**Map trong C++ là gì**

21 tháng 1 ,2022

Cùng tìm hiểu về kiểu map trong C++. Bạn sẽ biết khái niệm **map trong c++ là gì**, cách **khai báo map trong C++**, cách **khởi tạo map trong C++**, cách **truy cập phần tử của map**, cũng như **sự khác biệt giữa set và map trong C++** sau bài học này.

**Map trong c++ là gì**

**Map trong C++** là một **tập hợp các phần tử được sắp xếp theo thứ tự cụ thể, mà mỗi phần tử trong đó được hình thành bởi sự kết hợp của một cặp khóa và giá trị (key & value), với mỗi khóa là duy nhất trong map**.

Trong map, các **khóa** (key) được sử dụng để sắp xếp và xác định **giá trị** (value) tương ứng được liên kết với nó. Mỗi khóa trong map là duy nhất và không được phép trùng lặp. Các giá trị trong map thì có thể trùng lặp, chúng có thể thay đổi giá trị, cũng như là được chèn hoặc xóa khỏi map.

Ví dụ cụ thể, một map chứa thông tin về một người sẽ gồm các khóa như name, old, weight. Tương ứng với các khóa này là các giá trị là thông tin cụ thể của người đó như sau:

Map trong c++ là gì

Về mặt nội bộ, các phần tử trong map luôn được sắp xếp theo khóa của nó theo thứ tự cụ thể một cách nghiêm ngặt, được chỉ ra bởi đối tượng so sánh nội bộ của map. Nếu bạn thêm các phần tử mới không theo thứ tự cụ thể vào một map, chúng sẽ tự động sắp xếp lại theo khóa trước khi được lưu trữ nội bộ.

Trong C++ cũng có một loại dữ liệu khá giống với map là **unordered\_map** khi các phần tử cũng được hình thành bởi các cặp khóa và giá trị. Tuy nhiên thì khác với unordered\_map có các phần tử không được sắp xếp, thì phần tử trong một map **luôn luôn được sắp xếp** theo khóa của chúng.

Về mặt tốc độ xử lý thì map có khả năng tìm kiếm dữ liệu theo khóa với tốc độ cực khá cao là O(log N). Tuy nhiên thì nó vẫn kém unordered\_map với tốc độ còn cao hơn là O(1), do unordered\_map sử dụng bảng hash.

Tuy nhiên thì map có lợi thế khi các phần tử của nó được sắp xếp sẵn, nên chúng ta nên dùng map trong các trường hợp cần ưu tiên tính chất này.

**Cấu trúc dữ liệu map trong c++**

https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/map-trong-cpp/map-trong-cpp-la-gi/

Cấu trúc dữ liệu map trong C++ thuộc dạng [Red–black tree (cây đỏ đen)](https://vi.wikipedia.org/wiki/C%C3%A2y_%C4%91%E1%BB%8F_%C4%91en) - một cây nhị phân, là một cấu trúc dữ liệu trong khoa học máy tính để tổ chức các thành phần dữ liệu có thể so sánh.

Cụ thể thì cấu trúc dữ liệu map trong C++ có được thể hiện như ví dụ dưới đây. Lưu ý là cấu trúc này có thể khác một chút so với thực tế cấu trúc trong môi trường máy của bạn.

Cấu trúc dữ liệu của map trong C++

Chúng ta cần đặc biệt lưu ý 3 điểm sau đây về **Cấu trúc dữ liệu map trong c++**:

1. Trong các Node sẽ lưu giữ cặp khóa:giá trị (key & value) cũng như con trỏ của các Node con (trái, phải) của nó.
2. Các giá trị trong Node thỏa mãn điều kiện **giá trị của Node con bên trái < Giá trị Node cha < Giá trị của Node con bên phải**. Do trong map không cho phép các khóa trùng nhau nên dấu < được sử dụng.
3. Độ sâu của các Node bằng nhau và cây Node thì cân bằng.

Nhờ vào cấu trúc dữ liệu kiểu này mà chúng ta có thể [tìm kiếm nhị phân](https://vi.wikipedia.org/wiki/T%C3%ACm_ki%E1%BA%BFm_nh%E1%BB%8B_ph%C3%A2n) trong map, qua đó có thể tìm kiếm trong map với tốc độ cao O(Log N).

**std::map trong C++**

**std::map trong C++** là một thư viện chuẩn được sử dụng để xử lý map trong C++.

**std::map** được cài sẵn trong header file **map** và để sử dụng được chức năng này, chúng ta cần thêm dòng #include <map> vào đầu chương trình.

COPY

|  |
| --- |
| #include <map>  int main() {  std::map<std::string, int> mp;   std::map<char, double> mp2;  } |

Lại nữa, namespace của **std::map** là std, do đó bằng cách khai báo sử dụng namespace này vào đầu chương trình mà chúng ta có thể viết gọn **std::map** trong chương trình như sau:

COPY

|  |
| --- |
| #include <map>  using namespace std; int main() {  map<std::string, int> mp;   map<std::char, double> mp2;  } |

**Khai báo map trong C++**

**Khai báo 1 map trong C++**

Để khai báo map trong C++, chúng ta viết dòng *std::map*, sau đó viết kiểu dữ liệu của key và value cách nhau bởi dấu phẩy ở giữa cặp dấu <>, và cuối cùng là tên biến map như sau:

std::map<k\_type, v\_type> mp;

Trong đó mp là tên biến map và k\_type, v\_type lần lượt là kiểu dữ liệu của key và value. Cách viết sử dụng cặp dấu <> như trên được viết theo cú pháp khi sử dụng chức năng *template* của C++ mà chúng ta sẽ cùng học trong các chuyên đề sau.

Chúng ta có thể dùng bất cứ kiểu dữ liệu nào có trong C++ để khai báo type, ví dụ như char, int, double, hay cấu trúc hoặc class tự tạo chẳng hạn.

Lưu ý, mặc dù chúng ta có thể dùng bất cứ kiểu dữ liệu nào có trong C++ để khai báo type, tuy nhiên do trong map các phần tử cần phải được sắp xếp, nên kiểu của chúng cũng phải là kiểu dữ liệu **có thể được so sánh**.

Đối với các kiểu dữ liệu nguyên thủy như char, int, double chẳng hạn thì chúng vốn có thể tự so sánh được, nhưng nếu chúng ta sử dụng các kiểu dữ liệu không phải là kiểu dữ liệu nguyên thủy, ví dụ như cấu trúc hoặc class tự tạo chẳng hạn, thì bắt buộc phải tự định nghĩa toán tử so sánh nội bộ **operator<()** để làm rõ quan hệ lớn nhỏ giữa chúng.

Lại nữa, trong trường hợp đã khai báo namespace std vào đầu chương trình, chúng ta cũng có thể lược bỏ dòng std:: và dùng cú pháp khai báo map như sau:

using namespace std;  
map<k\_type, v\_type> mp;

Thông thường chúng ta hay khai báo namespace std vào đầu chương trình để sử dụng tới các chức năng thông dụng khác như nhập xuất chẳng hạn, nên trong 2 phương pháp khai báo map ở trên thì phương pháp thứ 2 thường được sử dụng nhiều hơn.

Lưu ý map được khai báo với cú pháp này sẽ có 0 phần tử bên trong nó. Sau khi khai báo map kiểu này, chúng ta có thể sử dụng các hàm thành viên để có thể thêm phần tử vào nó sau này.

