(Ghi chú)

Ngôn ngữ lập trình cpp

**Thực hiện: Nguyễn Văn Hào**

**Email:** [hao2205tb@gmail.com](mailto:hao2205tb@gmail.com)

(Tài liệu được biên xoạn lại từ nhiều nguồn và không dùng trong mục đích kinh doanh kiếm tiền)

# Mục lục

[Mục lục 1](#_Toc126827019)

[Set 2](#_Toc126827020)

[Tham khảo 3](#_Toc126827021)

# Set

## Tổng quan

### Set trong c++ là gì

Set trong C++ là một tập hợp các phần tử duy nhất được sắp xếp theo thứ tự cụ thể, và được sử dụng làm tiêu chuẩn để xử lý các đối tượng chứa nhiều phần tử trong C++.

Mỗi phần tử trong set có giá trị phải là duy nhất, có nghĩa là nó không được trùng lặp với các giá trị còn lại trong set. Ngoài ra thì phần tử trong set không thể thay đổi giá trị, tuy nhiên chúng có thể được chèn hoặc xóa khỏi set.

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

Trong C++ cũng có một loại dữ liệu khá giống với set là multiset khi các phần tử trong chúng luôn được sắp xếp theo thứ tự. Tuy nhiên thì khác với multiset vốn cho phép các phần tử có thể trùng nhau cùng tồn tại, thì các phần tử trong set không được trùng nhau và luôn phải là duy nhất.

Nói tóm lại thì set trong C+++ sẽ giống như một mảng với các phần tử không trùng lặp và luôn được sắp xếp.

Về mặt tốc độ xử lý thì set có khả năng thêm, xóa, tìm kiếm dữ liệu với tốc độ cực cao là O(log N), và nó còn cao hơn cả vector với tốc là O(1). Tuy nhiên thì do các phần tử được quản lý dạng cây nhị phân nên tốc độ truy cập ngẫu nhiên tới một phần tử chỉ định trong set lại cực thấp.

Do đó, trong trường hợp chúng ta không hay thêm xóa tìm kiếm phần tử thì việc dùng vector sẽ có lợi hơn, do tốc độ cũng tương đương mà lại tiết kiệm bộ nhớ.

Tuy nhiên trong các trường hợp không cần thiết việc truy cập ngẫu nhiên và hay thêm xóa tìm kiếm phần tử thì việc sử dụng set thay cho vector sẽ có lợi nhiều hơn.

Loại Truy cập ngẫu nhiên Thêm xóa tìm kiếm ngẫu nhiên

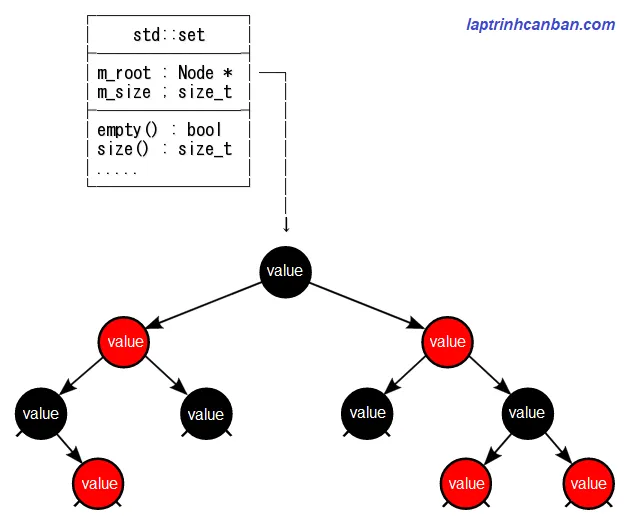
vector O(1) O(N)

set, multiset chậm O(log N)

### Cấu trúc dữ liệu set trong c++

Cấu trúc dữ liệu set trong C++ thuộc dạng Red–black tree (cây đỏ đen) - 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 set 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.



Trong các Node sẽ lưu giữ giá trị (value) cũng như con trỏ của các Node con (trái, phải) của nó.

