de um contêiner usando iteradores
 - Vários deles utilizam pares de iteradores, um apontando para o início e outro apontando para o final;
 - Frequentemente os algoritmos também retornam iteradores como resultado;
 - Este desacoplamento dos contêineres permite que os algoritmos sejam genéricos.

Algoritmos

<code>copy</code>	<code>remove</code>	<code>reverse_copy</code>
<code>copy_backward</code>	<code>remove_copy</code>	<code>rotate</code>
<code>fill</code>	<code>remove_copy_if</code>	<code>rotate_copy</code>
<code>fill_n</code>	<code>remove_if</code>	<code>stable_partition</code>
<code>generate</code>	<code>replace</code>	<code>swap</code>
<code>generate_n</code>	<code>replace_copy</code>	<code>swap_ranges</code>
<code>iter_swap</code>	<code>replace_copy_if</code>	<code>transform</code>
<code>partition</code>	<code>replace_if</code>	<code>unique</code>
<code>random_shuffle</code>	<code>reverse</code>	<code>unique_copy</code>

<code>adjacent_find</code>	<code>find</code>	<code>find_if</code>
<code>count</code>	<code>find_each</code>	<code>mismatch</code>
<code>count_if</code>	<code>find_end</code>	<code>search</code>
<code>equal</code>	<code>find_first_of</code>	<code>search_n</code>



Altera sequência

Não Altera sequência

Ordenação

Algoritmo	Descrição
<i>sort</i>	Ordena os elementos do contêiner
<i>stable_sort</i>	Ordena os elementos do contêiner preservando a ordem relativa dos equivalentes.
<i>partial_sort</i>	Ordena parcialmente o contêiner.
<i>partial_sort_copy</i>	Copia os menores elementos e os ordena no contêiner de destino.
<i>nth_element</i>	Ordena o n -ésimo elemento.

Busca Binária e Operações em Conjuntos

Algoritmo	Descrição
<i>binary_search</i>	Testa se um valor existe em um intervalo.
<i>set_union</i>	Calcula a união entre dois intervalos de valores.
<i>set_intersection</i>	Calcula a interseção entre dois intervalos de valores.
<i>set_difference</i>	Calcula a diferença entre dois intervalos de valores.
<i>set_symmetric_difference</i>	Calcula a diferença simétrica entre dois intervalos de valores.

Min/Max

Algoritmo	Descrição
<i>min</i>	Retorna o menor de dois argumentos.
<i>max</i>	Retorna o maior de dois argumentos.
<i>min_element</i>	Retorna o menor elemento de uma sequência.
<i>max_element</i>	Retorna o maior elemento de uma sequência.

vector

vector

- A classe *vector* implementa a estrutura de dados sequencial e contígua **vetor** (ou *array*);
- Vantagens:
 - Pode alterar seu tamanho dinamicamente
 - Normalmente, somente a extremidade final.
 - Rápido acesso a elementos, usando o índice adequado;
 - Relativamente eficiente para remover elementos do final.
- Desvantagens:
 - A remoção de elementos no início não é tão eficiente.

vector

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

```
int main ()
{
    unsigned int i;
```

```
    vector<int> first; // vector de ints vazio
    vector<int> second, third;
    vector<int> fourth (third); // copia o terceiro vector
```

```
    // o construtor iterador também pode ser utilizado com arrays
```

```
    int myints[] = {16, 2, 77, 29};
    vector<int> fifth (myints, myints + sizeof(myints) / sizeof(int) );
    fifth[0] = 10;
```

vector

```
cout << "O tamanho do quinto é:" << fifth.size() << endl;
//adiciona o elemento ao final do vector
first.push_back(3);
//remove o elemento ao final do vector
first.pop_back();
//atribuição direta
second = third;
//iterador inicio para o final
vector< int >::iterator it;
// exibe elementos vector utilizando const_iterator
for ( it = first.begin(); it != first.end(); ++it )
    cout << *it << ' ';
return 0;
}
```

vector

- A classe *vector* é genérica, logo, deve ser definido o tipo na declaração de um objeto;
- Este contêiner é dinâmico
 - A cada inserção o contêiner se redimensiona automaticamente.
- O método ***push_back*** adiciona um elemento ao final do *vector*;
- Analogamente, o método ***pop_back*** remove o elemento ao final do *vector*.

vector

- ▣ Outros possíveis métodos incluem:
 - ▣ **front**: determina o primeiro elemento;
 - ▣ **back**: determina o último elemento;
 - ▣ **at**: determina o elemento em uma determinada posição, mas antes verifica se é uma posição válida;
 - ▣ **insert**: insere um elemento em uma posição especificada por um iterador;
 - ▣ **erase**: remove um elemento em uma posição especificada por um iterador;
 - ▣ **clear**: esvazia o vector.

cplusplus.com

- Verificar conteúdo de vector no site cplusplus.com



Perguntas?