Linguagem C++ IV

16/12/2024

Sumário

- STL Iterators
- The Standard Algorithms Library
- Exemplos

Iteradores

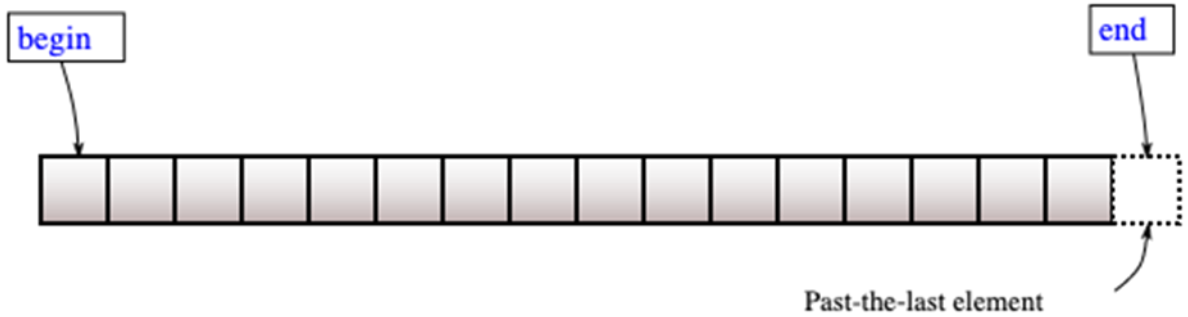
Iteradores

- Permitem aceder a elementos individuais de uma estrutura de dados (container) usando o operador *
- Usados habitualmente para percorrer sequencialmente todos os elementos
- São incrementados ou decrementados usando o operadores ++ e --, passando a referenciar o elemento seguinte ou anterior, caso exista

Iteradores

- Classes de iteradores : forward iterator, bidirectional iterator, random access iterator
- Forward iterator : avançar ao longo de uma estrutura de dados, um elemento de cada vez, usando o operador ++
- Biderectional iterator : avançar ou recuar, um elemento de cada vez, usando o operador ++ ou o operador --
- Random access iterator : aceder a uma posição arbitrária

Iteradores




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

int main(void) {
    std::vector<int> v = { 3, 1, 4 };
    auto vi = std::begin(v);
    std::cout << *vi << '\n'; //3

    int a[] = { -5, 10, 15 };
    auto ai = std::begin(a);
    std::cout << *ai << '\n'; //-5
    return 0;
}
```

```
int main(void) {
    std::vector<int> v = { 3, 1, 4 };
    std::vector<int>::iterator it;

    it = v.begin();
    while(it != v.end()){
        std::cout << *it << "-"; //3-1-4
        it++;
    }

    return 0;
}
```

Ciclo iterador

```
std::vector<int> elems = {1, 2, 3, 4};

for (auto it = elems.begin(); it!=elems.end(); it++){
    std::cout << *it << '\n';
}

for (auto e : elems){
    std::cout << e << '\n';
}
```

Apagar um elemento

```
auto it = elems.begin() + 2;
elems.erase(it);
```

- Apagar o item na posição referenciada pelo iterador
- É devolvido um iterador para o elemento seguinte, se existir
- Ficam inválidos os iteradores e referências anteriormente definidos para o item apagado e para os itens que se lhe seguem, incluindo o iterador .end()

The Standard Algorithms Library

The Standard Algorithms Library

- Disponibiliza funções que permitem executar diferentes tipos de operações sobre sequências de elementos (ranges), usando iteradores

```
#include<algorithm>
#include<numeric>
```

- Essas funções estão incluídas no **`namespace std`**

Funções que não modificam uma sequência

- Verificação de predicados

  `all_of` **`any_of`** **`none_of`** **`count`** …

- Comparação de sequências

  `equal` **`mismatch`** …

- Procura

  `find` **`find_if`** **`search`** **`binary_search`** …

- Procura do maior e do menor

  `max` **`min`** **`minmax`** **`min_element`** …

Funções que modificam uma sequência

- Geração e Cópia

  `fill` **`generate`** **`copy`** …

- Partição

  `unique` **`remove`** **`remove_if`** **`partition`** …

- Ordenação e Reordenação

  `sort` **`reverse`** **`rotate`** **`shuffle`** …

- Transformação

  `for_each` **`transform`** …

Verificação de Predicados

count

```
vector<int> vi = {3, 4, 2, 8, 7, 1, 3, 5};
// Count elements that match target
int c1 = count(vi.begin(), vi.end(), 3);
int c2 = count(vi.begin(), vi.end(), 6);
cout << c1 << endl; // 2
cout << c2 << endl; // 0
```

count_if