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 set không tồn tại giá trị trùng nhau nên dấu < được sử dụng.

Độ sâu của các Node bằng nhau và cây Node thì cân bằng.

### std::set trong C++

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

## Khai báo set trong C++

### Khai báo 1 set trong C++

Cú pháp:

std::set<type> st;

Trong đó st là tên biến set và type là kiểu dữ liệu. 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.

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 set 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.

VD:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    set<double> name; //Khai báo set name kiểu double

    set<int> age;     //Khai báo set age kiểu int

}

Khi khai báo 1 set 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:

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 set thuộc kiểu struct\*/

std::set<Person> st;

### Khai báo đồng thời nhiều set trong C++

Cú pháp:

set<type> name1, name2, name3, ... ;

VD:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    set<string> name, job, sex;

    set<int> age;

}

## Khởi tạo set trong C++

Cú pháp:

std::set<type> st {value1, value2, value3, ...};

Trong đó

* type là kiểu dữ liệu
* st là tên biến set
* value là các giá trị của set

VD:

std::set<string> user{"Kiyoshi", "male", "Tokyo"};

//{"Kiyoshi", "male", "Tokyo"}

### Khai báo set 2 chiều trong C++

Cú pháp:

using namespace std;

set<set<type> > st {l1, l2, l3, ...};

Trong đó:

* st là tên biến set 2 chiều
* l là các set 1 chiều được sử dụng như phần tử của set 2 chiều

Lưu ý, chúng ta cần phải viết thêm dấu cách giữa cặp dấu > > khi khai báo set 2 chiều. Lý do là để phân biệt với toán tử >> được sử dụng để dịch chuyển bit trong C++.

VD:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    /\*Khai báo set 2 chiều\*/

    set<set<string> > all\_user{

        {"Kiyoshi", "male", "Hanoi"},

        {"Honda", "male", "Tokyo"},

        {"Ajinomoto", "female", "Osaka"}};

    return 0;

}

Chúng ta cũng có thể khởi tạo các set 1 chiều trước rồi dùng chúng để khai báo set 2 chiều như sau:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    /\*Khởi tạo các set 1 chiều làm phần tử trong set 2 chiều\*/

    set<string> user1{"Kiyoshi", "male", "Hanoi"};

    set<string> user2{"Honda", "male", "Tokyo"};

    set<string> user3{"Ajinomoto", "female", "Osaka"};

    /\*Khai báo set 2 chiều\*/

    set<set<string> > all\_user{ user1, user2, user3};

    return 0;

}

## Truy cập phần tử trong set C++

Khác với vector hay mảng, do cấu trúc của set theo dạng cây chứ không phải dạng mảng nên chúng ta không thể truy cập ngẫu nhiên vào phần tử bất kỳ trong một set.Do vậy chúng ta cũng không thể sử dụng index của các phần tử để truy cập vào nó theo cách thông thường được.

Thay vào đó, chúng ta cần phải tiến hành truy cập tuần tự vào các phần tử của set, thông qua vòng lặp hoặc là trình lặp.

VD:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    set<string> user{"Kiyoshi", "male", "Tokyo"};

    int n=0, index = 2;

    for (string x: user) {

        /\*Truy cập vào phần tử thứ 2 trong set và kết thúc khi tìm thấy\*/

        if (n == index) {

            cout << x << endl;

            break;

        }

        ++n ;

    }

    return 0;

}

male

Chúng ta cũng có thể truy cập và in ra toàn bộ các phần tử trong set 1 chiều như sau:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    set<string> user{"Kiyoshi", "male", "Tokyo"};

    for (string x: user) {

        cout << x << endl;

    }

    return 0;

}

Kiyoshi

Tokyo

male

Tương tự khi chúng ta cần truy cập vào phần tử trong set 2 chiều trong C++:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    /\*Khởi tạo các set 1 chiều làm phần tử trong set 2 chiều\*/

    set<string> user1{"Kiyoshi", "male", "Hanoi"};

    set<string> user2{"Honda", "male", "Tokyo"};

    set<string> user3{"Ajinomoto", "female", "Osaka"};