Ví dụ cụ thể, chúng ta khai báo 1 map thuộc kiểu dữ liệu nguyên thủy như sau:

COPY

|  |
| --- |
| #include <iostream> #include <map>  using namespace std;  int main() {  map<double, string> user;   map<int, char> info;  } |

Còn khi khai báo 1 map không thuộc kiểu dữ liệu nguyên thủy, ví dụ như struct chẳng hạn thì chúng ta phải tự tạo ra toán tử so sánh nội bộ **operator<()** để làm rõ quan hệ lớn nhỏ giữa các phần tử như ví dụ sau:

COPY

|  |
| --- |
| struct Person {  string m\_name;  int m\_height; }; // Định nghĩa toán tử so sánh nội bộ của struct bool operator<(const Person &lhs, const Person &rhs) {  return lhs.m\_name < rhs.m\_name; }  /\*Khai báo map thuộc kiểu struct\*/ std::map<Person, int> mp; |

**Khai báo đồng thời nhiều map trong C++**

Trong trường hợp cần khai báo đồng thời nhiều map trong C++, chúng ta viết các tên các biến cách nhau bởi dấu phẩy vào đằng sau std::map với cú pháp sau đây:

using namespace std;  
map<k\_type, v\_type> name1, name2, name3, ... ;

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map>  using namespace std;  int main() {  map<string, int> name, job, sex;  map<int, char> age; } |

**Gán giá trị cho map trong C++**

Sau khi đã khai báo map, chúng ta có thể gán giá trị các phần tử vào nó bằng cách sử dụng toán tử [] với cú pháp sau đây:

mp[key] = value;

Trong đó

* mp là tên biến map
* key và value là khóa và giá trị của phần tử cần gán vào map

Ví dụ:

COPY

|  |
| --- |
| std::map<std::string, int> mp;  mp["Kiyoshi"] = 1; // {"Kiyoshi", 1} mp["Honda"] = 2; // {"Honda", 2} mp["Suzuki"] = 3; // {"Suzuki", 3} |

Lại nữa, giá trị của phần tử có thể được ghi đè và thay thế bởi một giá trị mới nhiều lần như sau:

COPY

|  |
| --- |
| mp["Kiyoshi"] = 4; // {"Kiyoshi", 4} mp["Kiyoshi"] = 8; // {"Kiyoshi", 8} mp["Kiyoshi"] = 88;// {"Kiyoshi", 88} |

Trong trường hợp thay đổi giá trị nhiều lần, thì giá trị trong lượt thay đổi giá trị cuối cùng sẽ được sử dụng làm giá trị phần tử.

**Khởi tạo map trong C++**

Ngoài cách khai báo rồi gán giá trị cho map sau đó thì chúng ta cũng có thể khởi tạo map và gán luôn giá trị ban đầu cho map.

Chúng ta **khởi tạo map trong C++** cách sử dụng cặp dấu ngoặc {} với cú pháp sau đây:

std::map<k\_type, v\_type> mp = { {k1, v1}, {k2, v2}, {k3, v3}, ...};

Trong đó

* mp là tên biến map
* k\_type, v\_type là kiểu dữ liệu của key và value
* k,v là các cặp key và value

Lưu ý do mỗi khóa trong map là duy nhất, nên nếu chúng ta chỉ định các phần tử có cùng khóa thì dù giá trị của chúng có giống hay khác nhau thì chỉ có duy nhất phần tử viết đầu tiên sẽ được lưu vào trong map mà thôi.

Ví dụ:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  int main () {  map<string,int> mp = {  { "alpha", 10 },  { "beta", 20 },  { "gamma", 30 }   };   for (auto& x: mp) {  std::cout << x.first << ": " << x.second << '\n';  }  return 0; } |

Kết quả:

COPY

|  |
| --- |
| alpha: 10 beta: 20 gamma: 30 |

Trong trường hợp chỉ định các phần tử có khóa giống nhau, bất kể giá trị của chúng có giống hay khác nhau thì chỉ có duy nhất phần tử viết đầu tiên được lưu vào map như sau:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  int main () {  map<string,int> mp = {  { "alpha", 20 },  { "beta", 20 },  { "alpha", 10 },  { "gamma", 30 },  { "alpha", 10 },  };   for (auto x: mp) {  cout << x.first << ": " << x.second << endl;  }     return 0; } // alpha: 20 // beta: 20 // gamma: 30 |

**Truy cập phần tử trong map C++**

Để truy cập phần tử trong map C++, chúng ta sử dụng toán tử [] hoặc hàm thành viên at() như sau:

**Truy cập phần tử trong map C++ bằng toán tử []**

Để truy cập phần tử trong map C++, chúng ta viết key của phần tử cần truy cập vào giữa cặp toán tử [] với cú pháp sau đây:

mp[key]

Trong đó

* mp là tên biến map
* key là khóa của phần tử cần truy cập trong map

Nếu như key tồn tại trong map, giá trị tương ứng của key sẽ được trả về. Tuy nhiên nếu không tồn tại, giá trị 0 sẽ được trả về

Ví dụ, chúng ta có thể truy cập và xuất giá trị của một phần tử trong map như sau:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  int main () {  /\*Khai báo và gán giá trị\*/  std::map<std::string, int> mp;   mp["Kiyoshi"] = 1;   /\*Truy cập vào phần tử vừa gán bằng key\*/  cout << mp["Kiyoshi"] << //1    /\*Truy cập vào phần tử không tồn tại trong map\*/  cout << mp["Honda"]; //0    return 0; } |

Ngoài việc xuất giá trị, chúng ta cũng có thể thay đổi giá trị của một phần tử trong map với cách này. Ví dụ:

COPY

|  |
| --- |
| /\*Khai báo và gán giá trị\*/ std::map<std::string, int> mp;  mp["Kiyoshi"] = 1;  /\*Truy cập và thay đổi giá trị\*/ mp["Kiyoshi"] = 2; |

**Truy cập phần tử trong map C++ bằng hàm at()**

**Hàm at() trong C++** là một hàm thành viên trong class std:map, có tác dụng **truy cập vào phần tử trong map thông qua key của nó**.

Nếu như key tồn tại trong map, giá trị tương ứng của key sẽ được trả về. Tuy nhiên nếu không tồn tại, hàm at() sẽ trả về lỗi *out\_of\_range*.

Cú pháp truy cập phần tử trong map C++ bằng hàm at() trong C++ như sau:

mp.at(key);

Trong đó mp là tên map và key là khóa của phần tử cần truy cập.

Ví dụ cụ thể:

COPY

|  |
| --- |
| /\*Khai báo và gán giá trị\*/ std::map<std::string, int> mp;  mp["Kiyoshi"] = 1;   /\*Truy cập vào phần tử vừa gán bằng key\*/ cout << mp.at("Kiyoshi"); //1  /\*Truy cập vào phần tử không tồn tại trong map\*/ cout << mp.at("Honda"); // throwing an instance of 'std::out\_of\_range |

Có thể thấy rõ sự khác biệt giữa cách truy cập phần tử trong map bằng hàm at() và toán tử [] chính là ở kết quả khi chỉ định một key không tồn tại trong map.

**set và map trong c++**

Giống với map thì [kiểu set trong C++](https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/set-trong-cpp/set-trong-cpp-la-gi) cũng chứa các phần tử duy nhất được sắp xếp theo thứ tự trong nó. Và cấu trúc dữ liệu của 2 kiểu này cũng đều thuộc dạng cây nhị phân [Red–black tree (cây đỏ đen)](https://vi.wikipedia.org/wiki/C%C3%A2y_%C4%91%E1%BB%8F_%C4%91en).

Tuy vậy thì giữa chúng có 3 điểm khác biệt rất rõ ràng như sau:

1. Phần tử trong set tạo bởi 1 giá trị, trong khi trong map tạo bởi một cặp khóa và giá trị.
2. Phần tử trong set được sắp xếp theo giá trị của chúng, còn trong map thì theo khóa của chúng
3. Vùng bộ nhớ chứa set sẽ nhỏ hơn so với một map có cùng số phần tử, do trong map ngoài giá trị thì còn lưu trữ cả key nữa

Do vậy chúng ta có thể sử dụng một trong 2 kiểu này để xử lý các đối tượng mà cần có sự sắp xếp phần tử trong nó, tùy theo nhu cầu của mình.