```
vector<int> vi = {3, 4, 2, 8, 7, 1, 3, 5};
// Using a lambda expression
int count_div4 = count_if(v.begin(), v.end(),
                 [](int i) { return i % 4 == 0; });
cout << "numbers divisible by four: "
     << count_div4 << '\n'; // 2
```

all_of

```
vector<int> vi = {3, 4, 2, 8, 7, 1, 3, 5};
// Checking a predicate
if (all_of(vi.begin(), vi.end(),
           [](int i) { return i % 2 == 0; })) {
    cout << "All numbers are even\n";
}
```

Comparação de Sequências

equal

```
vector<int> v1 { 0, 5, 10, 15, 20, 25 };
vector<int> v2 { 0, 5, 10, 15, 20, 25 };
vector<int> v3 { 0, 5, 10, 15, 20, 25, 30, 35, 40 };

// Using range-and-a-half equal:
bool b = equal(v1.begin(), v1.end(), v2.begin());
cout << "v1 and v2 are equal: "
     << b << endl;                  // true, as expected
```

equal

```
b = equal(v1.begin(), v1.end(), v3.begin());
cout << "v1 and v3 are equal: "
     << b << endl;                  // true, surprisingly !!

// Using dual-range equal:
b = equal(v1.begin(), v1.end(), v3.begin(), v3.end());
cout << "v1 and v3 are equal with dual-range overload: "
     << b << endl;                  // false
```

Procuras

min

```
cout << "smaller of 'd' and 'b' is `”
     << min('d', 'b') << "'\n"
     << "shortest of \"foo\", \"bar\", and \"hello\" is \""
     << min({"foo", "bar", "hello"},
            [](const string_view s1, const string_view s2)
            { return s1.size() < s2.size(); })
     << "\"\n";
```

max

```
auto comparator = [](const string_view s1, const string_view s2)
                  { return s1.size() < s2.size(); };

cout << "Larger of 69 and 96 is " << max(69, 96) << "\n"
     << "Larger of 'q' and 'p' is '" << max('q', 'p') << "'\n"
     << "(Longest of "long", "short", and "int" is )"
     << quoted(max({"long", "short", "int"}, comparator)) << '\n';
```

max_element

```
vector<int> elems = { 1, 2, 3, 4};

auto e = max_element(begin(elems), end(elems));

cout << "The max element in the vector is: "
     << *e;
```

min_element

```
vector<int> v {3, 1, -4, 1, 5, 9};

vector<int>::iterator result;
result = std::min_element(v.begin(), v.end());
std::cout << "min element has value "
          << *result << " and index ["
          << std::distance(v.begin(), result)
          << "]\n";
```

find

```
std::vector<int> elem = { 1, 5, 2, 15, 3, 10 };
auto e = std::find(std::begin(elem), std::end(elem), 5);
if (e!=elem.end())
{
   std::cout << "Element found: " << *e;
}
else
{
    std::cout << "Element not found";
}
```

find

```
map<int,char> mymap = {{1,'a'}, {2,'b'}, {3,'c'}, {4,'d'}};
auto e = mymap.find(2);
if (e != mymap.end())
{
   cout << "Found: " << e->first << " "
        << e->second << '\n';
}
else { cout << "Not found"; }
```

binary_search

```
// binary_search: works in ordered ranges
vector<int> vi = {1, 2, 3, 4, 5, 6, 7, 8};
if(binary_search(vi.begin(), vi.end(), 3))
    cout << "found it!" << endl;
else
    cout << "not found!" << endl;
binary_search(vi.begin(), vi.begin() + 4, 8) //???
```

Cópia

copy

- Copiar elementos de um container para outro container
- O container de destino deverá ter alocado espaço suficiente

```
std::vector<int> vect_orig = { 1, 2, 3, 4, 5 };
std::vector<int> vect_dest(vect_orig.size());

std::copy(vect_orig.begin(), vect_orig().end,
          vect_dest().begin);
```

Ordenação

sort

- Ordenar de modo crescente os elementos do range [first, last[
- Por omissão, é usado o operador <
- Pode ser indicada outra função de comparação

```
vector<int> v = {5, 2, 3, 1, 4};
// by default elements are sorted in ascending order
sort(v.begin(), v.end());
// sorting in descending order using the greater func.
sort(v.begin(), v.end(), greater());
```

Transformação

transform

```
// Transform and the use of a lambda function

string city = "aveiro";

transform(city.begin(), city.end(), city.begin(),
                  [](char c) {return toupper(c);});

cout << "city: " << city << endl;                    //AVEIRO
```