    /\*Khai báo set 2 chiều\*/

    set<set<string> > all\_user{ user1, user2, user3};

    for (auto x: all\_user) {

        for (auto y: x) {

            cout << y << endl;

        }

    }

    return 0;

}

Honda

male

Tokyo

Ajinomoto

female

Osaka

## vector vs set trong C++

Như đã phân tích ở trên thì giữa set và vector trong C++ có một số điểm khác biệt như sau:

Vector là mảng động còn set có cấu trúc cây nhị phân tạo bởi các Node

Vector chấp nhận phần tử trùng lặp còn set thì không. Phần tử trong set phải là duy nhất

Phần tử trong vector không được sắp xếp còn phần tử trong set thì được tự động sắp xếp thứ tự cụ thể.

Vector có tốc độ truy cập ngẫu nhiên nhanh hơn set, nhưng có tốc độ thêm xóa tìm kiếm ngẫu nhiên kém hơn set.

Thông thường trong trường hợp chúng ta không hay thêm xóa tìm kiếm phần tử thì việc dùng vector sẽ có lợi hơn, do tốc độ cũng tương đương mà lại tiết kiệm bộ nhớ.

Tuy nhiên trong các trường hợp không cần thiết việc truy cập ngẫu nhiên và hay thêm xóa tìm kiếm phần tử thì việc sử dụng set thay cho vector sẽ có lợi nhiều hơn.

## Duyệt set

### Duyệt set trong C++ bằng vòng lặp dựa trên phạm vi

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

for ( auto& x : v) {

    // Xử lý

}

Trong đó:

* v là tên set
* x là tên một biến dùng để gán từng phần tử được lấy từ set
* auto là kiểu suy luận giúp tự xác định kiểu dữ liệu của giá trị lấy từ set

VD:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    set<int> data{5, 6, 88, -2};

    for (auto x: data) {

        cout << x << endl;

    }

}

-2

5

6

88

Nếu trong set chỉ chứa các phần tử thuộc kiểu dữ liệu nguyên thủy, chúng ta cũng có thể thay thế auto bằng tên kiểu, ví dụ set ở trên chỉ chứa các phần tử thuộc kiểu int nên chúng ta có thể viết:

set<int> data{5, 6, 88, -2};

for (int x: data) {

    cout << x << endl;

}

Một cách tương tự chúng ta cũng có thể dùng vòng lặp dựa trên phạm vi để duyệt set 2 chiều trong C++ như sau:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    set<set<int> > data{{5, 2}, {6,3}, {88, -2}};

    for (auto x: data) {

        for (auto y: x) {

            cout << y << endl;

        }

    }

}

-2

88

2

5

3

6

### Duyệt set trong C++ bằng iterator

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

for(auto itr = st.begin(); itr != st.end(); ++itr) {

cout << \*itr << endl;

}

Trong đó:

* st là tên set
* itr là tên iterator

VD:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    set<int> data{5, 6, 88, -2};

    for(auto itr = data.begin(); itr != data.end(); ++itr) {

        cout << \*itr << endl;

    }

}

-2

5

6

88

## Lấy kích thước set trong C++ bằng hàm size

Cú pháp:

st.size();

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

VD:

#include <iostream>

#include <set>

using namespace std;

int main ()

{

    set<int> st{3, 1, 4,5};

    cout<< st.size() <<endl;

    return 0;

}

4

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

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

#include <iostream>

#include <set>

using namespace std;

int main ()

{

    set<int> st{3, 3,  1, 4, 4, 5};

    for (auto x: st) {

        cout << " "<< x ;

    }

    cout<< "\n "<< st.size() ;

    return 0;

}

1 3 4 5

4

## Thêm chèn phần tử vào set trong C++

### Chèn 1 phần tử vào set trong C++ bằng hàm insert

Cú pháp:

st.insert(value);

Trong đó st là set ban đầu, và value là phần tử cần chèn.