**Tổng kết**

Trên đây Kiyoshi đã hướng dẫn bạn về **map trong C++** rồi. Để nắm rõ nội dung bài học hơn, bạn hãy thực hành viết lại các ví dụ của ngày hôm nay nhé.

**Duyệt map trong C++**

21 tháng 1 ,2022

Hướng dẫn cách duyệt map trong C++. Bạn sẽ học được 2 cách căn bản để duyệt map trong C++ sau bài học này.

Trong bài [map trong C++](https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/map-trong-cpp/map-trong-cpp-la-gi) chúng ta đã biết các phần tử trong map được xác định thông qua key của chúng chứ không phải là bằng index như các containers khác trong C++ như list vector. Do đó chúng ta cũng không thể sử dụng index của các phần tử để truy cập vào phần tử của map theo cách thông thường được.

Thay vào đó thì chúng ta có 2 phương pháp duyệt map trong C++ như sau:

* Sử dụng vòng lặp dựa trên phạm vi
* Sử dụng iterator

**Duyệt map trong C++ bằng vòng lặp dựa trên phạm vi**

https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/map-trong-cpp/duyet-map-trong-cpp/

Có rất nhiều thứ tiện lợi đã được thêm vào trong C++, và một trong số đó chính là **vòng lặp dựa trên phạm vi**.

Bằng cách sử dụng vòng lặp dựa trên phạm vi, chúng ta có thể **duyệt map trong C++** với cú pháp như sau:

for ( auto& x : mp) {  
    cout << x.first << “: “ << x.second << endl;  
}

Trong đó:

* mp là tên map.
* auto là kiểu suy luận giúp tự xác định kiểu dữ liệu của giá trị lấy từ map.
* x là tên một biến dùng để gán từng phần tử được lấy từ map.
* x.first và x.second lần lượt được sử dụng để lấy key và value của phần tử

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map>  using namespace std;  int main() {  map<string,int> mp = {  { "alpha", 10 },  { "beta", 20 },  { "gamma", 30 } };   for (auto x: mp) {  cout << x.first << ": " << x.second << endl;  } } |

Kết quả:

COPY

|  |
| --- |
| alpha: 10 beta: 20 gamma: 30 |

**Duyệt map trong C++ bằng iterator**

Trong C++, các kiểu dữ liệu như vector, map, map đều được thêm một chức năng là **iterator** nhằm giúp biến chúng thành các trình lặp để dễ dàng xử lý.

Bằng cách sử dụng iterator, chúng ta có thể **duyệt map trong C++** với cú pháp như sau:

for(auto itr = mp.begin(); itr != mp.end(); ++itr) {  
    cout << itr->first << ": "<< itr->second << "\n";  
}

Trong đó:

* mp là tên map
* itr là tên iterator dùng để trỏ đến phần tử
* itr->first và itr->second lần lượt được sử dụng để lấy key và value của phần tử

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map>  using namespace std;  int main() {  map<string,int> mp = {  { "alpha", 10 },  { "beta", 20 },  { "gamma", 30 } };   for(auto itr = mp.begin(); itr != mp.end(); ++itr) {  cout << itr->first << ": "<< itr->second << "\n";  } } |

Và kết quả thu về cũng tương tự với phương pháp duyệt map bằng vòng lặp dựa trên phạm vi ở trên:

COPY

|  |
| --- |
| alpha: 10 beta: 20 gamma: 30 |

**Tổng kết**

Trên đây Kiyoshi đã hướng dẫn bạn về cách **duyệt map trong C++** rồi. Để nắm rõ nội dung bài học hơn, bạn hãy thực hành viết lại các ví dụ của ngày hôm nay nhé.

**Lấy kích thước map trong C++ bằng hàm size**

https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/map-trong-cpp/lay-kich-thuoc-map-trong-cpp/

**Hàm size** là một hàm thành viên trong class std:map, có tác dụng **lấy kích thước (số phần tử) có trong 1 map C++**.

Chúng ta sử dụng hàm size để lấy kích thước map trong C++ với cú pháp như sau:

mp.size();

Trong đó mp là map cần lấy kích thước (số phần tử) chứa trong nó.

Ví dụ cụ thể, chúng ta **lấy kích thước map trong C++** bằng hàm size như sau:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  int main () {  map<string,int> mp = {  { "alpha", 10 },  { "beta", 20 },  { "gamma", 30 } };   cout<< mp.size() <<endl;  return 0; } //3 |

Lưu ý, số phần tử hay kích thước của map ở đây được tính sau khi các phần tử được kiểm tra key có trùng lặp và sắp xếp trong map, chứ không phải là số phần tử mà chúng ta đã dùng khi khai báo map.

Ví dụ, nếu khi khai báo map mà tồn tại các phần tử trùng lặp key thì số phần tử được đếm bởi hàm size() sẽ khác với số phần tử chỉ định trong khai báo như sau:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  int main () {  map<string,int> mp = {  { "alpha", 20 },  { "beta", 20 },  { "alpha", 10 },  { "gamma", 30 },  { "alpha", 10 },  };  for (auto x: mp) {  cout << x.first << ": " << x.second << endl;  }  cout<< mp.size() <<endl;     return 0; } |

Kết quả:

COPY

|  |
| --- |
| alpha: 20 beta: 20 gamma: 30 3 |

Có thể thấy mặc dù có 5 phần tử được dùng trong khai báo map nhưng chỉ có 3 phần tử với key không trùng lặp được lưu trong map, dẫn đến kết quả của hàm size() bằng 3 mà thôi.

**Tổng kết**

Trên đây Kiyoshi đã hướng dẫn bạn về cách **lấy kích thước của map trong C++** rồi. Để nắm rõ nội dung bài học hơn, bạn hãy thực hành viết lại các ví dụ của ngày hôm nay nhé.

**map trống trong C++ (clear, empty)**

21 tháng 1 ,2022

Cùng tìm hiểu về map trống trong C++. Bạn sẽ học được cách sử dụng các hàm clear() và empty() để kiểm tra một map có phải là map trống, cũng như là cách làm trống một map trong C++ sau bài học này.

**Kiểm tra map trống trong C++ bằng hàm empty**

https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/map-trong-cpp/map-trong-trong-cpp/

**Hàm empty** là một hàm thành viên trong class std:map, có tác dụng **kiểm tra một map có phải là map trống hay không** trong C++.

Cú pháp của hàm empty trong C++ như sau:

mp.empty();

Trong đó mp là map cần kiểm tra.

Hàm empty sẽ trả về true nếu map đã cho là map trống, cũng trả về false, nếu map đã cho có chứa phần tử.

Ví dụ cụ thể, chúng ta **kiểm tra map trống trong C++** bằng hàm empty như sau:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  int main () {  map<int, char> mp;  if( mp.empty() )  cout << "empty.\n";  else  cout << "not empty.\n";   map<string,int> mp2 = {  { "alpha", 10 },  { "beta", 20 },  { "gamma", 30 } };  if( mp2.empty() )  cout << "empty.\n";  else  cout << "not empty.\n";    return 0; } |

Kết quả:

COPY

|  |
| --- |
| empty. not empty. |

**Làm trống 1 map trong C++ bằng hàm clear**

**Hàm clear** là một hàm thành viên trong class std:map, có tác dụng **làm trống một map** trong C++.

Cú pháp của hàm clear trong C++ như sau:

mp.clear();

Trong đó mp là map cần làm trống.

Khác với vector thì hàm map clear ngoài việc làm trống map chỉ định (xóa đi tất cả phần tử) thì còn giải phóng bộ nhớ sử dụng cho việc lưu trữ dữ liệu đã dùng.

Ví dụ cụ thể, chúng ta **làm trống 1 map trong C++** bằng hàm clear như sau:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  int main () {  map<char,int> mymap;   mymap['x']=100;  mymap['y']=200;  mymap['z']=300;   cout << mymap.size() << "\n"; //3   mymap.clear();  cout << mymap.size() << "\n"; //0  } |

**Tổng kết**

Trên đây Kiyoshi đã hướng dẫn bạn về **map trống trong C++** rồi. Để nắm rõ nội dung bài học hơn, bạn hãy thực hành viết lại các ví dụ của ngày hôm nay nhé.

