

IMD0030

LINGUAGEM DE PROGRAMAÇÃO I

Aula 23 – *Standard Template Library* (STL)

Algoritmos e Laboratório 5

Objetivos desta aula

- Introduzir elementos adicionais da *Standard Template Library* (STL)
- Para isso, estudaremos:
 - Algoritmos (*algorithms*) definidos na STL
- Ao final da aula, espera-se que o aluno seja capaz de fazer uso de algoritmos providos pela STL

Algoritmos (*algorithms*)

Funções genéricas baseadas em iteradores que executam operações sobre os elementos de um *container*

Algoritmos

- Embora tenham sido projetados para operar sobre os *containers* da STL, algoritmos também podem ser utilizados sobre arranjos de C++
- A STL adota a filosofia de manter os algoritmos **fora** das classes que implementam os *containers*, para permitir que um mesmo algoritmo possa agir sobre *containers* diferentes
- Os algoritmos estão agrupados em quatro categorias:

Categoria	Biblioteca
Não-modificação de sequências	<algorithm>
Modificação de sequências	<algorithm>
Ordenação	<algorithm>
Algoritmos numéricos	<numeric>

Algoritmos de não-modificação de sequências

- Não alteram diretamente os *containers* sobre os quais operam
- Necessário incluir a biblioteca `<algorithm>`
<http://www.cplusplus.com/reference/algorithm/>
- Exemplos:
 - **count()** : conta o número de ocorrências de um determinado elemento no *container*
 - **find()** : busca por um elemento no *container*, retornando um iterador que aponta para a primeira posição na qual tal elemento foi encontrado
 - **for_each()** : itera sobre o *container*, aplicando uma função sobre cada elemento

Algoritmos de não-modificação de sequências

Exemplo:

```
#include <algorithm>
using std::count;
using std::find;
using std::for_each;

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

#include <string>
using std::string;

#include <vector>
using std::vector;

void imprime(string nome) {
    cout << nome << " ";
}
```

Algoritmos de não-modificação de sequências

Exemplo:

```
int main() {  
    // Vetor de strings com quatro elementos (exclusivamente em C++11)  
    vector<string> nomes = { "Maria", "Joao", "Mauricio", "Joao" };  
  
    // Conta o numero de ocorrencias de "Joao" no vetor  
    int ocorrencias = count(nomes.begin(), nomes.end(), "Joao");  
    cout << "Foram encontradas " << ocorrencias << " ocorrencias no vetor" << endl;  
  
    // Encontra a primeira ocorrência de "Joao" no vetor  
    vector<string>::iterator it = find(nomes.begin(), nomes.end(), "Joao");  
    if (it == nomes.end()) {  
        cout << "String nao encontrada" << endl;  
    }  
  
    // Chama a funcao imprime para cada elemento do vetor  
    for_each(nomes.begin(), nomes.end(), imprime);  
  
    return 0;  
}
```

O uso do `find_if()` e o conceito de Functor

- Definição básica:

```
template< class InputIt, class UnaryPredicate >
InputIt find_if( InputIt first, InputIt last, UnaryPredicate p );
```

- **`find_if()`**: busca por um elemento no *container*, retornando um iterador que aponta para a primeira posição na qual um **predicado** é verdadeiro
 - Predicados são funções ou *functors* que retornam um valor booleano e são aplicadas a cada elemento de um container
 - **Functors** são funções objeto. Em outras palavras, são objetos que se comportam como funções ao mesmo tempo que armazenam dados.
 - Para isso, todo **functor** deve implementar o método **`bool operator()`**

O uso do find_if() e o conceito de Functor

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

// Definindo uma funcao a ser usada como predicado
bool maiorQue60 (int valor) {
    return 60 < valor;
}

// Definindo o Functor MaiorQue
class MaiorQue
{
    int valor;
public:
    MaiorQue(int n):valor(n) {}
    bool operator()(int n) const { return n > valor; }
};
```

O uso do find_if() e o conceito de Functor

```
int main() {  
    // Vetor de inteiros  
    vector<int> valores { 2, 44, 56, 13, 88, 78, 90, 3, 4, 5 };  
  
    // Encontra a primeira ocorrência maior que 60 usando uma função como predicado  
    auto it = find_if(valores.begin(), valores.end(), maiorQue60);  
    if (it != valores.end()) {  
        cout << "Primeiro valor: " << (*it) << endl;  
    } else {  
        cout << "Nao ha valores que satisfacam o predicado. " << endl;  
    }  
  
    // Encontra a primeira ocorrência maior que 89 usando uma functor como predicado  
    it = find_if(valores.begin(), valores.end(), MaiorQue(89));  
    if (it != valores.end()) {  
        cout << "Primeiro valor: " << (*it) << endl;  
    } else {  
        cout << "Nao ha valores que satisfacam o predicado. " << endl;  
    }  
    return 0;  
}
```

Algoritmos de modificação de sequências

- Alteram o conteúdo dos *containers* sobre os quais operam
- Necessário incluir a biblioteca `<algorithm>`
<http://www.cplusplus.com/reference/algorithm/>
- Exemplos:
 - **reverse()** : inverte a ordem dos elementos do *container*
 - **sort()** : reordena os elementos do *container* de forma aleatória
 - **replace()** : substitui a ocorrência de um elemento do *container* por outro elemento
 - **fill()** : preenche o *container* com um determinado elemento

Algoritmos de modificação de sequências

Exemplo:

```
#include <algorithm>
using std::fill;
using std::random_shuffle;
using std::replace;
using std::reverse;

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

#include <vector>
using std::vector;

void imprime(vector<int> v) {                // Funcao para imprimir os elementos de um vetor
    vector<int>::iterator i;
    for (i = v.begin(); i != v.end(); ++i) {
        cout << *i << " ";
    }
    cout << endl;
}
```

Algoritmos de modificação de sequências

Exemplo:

```
int main() {  
    vector<int> numeros; // Vetor de inteiros vazio  
    for (int i = 1; i <= 10; i++) numeros.push_back(i); // Insercao de elementos no vetor  
  
    reverse(numeros.begin(), numeros.end()); // Inverte a ordem dos elementos  
    imprime(numeros);  
  
    random_shuffle(numeros.begin(), numeros.end()); // Reordena aleatoriamente os elementos  
    imprime(numeros);  
  
    replace(numeros.begin(), numeros.end(), 10, 100); // Substitui a ocorrencia de um valor  
    imprime(numeros);  
  
    fill(numeros.begin(), numeros.end(), 0); // Preenche todos os elementos do vetor com 0  
    imprime(numeros);  
  
    return 0;  
}
```

Algoritmos relacionados a ordenação

- Executam operações de ordenação, intercalação, comparação, busca e de conjunto sobre os elementos de um *container*
- Necessário incluir a biblioteca `<algorithm>`
<http://www.cplusplus.com/reference/algorithm/>
- Exemplos:
 - **max_element()** : retorna um iterador que aponta para a primeira ocorrência do maior elemento do *container*
 - **sort()** : ordena os elementos do *container*
 - **binary_search()** : realiza uma busca binária sobre os elementos do *container*

Algoritmos relacionados a ordenação

Exemplo:

```
#include <algorithm>
using std::binary_search
using std::max_element;
using std::sort;

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

#include <vector>
using std::vector;

void imprime(vector<int> v) { // Funcao para imprimir os elementos de um vetor
    vector<int>::iterator i;
    for (i = v.begin(); i != v.end(); ++i) {
        cout << *i << " ";
    }
    cout << endl;
}
```

Algoritmos relacionados a ordenação

Exemplo:

```
int main() {  
    int num_array[] = { 12, 5, 21, 23, 44, 38 };  
  
    // Criação de um vetor de inteiros a partir do array  
    vector<int> numeros(num_array, num_array + sizeof(num_array) / sizeof(int));  
  
    // Retorna um iterador para o primeiro elemento com maior valor no vetor  
    vector<int>::iterator it = max_element(numeros.begin(), numeros.end());  
    cout << "Maior valor no vetor: " << *it << endl;  
  
    // Ordena os tres primeiros elementos do vetor  
    sort(numeros.begin(), numeros.begin()+3);  
    imprime(numeros);  
}
```

Algoritmos relacionados a ordenação

Exemplo:

```
// Ordena todo o vetor e realiza uma busca binaria pelo valor 38
sort(numeros.begin(), numeros.end());
if (binary_search(numeros.begin(), numeros.end(), 38)) {
    cout << "Valor encontrado" << endl;
} else {
    cout << "Valor nao encontrado" << endl;
}

return 0;
}
```

Algoritmos numéricos genéricos

- Executam operações numéricas sobre os elementos de um *container*
- Necessário incluir a biblioteca `<numeric>`
<http://www.cplusplus.com/reference/numeric/>
- Exemplos:
 - **`accumulate()`** : retorna o acumulado dos elementos do vetor
 - Por padrão, a operação a ser realizada é a de soma dos elementos
 - É possível fornecer um parâmetro adicional referente ao operador binário (por exemplo, uma multiplicação) a ser utilizado para computação do acumulado, presente na biblioteca `<functional>`

Algoritmos numéricos genéricos

Exemplo:

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

#include <numeric>
using std::accumulate;

#include <vector>
using std::vector;

int main() {
    vector<int> numeros;
    for (int i = 1; i <= 5; i++) numeros.push_back(i);

    // Calcula o acumulado (somatório) dos valores no vetor
    int soma = accumulate(numeros.begin(), numeros.end(), 0);
    cout << "Soma dos elementos do vetor: " << soma << endl;
```

Algoritmos numéricos genéricos

Exemplo:

```
// Calcula o acumulado (produtorio) dos valores no vetor
int produto = accumulate(numeros.begin(), numeros.end(), 1, std::multiplies<int>());
std::cout << "Produto dos elementos do vetor: " << produto << endl;

return 0;
}
```

Alguma Questão?