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

VD:

#include <iostream>

#include <set>

using namespace std;

//Tạo hàm xuất set

void dump(set<int>& st)

{

    for (auto x: st) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main ()

{

    set<int> st{1, 3, 6, 7, 5};

    dump(st);

    //chèn phần tử đã tồn tại trong set

    st.insert(3);

    dump(st);

    //chèn phần tử chưa tồn tại trong set

    st.insert(4);

    dump(st);

    return 0;

}

1 3 5 6 7

1 3 5 6 7

1 3 4 5 6 7

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:

#include <iostream>

#include <set>

using namespace std;

int main ()

{

    set<int> st{1, 3, 6, 7, 5};

    auto r = st.insert(3);

    std::cout << r.second << "\n";

    // 0 : không chèn

    r = st.insert(4);

    std::cout << r.second << "\n";

    // 1 : đã chèn

    return 0;

}

### Chèn nhiều phần tử vào set trong C++ bằng hàm insert

Cú pháp:

st.insert(il);

Trong đó st là set ban đầu, và il là một initializer\_list chứa các phần tử cần chèn vào set.

VD:

#include <iostream>

#include <set>

using namespace std;

//Tạo hàm xuất set

void dump(set<int>& st)

{

    for (auto x: st) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main ()

{

    set<int> st{1, 3, 6, 7, 5};

    dump(st);

    /\*Khai báo initializer\_list chứa phần tử cần chèn \*/

    int myints[]= {2,10,15};

    /\*Chèn initializer\_list vào set\*/

    st.insert (myints,myints+3);

    dump(st);

    return 0;

}

1 3 5 6 7

1 2 3 5 6 7 10 15

## Xoá phần tử

### Xóa 1 phần tử trong set bằng hàm erase c++

Cú pháp:

st.erase(itr);

OR

st.erase(value);

Trong đó st là set ban đầu, itr và value là trình lặp hoặc là giá trị 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(value) thì hàm sẽ tìm phần tử có giá trị bằng với value trong list và tiến hành xóa nó đi.

VD:

#include <iostream>

#include <set>

using namespace std;

//Tạo hàm xuất set

void dump(set<int>& st)

{

    for (auto x: st) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main ()

{

    set<int> st{1, 3, 6, 7, 5};

    dump(st);

    /\*Tạo trình lặp trỏ đến vị trí đầu tiên của set\*/

    auto itr = st.begin();

    ++itr; //Di chuyển trình lặp đến vị trí thứ nhất trong set

    //xóa phần tử tại vị trí itr chỉ đến (vị trí thứ nhất)

    st.erase(itr);

    dump(st);

    //xóa phần tử có giá trị bằng 6 trong set

    st.erase(6);

    dump(st);

    return 0;

}

1 3 5 6 7

1 5 6 7

1 5 7

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

#include <iostream>

#include <set>

using namespace std;

int main ()

{

    set<int> st{1, 3, 6, 7, 5};

    auto r = st.erase(2); //Xóa phần tử không tồn tại trong set

    std::cout << r << "\n";

    // 0

    r = st.erase(6); //Xóa phần tử tồn tại trong set

    std::cout << r << "\n";

    // 1

    return 0;

}

### Xóa các phần tử trong một phạm vi chỉ định bằng set erase c++

Cú pháp:

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

Trong đó st là set 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 là [first,last) 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 set, chứ không phải là theo thứ tự các phần tử khi chúng ta khai báo set.

VD:

#include <iostream>

#include <set>

using namespace std;

//Tạo hàm xuất set

void dump(set<int>& st)

{

    for (auto x: st) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main ()

{

    set<int> st{1, 3, 6, 7, 5, 9, 2};

    dump(st);

    int n;

    /\*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 = st.begin();

    auto itr\_end = st.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

    st.erase(itr\_start, itr\_end);

    dump(st);

    return 0;

}

1 2 3 5 6 7 9

1 2 7 9

## Sao chép set trong C++

set 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 set vào một set mới.