**Sao chép và hoán đổi map trong C++ (swap)**

21 tháng 1 ,2022

Hướng dẫn cách sao chép và hoán đổi map trong C++ Bạn sẽ học được cách sao chép một map vào một map khác, cũng như cách hoán đổi 2 map cho nhau trong C++ sau bài học này.

**Sao chép map trong C++**

map trong C++ thuộc kiểu dữ liệu đối tượng, do vậy khác với các kiểu dữ liệu nguyên thủy, chúng ta không thể sử dụng toán tử bằng = để gán và sao chép một map vào một map mới.

Thay vào đó, chúng ta sẽ sử dụng cách copy constructor trong map với cú pháp như sau:

std::map<k\_type, v\_type> mp\_des( mp\_src );

Trong đó type là kiểu dữ liệu, st\_src là map nguồn để copy và st\_dest là map đích dùng để dán kết quả sao chép.

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map>  using namespace std;  //Tạo hàm xuất map void dump(map<string,int>& mp) {  for (auto x: mp) {  cout << x.first << ": " << x.second << " ";  }  cout << endl; }  int main() {  map<string,int> mp\_src = {  { "alpha", 10 },  { "beta", 20 },  { "gamma", 30 } };   cout << "Orginary map" << endl;  dump (mp\_src);   /\*Sao chép map\*/  map<string,int> mp\_des(mp\_src);   cout << "Copy map" << endl;  dump (mp\_des);  return 0; } |

Kết quả:

COPY

|  |
| --- |
| Orginary map alpha: 10 beta: 20 gamma: 30  Copy map alpha: 10 beta: 20 gamma: 30 |

**Hoán đổi 2 map trong C++**

https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/map-trong-cpp/sao-chep-va-hoan-doi-map-trong-cpp/

**map swap trong C++** là một hàm thành viên trong class std:map, có tác dụng **hoán đổi 2 map trong C++**.

Hàm swap sẽ hoán đổi toàn bộ nội dung của 2 map đã cho cho nhau và làm thay đổi nội dung cũng như độ dài của chúng.

Cú pháp hàm map swap để hoán đổi 2 map trong C++ như sau:

st1.swap(st2);

Trong đó st1 và st2 là 2 map cần hoán đổi nội dung cho nhau.

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map>  using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ": " << x.second << " ";  }  cout << endl; }  int main() {   /\*Khai báo và gán giá trị cho foo và bar\*/  map<char,int> foo,bar;   foo['x']=100;  foo['y']=200;   bar['a']=11;  bar['b']=22;  bar['c']=33;       cout << "Before swap" << endl;  dump(foo);  dump(bar);     /\*Hoán đổi for cho bar\*/  foo.swap(bar);    cout << "After swap" << endl;  dump(foo);  dump(bar); } |

Kết quả:

COPY

|  |
| --- |
| Before swap x: 100 y: 200  a: 11 b: 22 c: 33  After swap a: 11 b: 22 c: 33  x: 100 y: 200 |

Ngoài cách dùng hàm map swap, chúng ta cũng có thể dùng function template là std::swap để tiến hành hoán đổi 2 map với nhau, cũng như là để hoán đổi các đối tượng khác như map, vector trong C++.

Lưu ý chúng ta cần phải thêm header file utility vào trong chương trình để có thể sử dụng được function template này

Ví dụ, chúng ta dùng std::swap để hoán đổi 2 map trong C++ như sau:

COPY

|  |
| --- |
| #include <iostream> #include <utility> #include <map>  using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ": " << x.second << " ";  }  cout << endl; }  int main() {  map<char,int> foo,bar;   foo['x']=100;  foo['y']=200;   bar['a']=11;  bar['b']=22;  bar['c']=33;       cout << "Before swap" << endl;  dump(foo);  dump(bar);     swap(foo,bar);    cout << "After swap" << endl;  dump(foo);  dump(bar); } |

Và chúng ta thu về kết quả hoán đổi tương tự:

COPY

|  |
| --- |
| Before swap x: 100 y: 200  a: 11 b: 22 c: 33  After swap a: 11 b: 22 c: 33  x: 100 y: 200 |

**Tổng kết**

Trên đây Kiyoshi đã hướng dẫn bạn về cách **sao chép và hoán đổi map trong C++** rồi. Để nắm rõ nội dung bài học hơn, bạn hãy thực hành viết lại các ví dụ của ngày hôm nay nhé.

**Thêm chèn phần tử vào map trong C++**

21 tháng 1 ,2022

Hướng dẫn cách **thêm chèn phần tử vào map trong C++**. Bạn sẽ học được cách dùng toán tử [] cũng hàm insert và hàm emplace để chèn thêm phần tử vào map trong C++ sau bài học này.

Để thêm chèn phần tử vào map trong C++, chúng ta sử dụng tới toán tử [] hoặc 2 hàm thành viên là insert() và emplace(). Do khi chèn phần tử vào map thì vị trí chèn sẽ được tự động quyết định tùy thuộc vào bộ sắp xếp trong map, nên lưu ý khác với các containers khác như list hay vector thì trong map không tồn tại</> các hàm push\_back() hay push\_front() để thêm phần tử vào đầu hay cuối map.

**Thêm 1 phần tử vào map trong C++ bằng toán tử []**

https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/map-trong-cpp/them-chen-map-trong-cpp/

Để thêm 1 phần tử vào map trong C++, chúng ta sử dụng toán tử [] với cú pháp sau đây:

mp[key] = value;

Trong đó

* mp là tên biến map
* key và value là khóa và giá trị của phần tử cần thêm vào map

Ví dụ:

COPY

|  |
| --- |
| map<char,int> foo,bar;  foo['x']=100; //() foo['y']=200; |

Lưu ý do mỗi key trong map đều là duy nhất nên nếu chúng ta thêm một phần tử mới vào map nhưng lại có key trùng với một phần tử đã tồn tại trước đó, thì giá trị của phần tử mới này sẽ được dùng để ghi đè lên phần tử trước đó.

Ví dụ:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ":" << x.second << " ";  }  cout << endl; }  int main () {  map<char,int> mymap;   mymap['x']=100;  mymap['y']=200;  mymap['z']=300;    dump(mymap); // x:100 y:200 z:300     mymap['z']=888;  dump(mymap); // x:100 y:200 z:888  } |

**Chèn 1 phần tử vào map trong C++ bằng hàm insert**

**Hàm insert** là một hàm thành viên trong class std:map, có tác dụng **chèn một hoặc nhiều phần tử vào map** cũng như tăng độ dài tương ứng của nó.

Để chèn 1 phần tử vào trong map C++, chúng ta sử dụng hàm insert với cú pháp sau đây:

mp.insert(std::pair<k\_type,x\_type>(k,v));

Trong đó

* mp là map ban đầu
* pair<k\_type,x\_type>(k,v) sử dụng để chỉ định key và value của phần tử cần thêm, trong đó k\_type,x\_type là kiểu và k,v là key và value.

Trong trường hợp không rõ kiểu, hoặc muốn rút bỏ chỉ định kiểu của key và value, chúng ta có thể dùng hàm make\_pair() để thay thế cho pair() trong hàm insert, với cú pháp sau đây:

mp.insert(std::make\_pair(k,v));

Hàm map insert sẽ trả về một cặp kết quả pair<iterator, bool> với iterator là trình lặp trỏ đến map kết quả, và bool là việc có thực hiện việc chèn hay không, dưới dạng 0 hoặc 1.

Bởi vì các phần tử trong một map là duy nhất, nên thao tác chèn sẽ kiểm tra xem mỗi phần tử được chèn đã tồn tại trong map hay chưa. Nếu chưa tồn tại thì phần tử đó sẽ được chèn và ngược lại nếu đã tồn tại thì không được chèn.