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

std::set<type> st\_dest(st\_src);

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

VD:

#include <iostream>

#include <set>

using namespace std;

//Tạo hàm xuất set

void dump(set<int>& st)

{

    for (auto x: st) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main()

{

    set<int> st\_src{3, 1, 4,5};

    cout << "Orginary set" << endl;

    dump (st\_src);

    /\*Sao chép set\*/

    set<int> st\_dest(st\_src);

    cout << "Copy set" << endl;

    dump (st\_dest);

    return 0;

}

Orginary set

1 3 4 5

Copy set

1 3 4 5

### Hoán đổi 2 set trong C++

Cú pháp:

st1.swap(st2);

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

VD:

#include <iostream>

#include <set>

using namespace std;

//Tạo hàm xuất set

void dump(set<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main()

{

    set<int> st1{3, 1, 4, 1, 5};

    set<int> st2{9, 8, 7};

    cout << "Before swap" << endl;

    dump(st1);

    dump(st2);

    st1.swap(st2);

    cout << "After swap" << endl;

    dump(st1);

    dump(st2);

}

Before swap

1 3 4 5

7 8 9

After swap

7 8 9

1 3 4 5

Ngoài cách dùng hàm set swap, chúng ta cũng có thể dùng function template là std::swap để tiến hành hoán đổi 2 set 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

VD:

#include <iostream>

#include <utility>

#include <set>

using namespace std;

//Tạo hàm xuất set

void dump(set<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main()

{

    set<int> st1{3, 1, 4, 1, 5};

    set<int> st2{9, 8, 7};

    cout << "Before swap" << endl;

    dump(st1);

    dump(st2);

    swap(st1,st2);

    cout << "After swap" << endl;

    dump(st1);

    dump(st2);

}

Before swap

3 1 4 1 5

9 8 7

After swap

9 8 7

3 1 4 1 5

## Tính tổng

### Tính tổng các phần tử trong set C++ bằng std::accumulate

Để sử dụng hàm std::accumulate, chúng ta cần include header file numeric vào đầu chương trình như sau:

Cú pháp:

std::accumulate(st.begin(), st.end(), 0);

Trong đó st là tên set cần tính tổng các phần tử trong thứ tự các phần tử, còn st.begin() và st.end() lần lượt là các trình lặp có tác dụng như con trỏ chỉ đến vị trí đầu tiên và cuối cùng của set. Đối số cuối cùng (0) là giá trị khởi tạo của tổng.

VD:

#include <iostream>

#include <numeric>

#include <set>

using namespace std;

int main ()

{

    set<int> st{3, 1, 4, 2, 5};

    int sum = accumulate(st.begin(), st.end(), 0);

    cout << "SUM = " << sum << endl;

    return 0;

}

SUM = 15

### Tính tổng các phần tử trong set C++ bằng vòng lặp

VD:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    set<int> st{3, 1, 4, 2, 5};

    int sum = 0;

    for (auto x: st) {

        sum += x;

    }

    cout << "SUM = " << sum << endl;

    return 0;

}

SUM = 15

## Set trống

### Kiểm tra set trống trong C++ bằng hàm empty

Cú pháp:

st.empty();

Trong đó st là set cần kiểm tra.

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

VD:

#include <iostream>

#include <set>

using namespace std;

int main ()

{

    std::set<int> st;

    if( st.empty() )

        std::cout << "empty.\n";

    else

        std::cout << "not empty.\n";

    std::set<int> st2{1, 2, 3};

    if( st2.empty() )

        std::cout << "empty.\n";

    else

        std::cout << "not empty.\n";

    return 0;

}

empty.

not empty.

### Làm trống 1 set trong C++ bằng hàm clear

Cú pháp:

st.clear();

Trong đó st là set cần làm trống.

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

VD:

#include <iostream>

#include <set>

using namespace std;

//Tạo hàm xuất set

void dump(set<int>& st)

{

    for (auto x: st) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main ()

{

    std::set<int> st{3, 1, 4};

    dump(st);

    st.clear();

    dump(st);

}

1 3 4

# Multiset

## Tổng quan

### Multiset trong c++ là gì

multiset trong C++ là một tập hợp các phần tử có thể trùng lặp được sắp xếp theo thứ tự cụ thể, và được sử dụng làm tiêu chuẩn để xử lý các đối tượng chứa nhiều phần tử trong C++.

Các phần tử trong multiset có thể trùng với các phần tử khác, ngoài ra thì phần tử trong multiset không thể thay đổi giá trị, tuy nhiên chúng có thể được chèn hoặc xóa khỏi multiset.

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

Trong C++ cũng có một loại dữ liệu khá giống với multiset là set khi các phần tử trong chúng luôn được sắp xếp theo thứ tự. Tuy nhiên thì khác với set vốn không cho phép các phần tử có thể trùng nhau cùng tồn tại, thì các phần tử trong multiset lại có thể trùng với các phần tử khác nó.

Nói tóm lại thì multiset trong C+++ sẽ giống như một mảng với các phần tử luôn được sắp xếp theo một thứ tự cụ thể.

Về mặt tốc độ xử lý thì multiset có khả năng thêm, xóa, tìm kiếm dữ liệu với tốc độ cực cao là O(log N), và nó còn cao hơn cả vector với tốc là O(1). Tuy nhiên thì giống với set thì do các phần tử được quản lý dạng cây nhị phân nên tốc độ truy cập ngẫu nhiên tới một phần tử chỉ định trong multiset lại cực thấp.

Do đó, trong trường hợp chúng ta không hay thêm xóa tìm kiếm phần tử thì việc dùng vector sẽ có lợi hơn, do tốc độ cũng tương đương mà lại tiết kiệm bộ nhớ.

Tuy nhiên trong các trường hợp không cần thiết việc truy cập ngẫu nhiên và hay thêm xóa tìm kiếm phần tử thì việc sử dụng multiset thay cho vector sẽ có lợi nhiều hơn.

Loại Truy cập ngẫu nhiên Thêm xóa tìm kiếm ngẫu nhiên

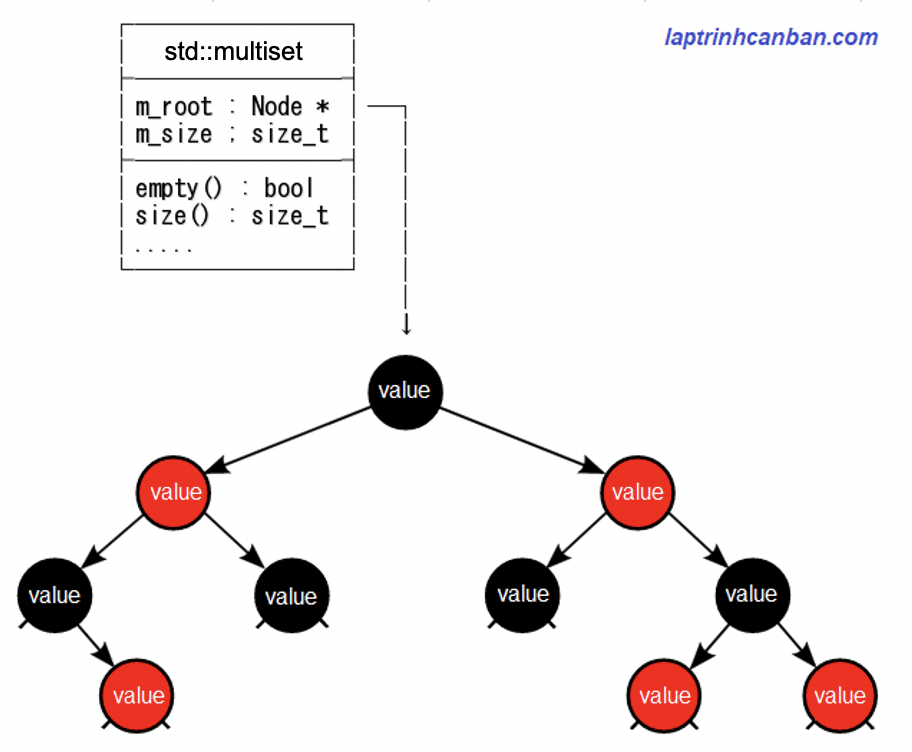
vector O(1) O(N)

multiset, set chậm O(log N)

### Cấu trúc dữ liệu multiset trong c++

Giống với set thì cấu trúc dữ liệu multiset trong C++ thuộc dạng Red–black tree (cây đỏ đen) - 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 multiset 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.