Đây là điểm khác biệt so với cách dùng toán tử [], vì khi một key đã tồn tại trong map, nếu chúng ta dùng toán tử [] thì giá trị mới sẽ được ghi đè, còn nếu dùng hàm insert thì việc chèn sẽ thất bại.

Trong trường hợp tất cả các phần tử chèn vào vốn đã tồn tại từ trước trong map, thì bản thân map ban đầu sẽ được trả về.

Lại nữa, phần tử trong map được sắp xếp theo thứ tự cụ thể trước khi được lưu, nên các phần tử cần chèn thêm vào map sẽ được tự động quyết định vị trí dựa trên bộ sắp xếp, và chúng ta sẽ không quyết định được vị trí cần chèn của phần tử.

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ":" << x.second << " ";  }  cout << endl; }   int main () {  /\*Khai báo và gán giá trị cho foo và bar\*/  map<char,int> mp;   mp['a']=100;  mp['b']=200;    // duyệt map ban đầu  dump(mp);    //chèn phần tử với key chưa tồn tại trong map  mp.insert(pair<char,int>('c',300));  dump(mp);   //chèn phần tử với key đã tồn tại trong map  mp.insert(pair<char,int>('a',3));  dump(mp);   //chèn phần tử bằng insert và make\_pair  mp.insert(make\_pair('x',50));  dump(mp);     return 0; } |

Kết quả:

COPY

|  |
| --- |
| a:100 b:200  a:100 b:200 c:300  a:100 b:200 c:300 // Do key đã tồn tại nên sẽ không được chèn  a:100 b:200 c:300 x:50 |

Chúng ta cũng có thể kiểm tra việc chèn phần tử đã thực hiện hay chưa bằng phương thức first() hoặc second() từ kết quả trả về của hàm như sau:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  int main () {  /\*Khai báo và gán giá trị cho foo và bar\*/  map<char,int> mp;   mp['a']=100;  mp['b']=200;    //chèn phần tử với key chưa tồn tại trong map  auto r = mp.insert(pair<char,int>('z',300));  cout << r.second << "\n"; // Trả về 1 do được chèn   //chèn phần tử với key tồn tại trong map,   r = mp.insert(pair<char,int>('a',3));  cout << r.second << "\n"; // Trả về 0 do không được chèn     return 0; } |

**Thêm chèn 1 phần tử vào map trong C++ bằng hàm emplace**

**Hàm emplace** là một hàm thành viên trong class std:map, có tác dụng **chèn một hoặc nhiều phần tử vào map** thông qua hàm tạo của kiểu dữ liệu, cũng như tăng độ dài tương ứng của map.

Để chèn thêm 1 phần tử vào trong map C++, chúng ta sử dụng hàm emplace với cú pháp sau đây:

mp.emplace(k,v);

Trong đó

* mp là map ban đầu
* k,v là key và value.

Hàm map emplace sẽ thực hiện việc chèn phần tử nếu key của nó chưa tồn tại trong map. Ngược lại nếu key đó đã tòn tại, việc chèn thất bại và false sẽ được trả về

Và Hàm map emplace sẽ trả về một trình lặp trỏ tới vị trí phần tử vừa được chèn vào, nếu việc chèn thành công.

Lại nữa, phần tử trong map được sắp xếp theo thứ tự cụ thể trước khi được lưu, nên các phần tử cần chèn thêm vào map sẽ được tự động quyết định vị trí dựa trên bộ sắp xếp, và chúng ta sẽ không quyết định được vị trí cần chèn của phần tử.

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ":" << x.second << " ";  }  cout << endl; }  int main () {  std::map<char,int> mymap;   mymap.emplace('x',100);  mymap.emplace('y',200);  mymap.emplace('z',100);    dump(mymap);  return 0; } |

Kết quả:

COPY

|  |
| --- |
| x:100 y:200 z:100 |

**Chèn nhiều phần tử vào map trong C++ bằng hàm insert**

Để chèn nhiều phần tử vào trong map C++, chúng ta sử dụng hàm insert với cú pháp sau đây:

mp.insert(iterator\_first, iterator\_last);

Trong đó

* mp là map ban đầu
* iterator\_first và iterator\_last là các trình lặp xác định phạm vi chứa các phần tử cần chèn ở trong một map khác vào map ban đầu.

Lưu ý ở đây, các phần tử cần chèn được lấy ra từ một phạm vi trong một map khác, và phạm vi này được chỉ định thông qua 2 trình lặp như trên

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<int, const char\*>& mp) {  for (auto x: mp) {  cout << x.first << ": " << x.second << " ";  }  cout << endl; }   int main () {  /\* Khởi tạo map mp1\*/  map<int, const char\*> mp1 =  {  {1, "a"},  {2, "b"},  {3, "c"},  {4, "d"},  {5, "e"},  };  // duyệt map mp1  dump(mp1);    /\*Khai báo map mp2\*/  map<int, const char\*> mp2;   /\*Chèn các phần tử từ đầu tới phần tử có key bằng 4  từ mp1 vào mp2\*/  mp2.insert(mp1.begin(),mp1.find(4));    // duyệt map mp2  dump(mp2);  return 0; } |

Và kết quả:

COPY

|  |
| --- |
| 1: a 2: b 3: c 4: d 5: e  1: a 2: b 3: c |

**Tổng kết**

Trên đây Kiyoshi đã hướng dẫn bạn về cách **thêm chèn chèn phần tử vào map trong C++** rồi. Để nắm rõ nội dung bài học hơn, bạn hãy thực hành viết lại các ví dụ của ngày hôm nay nhé.

**Tìm phần tử trong map C++ (find, lower\_bound, upper\_bound, equal\_range, cout)**

21 tháng 1 ,2022

Hướng dẫn cách **tìm phần tử trong map**. Bạn sẽ học được cách sử dụng các hàm như find, lower\_bound, upper\_bound và equal\_range để tìm phần tử trong map, cũng như hàm cout() để đếm số lần xuất hiện của phần tử đó theo khóa trong map C++ sau bài học này.

**Tìm phần tử trong map C++ bằng hàm find**

**Hàm find** là một hàm thành viên trong class std:map, có tác dụng tìm vị trí phần tử có khóa chỉ định trong map.

Chúng ta sử dụng hàm find trong C++ với cú pháp sau đây:

mp.find(key);

Trong đó key là khóa của phần tử cần tìm trong map mp.

Hàm find() sẽ trả về trình lặp trỏ đến vị trí phần tử, nếu nó tồn tại trong map. Và nếu phần tử đó không tồn tại, hàm sẽ trả về trình lặp trỏ đến vị trí cuối cùng trong map.

Bằng cách ứng dụng hàm find(), chúng ta có thể tìm ra vị trí của phần tử đó trong map, rồi kết hợp với hàm clear() để xóa nó đi như ví dụ cụ thể sau đây:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ":" << x.second << " ";  }  cout << endl; }   int main() {  map<char,int> mp;  mp.insert(make\_pair('a', 1));  mp.insert(make\_pair('b', 2));  mp.insert(make\_pair('c', 3));   mp.insert(make\_pair('d', 1));  mp.insert(make\_pair('e', 2));   dump(mp);    //Tìm phần tử có khóa bằng 'c' trong map  auto ret = mp.equal\_range('c');    //Xóa phần tử vừa tìm thấy  mp.erase (ret.first,ret.second);  dump(mp);  return 0; } |

Kết quả:

COPY

|  |
| --- |
| a:1 b:2 c:3 d:1 e:2  a:1 b:2 d:1 e:2 |

**Tìm phần tử trong map C++ bằng hàm equal\_range**

**Hàm equal\_range** là một hàm thành viên trong class std:map, có tác dụng tìm phạm vi của tất cả các phần tử có khóa giống với khóa chỉ định trong map.