Trong các Node sẽ lưu giữ giá trị (value) cũng như con trỏ của các Node con (trái, phải) của nó.

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 multiset có thể trùng nhau nên dấu <= được sử dụng.

Độ sâu của các Node bằng nhau và cây Node thì cân bằng.

### std::multiset trong C++

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

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

Cần đặc biệt lưu ý ở đây là #include <set> chứ không phải là #include <multiset> .Khác với các container khác vốn có header file riêng biệt để quản lý thì multiset lại dùng chung header file với set.

#include <set>

int main()

{

    std::multiset<int> st1;

    std::multiset<double> st2;

}

## Khai báo multiset trong C++

### Khai báo 1 multiset trong C++

Cú pháp:

std::multiset<type> st;

Trong đó st là tên biến multiset và type là kiểu dữ liệu.

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 multiset 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.

VD:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    multiset<double> name; //Khai báo multiset name kiểu double

    multiset<int> age;     //Khai báo multiset age kiểu int

}

Khi khai báo 1 multiset 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:

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 multiset thuộc kiểu struct\*/

std::multiset<Person> st;

### Khai báo đồng thời nhiều multiset trong C++

Cú pháp:

multiset<type> name1, name2, name3, ... ;

VD:

#include <iostream>

#include <set>

using namespace std;

int main()

{

    multiset<string> name, job, sex;

    multiset<int> age;

}

## Khởi tạo multiset trong C++

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

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

std::multiset<type> st {value1, value2, value3, ...};

Trong đó

type là kiểu dữ liệu

st là tên biến multiset

value là các giá trị của multiset

Ví dụ:

COPY

std::multiset<string> user{"Kiyoshi", "male", "Tokyo"};

//{"Kiyoshi", "male", "Tokyo"}

Khai báo multiset 2 chiều trong C++

Giống như mảng thì chúng ta cũng có thể sử dụng multiset đa chiều trong C++, và loại multiset đa chiều hay được sử dụng đó chính là multiset 2 chiều trong C++.

Để khai báo multiset 2 chiều trong C++ cũng như các loại multiset đa chiều khác, chúng ta sử dụng tới cú pháp sau đây:

using namespace std;

multiset<multiset<type> > st {l1, l2, l3, ...};

Trong đó:

st là tên biến multiset 2 chiều

l là các multiset 1 chiều được sử dụng như phần tử của multiset 2 chiều

Lưu ý, chúng ta cần phải viết thêm dấu cách giữa cặp dấu > > khi khai báo multiset 2 chiều. Lý do là để phân biệt với toán tử >> được sử dụng để dịch chuyển bit trong C++.

Ví dụ cụ thể:

COPY

#include <iostream>

#include <set>

using namespace std;

int main()

{

/\*Khai báo multiset 2 chiều\*/

multiset<multiset<string> > all\_user{

{"Kiyoshi", "male", "Hanoi"},

{"Honda", "male", "Tokyo"},

{"Ajinomoto", "female", "Osaka"}};

return 0;

}

Chúng ta cũng có thể khởi tạo các multiset 1 chiều trước rồi dùng chúng để khai báo multiset 2 chiều như sau:

COPY

#include <iostream>

#include <set>

using namespace std;

int main()

{

/\*Khởi tạo các multiset 1 chiều làm phần tử trong multiset 2 chiều\*/

multiset<string> user1{"Kiyoshi", "male", "Hanoi"};

multiset<string> user2{"Honda", "male", "Tokyo"};

multiset<string> user3{"Ajinomoto", "female", "Osaka"};

/\*Khai báo multiset 2 chiều\*/

multiset<multiset<string> > all\_user{ user1, user2, user3};

return 0;

}

Truy cập phần tử trong multiset C++

Khác với vector hay mảng, do cấu trúc của multiset theo dạng cây chứ không phải dạng mảng nên chúng ta không thể truy cập ngẫu nhiên vào phần tử bất kỳ trong một multiset.

Do vậy chúng ta cũng không thể sử dụng index của các phần tử để truy cập vào nó theo cách thông thường được.

Ví dụ nếu dùng index để truy cập vào vị trí ngẫu nhiên trong multiset thì lỗi sẽ trả về như sau:

COPY

multiset<string> user{"Kiyoshi", "male", "Tokyo"};

cout << user[1];

//main.cpp:9:13: error: no match for ‘operator[]’

Thay vào đó, chúng ta cần phải tiến hành truy cập tuần tự vào các phần tử của multiset, thông qua vòng lặp hoặc là trình lặp mà Kiyoshi đã giới thiệu trong bài Duyệt multiset trong C++.

Ví dụ, chúng ta có thể truy cập vào phần tử của multiset 1 chiều thông qua vòng lặp dựa trên phạm vi như sau:

COPY

#include <iostream>

#include <set>

using namespace std;

int main()

{

multiset<string> user{"Kiyoshi", "male", "Tokyo"};

int n=0, index = 2;

for (string x: user) {

/\*Truy cập vào phần tử thứ 2 trong multiset và kết thúc khi tìm thấy\*/

if (n == index) {

cout << x << endl;

break;

}

++n ;

}

return 0;

}

// male

Chúng ta cũng có thể truy cập và in ra toàn bộ các phần tử trong multiset 1 chiều như sau:

COPY

#include <iostream>

#include <set>

using namespace std;

int main()

{

multiset<string> user{"Kiyoshi", "male", "Tokyo"};

for (string x: user) {

cout << x << endl;

}

return 0;

}

Kết quả:

COPY

Kiyoshi

Tokyo

male

Tương tự khi chúng ta cần truy cập vào phần tử trong multiset 2 chiều trong C++:

COPY

#include <iostream>

#include <set>

using namespace std;

int main()

{

/\*Khởi tạo các multiset 1 chiều làm phần tử trong multiset 2 chiều\*/

multiset<string> user1{"Kiyoshi", "male", "Hanoi"};

multiset<string> user2{"Honda", "male", "Tokyo"};

multiset<string> user3{"Ajinomoto", "female", "Osaka"};

/\*Khai báo multiset 2 chiều\*/

multiset<multiset<string> > all\_user{ user1, user2, user3};

for (auto x: all\_user) {

for (auto y: x) {

cout << y << endl;

}

}

return 0;

}

Và kết quả:

COPY

Honda

male

Tokyo

Ajinomoto

female

Osaka

vector vs multiset trong C++

Như đã phân tích ở trên thì giữa multiset và vector trong C++ có một số điểm khác biệt như sau:

Vector là mảng động còn multiset có cấu trúc cây nhị phân tạo bởi các Node

Phần tử trong vector không được sắp xếp còn phần tử trong multiset thì được tự động sắp xếp thứ tự cụ thể.

Vector có tốc độ truy cập ngẫu nhiên nhanh hơn multiset, nhưng có tốc độ thêm xóa tìm kiếm ngẫu nhiên kém hơn multiset.

Thông thường trong trường hợp chúng ta không hay thêm xóa tìm kiếm phần tử thì việc dùng vector sẽ có lợi hơn, do tốc độ cũng tương đương mà lại tiết kiệm bộ nhớ.

Tuy nhiên trong các trường hợp không cần thiết việc truy cập ngẫu nhiên và hay thêm xóa tìm kiếm phần tử thì việc sử dụng multiset thay cho vector sẽ có lợi nhiều hơn.

# Tham khảo

Lập trình căn bản:

<https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/>