Chúng ta sử dụng hàm equal\_range trong C++ với cú pháp sau đây:

mp.equal\_range(key);

Trong đó key là khóa của phần tử cần tìm trong map mp.

Hàm equal\_range() sẽ trả về một cặp giá trị, với giá trị đầu tiên trỏ đến đầu phạm vi, và giá trị thứ hai trỏ đến cuối phạm vi chứa tất cả các phần tử có khóa giống khóa chỉ định.

Tuy nhiên do trong map thì các khóa là duy nhất, do đó nếu khóa tồn tại trong map thì hàm equal\_range sẽ trả về một khoảng chứa giá trị duy nhất đó mà thôi.

Bằng cách ứng dụng hàm equal\_range(), chúng ta có thể tìm ra phạm vi chứa phần tử có khóa chỉ định trong map. Và kết quả thu về sẽ tương ứng với các giá trị nếu được tính bằng hàm lower\_bound() và upper\_bound mà Kiyoshi sẽ giới thiệu ở phần dưới.

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ":" << x.second << " ";  }  cout << endl; }   int main() {  map<char,int> mp;  mp.insert(make\_pair('a', 1));  mp.insert(make\_pair('b', 2));  mp.insert(make\_pair('c', 3));   mp.insert(make\_pair('d', 1));  mp.insert(make\_pair('e', 2));   dump(mp);    //Tìm phần tử có khóa bằng 'c' trong map  auto ret = mp.equal\_range('c');    cout << "lower bound points to: ";  cout << ret.first->first << " => " << ret.first->second << '\n';   cout << "upper bound points to: ";  cout << ret.second->first << " => " << ret.second->second << '\n';    return 0; } |

Kết quả:

COPY

|  |
| --- |
| a:1 b:2 c:3 d:1 e:5 e:2 e:8 f:3  a:1 b:2 c:3 d:1 f:3 |

**Tìm phần tử trong map C++ bằng hàm lower\_bound**

https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/map-trong-cpp/tim-va-dem-phan-tu-trong-map-cpp/

**Hàm lower\_bound** là một hàm thành viên trong class std::map, có tác dụng tìm vị trí **phần tử đầu tiên** trong map có khóa lớn hơn hoặc bằng với khóa chỉ định.

Chúng ta sử dụng hàm lower\_bound trong C++ với cú pháp sau đây:

mp.lower\_bound(key);

Trong đó key là khóa của phần tử cần tìm trong map mp.

Hàm lower\_bound() sẽ trả về trình lặp trỏ đến vị trí **phần tử đầu tiên** có khóa lớn hơn hoặc bằng với khóa chỉ định. Và nếu không tìm thấy, hàm sẽ trả về trình lặp trỏ đến vị trí cuối cùng trong map.

Và trong trường hợp chỉ có một phần tử trong map có khóa giống với khóa chỉ định thì hàm lower\_bound sẽ trả về con trỏ chỉ đến phần tử đó.

Bằng cách ứng dụng hàm lower\_bound() kết hợp hàm upper\_bound() mà Kiyoshi sẽ hướng dẫn ở phần dưới, chúng ta có thể tìm ra 2 vị trí rồi xóa các phần tử có trong phạm vi xác định bởi vị trí đó như ví dụ sau:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ":" << x.second << " ";  }  cout << endl; }   int main() {  map<char,int> mp;  mp.insert(make\_pair('a', 1));  mp.insert(make\_pair('b', 2));  mp.insert(make\_pair('c', 3));   mp.insert(make\_pair('d', 1));  mp.insert(make\_pair('e', 2));  mp.insert(make\_pair('f', 3));     dump(mp);     /\*Tìm vị trí phần tử đầu tiên có khóa lớn hơn hoặc bằng 'b' trong map\*/  auto itr1 = mp.lower\_bound('b'); // itr1 trỏ đến b:2   //Tìm vị trí phần tử đầu tiên có khóa lớn hơn 'e' trong map  auto itr2 = mp.upper\_bound('e'); // itr2 trỏ đến f:3    //Xóa các phần tử trong phạm vi [itr1, itr2)  mp.erase (itr1, itr2);  dump(mp);     return 0; } |

Kết quả, các phần tử trong phạm vi từ b:2 đến trước f:3 đã bị xóa đi.

COPY

|  |
| --- |
| a:1 b:2 c:3 d:1 e:2 f:3  a:1 f:3 |

**Tìm phần tử trong map C++ bằng hàm upper\_bound**

Ngược với hàm lower\_bound chính là hàm upper\_bound trong C++.

**Hàm upper\_bound** là một hàm thành viên trong class std::map, có tác dụng tìm vị trí **phần tử đầu tiên** trong map có khóa lớn hơn khóa chỉ định.

Chúng ta sử dụng hàm upper\_bound trong C++ với cú pháp sau đây:

mp.upper\_bound(key);

Trong đó key là khóa của phần tử cần tìm trong map mp.

Hàm lower\_bound() sẽ trả về trình lặp trỏ đến vị trí **phần tử đầu tiên** có khóa lớn hơn khóa chỉ định. Và nếu không tìm thấy, hàm sẽ trả về trình lặp trỏ đến vị trí cuối cùng trong map.

Và trong trường hợp tìm thấy một phần tử trong map có khóa giống với khóa chỉ định thì hàm lower\_bound sẽ trả về con trỏ chỉ đến phần tử tiếp theo phần tử đó.

Bằng cách ứng dụng hàm upper\_bound() kết hợp hàm upper\_bound() mà Kiyoshi đã hướng dẫn ở trên, chúng ta có thể tìm ra 2 vị trí rồi xuất các phần tử có trong phạm vi xác định bởi vị trí đó như ví dụ sau:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std; int main() {  map<char,int> mp;  mp['a']=100;  mp['b']=200;  mp['c']=300;  mp['d']=300;   mp['e']=300;   mp['f']=300;     /\*Duyệt map mp\*/  for (auto x: mp) {  cout << x.first << ":" << x.second << " ";  }  cout << endl; //a:1 b:2 c:3 d:1 e:2     //Tìm phần tử có khóa bằng 'b' trong map  auto itr1 = mp.lower\_bound('b'); // itr1 trỏ đến b:200   //Tìm vị trí phần tử đầu tiên có khóa lớn hơn 'e' trong map  auto itr2 = mp.upper\_bound('e'); // itr2 trỏ đến f:300   //In các phần tử trong phạm vi (itr1, itr2)  for (auto it=itr1; it!=itr2; ++it)  cout << (\*it).first << ":" << (\*it).second << ' ';    return 0; } |

Kết quả:

COPY

|  |
| --- |
| a:100 b:200 c:300 d:300 e:300 f:300  b:200 c:300 d:300 e:300 |

**Đếm số lần xuất hiện của phần tử trong map C++ bằng hàm count**

**Hàm count** là một hàm thành viên trong class std:map, có tác dụng đếm số lần xuất hiện của phần tử trong map C++ thông qua khóa của nó.

Chúng ta sử dụng hàm count trong C++ với cú pháp sau đây:

mp.count(key);

Trong đó key là khóa của phần tử cần đếm số lần xuất hiện trong map mp.

Do trong map các khóa là duy nhất, nên một phần tử nếu tồn tại cũng chỉ có xuất hiện 1 lần duy nhất trong map mà thôi.

Do vậy, kết quả trả về của hàm count() cũng chỉ là 0 tương ứng với phần tử không tồn tại, hoặc 1 tương ứng với phần tử tồn tại trong map đó.

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  int main() {  map<char,int> mp;  mp['a']=100;  mp['b']=200;  mp['c']=300;    //Đếm số lần xuất hiện của phần tử tồn tại trong map  cout << mp.count('b') <<endl;    //Đếm số lần xuất hiện của phần tử không tồn tại trong map  cout << mp.count('f') <<endl;    return 0; } |

Kết quả:

COPY

|  |
| --- |
| 1 0 |

**Tổng kết**

Trên đây Kiyoshi đã hướng dẫn bạn về cách **tìm phần tử map trong C++** rồi. Để nắm rõ nội dung bài học hơn, bạn hãy thực hành viết lại các ví dụ của ngày hôm nay nhé.

**Thêm chèn phần tử vào map trong C++**

21 tháng 1 ,2022

Hướng dẫn cách **thêm chèn phần tử vào map trong C++**. Bạn sẽ học được cách dùng toán tử [] cũng hàm insert và hàm emplace để chèn thêm phần tử vào map trong C++ sau bài học này.

Để thêm chèn phần tử vào map trong C++, chúng ta sử dụng tới toán tử [] hoặc 2 hàm thành viên là insert() và emplace(). Do khi chèn phần tử vào map thì vị trí chèn sẽ được tự động quyết định tùy thuộc vào bộ sắp xếp trong map, nên lưu ý khác với các containers khác như list hay vector thì trong map không tồn tại</> các hàm push\_back() hay push\_front() để thêm phần tử vào đầu hay cuối map.

**Thêm 1 phần tử vào map trong C++ bằng toán tử []**

Để thêm 1 phần tử vào map trong C++, chúng ta sử dụng toán tử [] với cú pháp sau đây:

mp[key] = value;

Trong đó

* mp là tên biến map
* key và value là khóa và giá trị của phần tử cần thêm vào map

Ví dụ:

COPY

|  |
| --- |
| map<char,int> foo,bar;  foo['x']=100; //() foo['y']=200; |

Lưu ý do mỗi key trong map đều là duy nhất nên nếu chúng ta thêm một phần tử mới vào map nhưng lại có key trùng với một phần tử đã tồn tại trước đó, thì giá trị của phần tử mới này sẽ được dùng để ghi đè lên phần tử trước đó.

Ví dụ:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ":" << x.second << " ";  }  cout << endl; }  int main () {  map<char,int> mymap;   mymap['x']=100;  mymap['y']=200;  mymap['z']=300;    dump(mymap); // x:100 y:200 z:300     mymap['z']=888;  dump(mymap); // x:100 y:200 z:888  } |

**Chèn 1 phần tử vào map trong C++ bằng hàm insert**

**Hàm insert** là một hàm thành viên trong class std:map, có tác dụng **chèn một hoặc nhiều phần tử vào map** cũng như tăng độ dài tương ứng của nó.

Để chèn 1 phần tử vào trong map C++, chúng ta sử dụng hàm insert với cú pháp sau đây:

mp.insert(std::pair<k\_type,x\_type>(k,v));

Trong đó

* mp là map ban đầu
* pair<k\_type,x\_type>(k,v) sử dụng để chỉ định key và value của phần tử cần thêm, trong đó k\_type,x\_type là kiểu và k,v là key và value.

Trong trường hợp không rõ kiểu, hoặc muốn rút bỏ chỉ định kiểu của key và value, chúng ta có thể dùng hàm make\_pair() để thay thế cho pair() trong hàm insert, với cú pháp sau đây:

mp.insert(std::make\_pair(k,v));

Hàm map insert sẽ trả về một cặp kết quả pair<iterator, bool> với iterator là trình lặp trỏ đến map kết quả, và bool là việc có thực hiện việc chèn hay không, dưới dạng 0 hoặc 1.

Bởi vì các phần tử trong một map là duy nhất, nên thao tác chèn sẽ kiểm tra xem mỗi phần tử được chèn đã tồn tại trong map hay chưa. Nếu chưa tồn tại thì phần tử đó sẽ được chèn và ngược lại nếu đã tồn tại thì không được chèn.

Đây là điểm khác biệt so với cách dùng toán tử [], vì khi một key đã tồn tại trong map, nếu chúng ta dùng toán tử [] thì giá trị mới sẽ được ghi đè, còn nếu dùng hàm insert thì việc chèn sẽ thất bại.

Trong trường hợp tất cả các phần tử chèn vào vốn đã tồn tại từ trước trong map, thì bản thân map ban đầu sẽ được trả về.

Lại nữa, phần tử trong map được sắp xếp theo thứ tự cụ thể trước khi được lưu, nên các phần tử cần chèn thêm vào map sẽ được tự động quyết định vị trí dựa trên bộ sắp xếp, và chúng ta sẽ không quyết định được vị trí cần chèn của phần tử.

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ":" << x.second << " ";  }  cout << endl; }   int main () {  /\*Khai báo và gán giá trị cho foo và bar\*/  map<char,int> mp;   mp['a']=100;  mp['b']=200;    // duyệt map ban đầu  dump(mp);    //chèn phần tử với key chưa tồn tại trong map  mp.insert(pair<char,int>('c',300));  dump(mp);   //chèn phần tử với key đã tồn tại trong map  mp.insert(pair<char,int>('a',3));  dump(mp);   //chèn phần tử bằng insert và make\_pair  mp.insert(make\_pair('x',50));  dump(mp);     return 0; } |

Kết quả:

COPY

|  |
| --- |
| a:100 b:200  a:100 b:200 c:300  a:100 b:200 c:300 // Do key đã tồn tại nên sẽ không được chèn  a:100 b:200 c:300 x:50 |

Chúng ta cũng có thể kiểm tra việc chèn phần tử đã thực hiện hay chưa bằng phương thức first() hoặc second() từ kết quả trả về của hàm như sau:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  int main () {  /\*Khai báo và gán giá trị cho foo và bar\*/  map<char,int> mp;   mp['a']=100;  mp['b']=200;    //chèn phần tử với key chưa tồn tại trong map  auto r = mp.insert(pair<char,int>('z',300));  cout << r.second << "\n"; // Trả về 1 do được chèn   //chèn phần tử với key tồn tại trong map,   r = mp.insert(pair<char,int>('a',3));  cout << r.second << "\n"; // Trả về 0 do không được chèn     return 0; } |

**Thêm chèn 1 phần tử vào map trong C++ bằng hàm emplace**

**Hàm emplace** là một hàm thành viên trong class std:map, có tác dụng **chèn một hoặc nhiều phần tử vào map** thông qua hàm tạo của kiểu dữ liệu, cũng như tăng độ dài tương ứng của map.

Để chèn thêm 1 phần tử vào trong map C++, chúng ta sử dụng hàm emplace với cú pháp sau đây:

mp.emplace(k,v);

Trong đó

* mp là map ban đầu
* k,v là key và value.

Hàm map emplace sẽ thực hiện việc chèn phần tử nếu key của nó chưa tồn tại trong map. Ngược lại nếu key đó đã tòn tại, việc chèn thất bại và false sẽ được trả về

Và Hàm map emplace sẽ trả về một trình lặp trỏ tới vị trí phần tử vừa được chèn vào, nếu việc chèn thành công.

Lại nữa, phần tử trong map được sắp xếp theo thứ tự cụ thể trước khi được lưu, nên các phần tử cần chèn thêm vào map sẽ được tự động quyết định vị trí dựa trên bộ sắp xếp, và chúng ta sẽ không quyết định được vị trí cần chèn của phần tử.

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ":" << x.second << " ";  }  cout << endl; }  int main () {  std::map<char,int> mymap;   mymap.emplace('x',100);  mymap.emplace('y',200);  mymap.emplace('z',100);    dump(mymap);  return 0; } |

Kết quả:

COPY

|  |
| --- |
| x:100 y:200 z:100 |

**Chèn nhiều phần tử vào map trong C++ bằng hàm insert**

https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/map-trong-cpp/them-chen-map-trong-cpp/

Để chèn nhiều phần tử vào trong map C++, chúng ta sử dụng hàm insert với cú pháp sau đây:

mp.insert(iterator\_first, iterator\_last);

Trong đó

* mp là map ban đầu
* iterator\_first và iterator\_last là các trình lặp xác định phạm vi chứa các phần tử cần chèn ở trong một map khác vào map ban đầu.

Lưu ý ở đây, các phần tử cần chèn được lấy ra từ một phạm vi trong một map khác, và phạm vi này được chỉ định thông qua 2 trình lặp như trên

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<int, const char\*>& mp) {  for (auto x: mp) {  cout << x.first << ": " << x.second << " ";  }  cout << endl; }   int main () {  /\* Khởi tạo map mp1\*/  map<int, const char\*> mp1 =  {  {1, "a"},  {2, "b"},  {3, "c"},  {4, "d"},  {5, "e"},  };  // duyệt map mp1  dump(mp1);    /\*Khai báo map mp2\*/  map<int, const char\*> mp2;   /\*Chèn các phần tử từ đầu tới phần tử có key bằng 4  từ mp1 vào mp2\*/  mp2.insert(mp1.begin(),mp1.find(4));    // duyệt map mp2  dump(mp2);  return 0; } |

Và kết quả:

COPY

|  |
| --- |
| 1: a 2: b 3: c 4: d 5: e  1: a 2: b 3: c |

**Tổng kết**

Trên đây Kiyoshi đã hướng dẫn bạn về cách **thêm chèn chèn phần tử vào map trong C++** rồi. Để nắm rõ nội dung bài học hơn, bạn hãy thực hành viết lại các ví dụ của ngày hôm nay nhé.

**Xóa phần tử trong map C++**

21 tháng 1 ,2022

Hướng dẫn cách **xóa map trong C++**. Bạn sẽ học được cách sử dụng hàm **map erase C++** để xóa phần tử trong map sau bài học này.

Để xóa phần tử trong map trong C++, chúng ta sử dụng tới hàm thành viên erase(). Do khi xóa phần tử trong map thì các phần tử còn lại sẽ được tự động sắp xếp, nên lưu ý khác với các containers khác như list hay vector thì trong map không tồn tại</> các hàm pop\_front() hay pop\_back() để xóa phần tử ở đầu hay cuối map.

**Xóa 1 phần tử trong map bằng hàm erase c++**

**erase trong c++** là một hàm thành viên trong class std:map, có tác dụng **xóa một hoặc nhiều phần tử trong map C++** cũng như làm giảm độ dài tương ứng của nó.

Để **xóa 1 phần tử trong map c++** chúng ta sử dụng hàm erase với một trong 2 cú pháp sau đây:

mp.erase(itr);  
OR  
mp.erase(key);

Trong đó mp là map ban đầu, itr và key là trình lặp hoặc là khóa của phần tử cần xóa.

Nếu sử dụng erase(itr) thì hàm sẽ xóa đi phần tử tại vị trí mà trình lặp xác định bởi *itr* chỉ đến.

Và nếu sử dụng erase(key) thì hàm sẽ tìm phần tử có khóa bằng với key trong map và tiến hành xóa nó đi.

Ở đây trình lặp (iterator) là một vòng lặp có tác dụng giống như con trỏ, giúp truy cập đến các phần tử ở vị trí cụ thể trong map.

Khác với các container khác thì để chuyển trình lặp chỉ đến vị trí index thứ n trong map, chúng ta không thể đơn giản cộng vào n vào trình lặp, mà cần phải di chuyển lần lượt qua từng vị trí, bằng toán tử ++ với đủ số vòng lặp.

Ví dụ, trong trường hợp cần chỉ đến vị trí index thứ 5 trong map, chúng ta sẽ viết trình lặp p như sau:

COPY

|  |
| --- |
| int n = 5; auto itr = mp.begin(); //Tạo trình lặp trỏ đến vị trí đầu map for (int i=1; i <= n; i++ )  ++itr; |

Hàm map erase sẽ trả về số lượng phần tử đã được xóa đi từ map ban đầu.

Lưu ý phần tử trong map được sắp xếp theo thứ tự cụ thể trước khi được lưu, nên các phần tử còn lại sau khi xóa phần tử trong map sẽ được tự động quyết định vị trí dựa trên bộ sắp xếp. Do đó chúng ta sẽ không quyết định được vị trí của các phần tử sau khi xóa đi phần tử từ map.

Ví dụ cụ thể:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ": " << x.second << " ";  }  cout << endl; }  int main () {  map<char,int> mp;   mp['x']=100;  mp['y']=200;  mp['z']=300;  mp['t']=400;    dump(mp);     /\*Tạo trình lặp trỏ đến vị trí đầu tiên của map\*/  auto itr = mp.begin();  ++itr; //Di chuyển trình lặp đến vị trí thứ nhất trong map    //xóa phần tử tại vị trí itr chỉ đến (vị trí thứ nhất)  mp.erase(itr);  dump(mp);   //Tìm vị trí và xóa phần tử có key bằng 'z' trong map  mp.erase('z');  dump(mp);    //xóa phần tử có có key bằng 'y' trong map  mp.erase('y');  dump(mp);    return 0; } |

Kết quả:

COPY

|  |
| --- |
| t: 400 x: 100 y: 200 z: 300  t: 400 y: 200 z: 300  t: 400 y: 200  t: 400 |

Chúng ta cũng có thể kiểm tra số phần tử đã được xóa đi từ kết quả trả về của hàm như sau:

COPY

|  |
| --- |
| map<char,int> mp;  mp['x']=100; mp['y']=200; mp['z']=300; mp['t']=400;   auto r = mp.erase('t'); /\*r là số phần tử có key bằng 't' bị xóa\*/ std::cout << r;  //1 |

**Xóa các phần tử trong một phạm vi chỉ định bằng map erase c++**

Chúng ta sử dụng **map erase c++** để xóa các phần tử trong một phạm vi chỉ định của một map với cú pháp sau đây:

mp.erase( iterator\_first, iterator\_last);

Trong đó mp là map ban đầu, iterator\_first và iterator\_last là các trình lặp trỏ đến phạm vi bắt đầu và kết thúc xóa.

Lưu ý là phạm vi xóa sẽ được tính từ iterator\_first đến trước iterator\_last, nghĩa là phần tử ở vị trí iterator\_first sẽ được xóa nhưng phần tử ở vị trí iterator\_last sẽ không bị xóa đi.

Và các giá trị trình lặp (iterator) này được tính sau khi các phần tử đã được sắp xếp và lưu trong map, chứ không phải là theo thứ tự các phần tử khi chúng ta khai báo map.

Ứng dụng điều này, chúng ta có thể chỉ định trình lặp và xóa đi các phần tử trong phạm vi chỉ định từ map như sau:

COPY

|  |
| --- |
| #include <iostream> #include <map> using namespace std;  //Tạo hàm xuất map void dump(map<char,int>& mp) {  for (auto x: mp) {  cout << x.first << ": " << x.second << " ";  }  cout << endl; }   int main () {  map<char,int> mp;   mp['a']=100;  mp['b']=200;  mp['c']=300;  mp['d']=400;   mp['e']=500;  mp['f']=600;     dump(mp);    /\*Khai báo phạm vi cần xóa\*/  int start = 2, end =5;    /\*Tạo các trình lặp trỏ tới start và end với giá trị ban đầu\*/  auto itr\_start = mp.begin();   auto itr\_end = mp.begin();     /\*Thay đổi trình lặp tương ứng tới các vị trí start và end\*/  for (int i=1; i <= start; i++ )  ++itr\_start;   for (int i=1; i <= end; i++ )  ++itr\_end;    //Sau đó dùng hàm erase để xóa phạm vi là xong  mp.erase(itr\_start, itr\_end);  dump(mp);     return 0; } |

Kết quả:

COPY

|  |
| --- |
| a: 100 b: 200 c: 300 d: 400 e: 500 f: 600  a: 100 b: 200 f: 600 |

**Tổng kết**

https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/map-trong-cpp/xoa-phan-tu-trong-map-cpp/

Trên đây Kiyoshi đã hướng dẫn bạn về cách **xóa phần tử trong map C++** rồi. Để nắm rõ nội dung bài học hơn, bạn hãy thực hành viết lại các ví dụ của ngày hôm nay nhé.