(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 **Error! Bookmark not defined.**](#_Toc126778440)

[Vector **Error! Bookmark not defined.**](#_Toc126778441)

[Tham khảo **Error! Bookmark not defined.**](#_Toc126778442)

# Vector

## Tổng quan

Kiểu vector trong C++ hay còn gọi là mảng động hoặc mảng độ dài thay đổi, là kiểu dữ liệu có chức năng tương tự như mảng trong C++ nhưng lại có khả năng tự thay đổi độ dài.

## Khai báo

### Khai báo 1 vector

using namespace std;

vector <type> data;

Trong đó:

* type: kiểu dữ liệu
* data: tên biến vector

VD

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    vector<string> name; //Khai báo vector name kiểu string

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

}

### Khai báo bao gồm chỉ định số phần tử

std::vector<type> data(length);

Trong đó length chính là độ dài(số phần tử() của vector cần tạo.

Với cách khai báo vector này thì các phần tử trong vector sẽ được gán giá trị mặc định tùy thuộc vào kiểu dữ liệu sử dụng

Ví dụ, nếu dùng kiểu số như int, double thì các phần tử sẽ có giá trị mặc định bằng 0 như sau:

VD

/\*Khai báo vector data kiểu số\*/

std::vector<int> data1(3);    //{0, 0, 0}

std::vector<double> data2(3); //{0, 0, 0}

Tuy nhiên nếu dùng kiểu chữ như char, string thì các phần tử sẽ có giá trị mặc định bằng Null được biểu diễn bởi trống khi in ra màn hình như sau:

VD

/\*Khai báo vector data kiểu chữ\*/

std::vector<char> data1(3)   //{, , }

std::vector<string> data2(3) //{, , }

### Khai báo đồng thời nhiều vector

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

VD

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    vector<string> name, job, sex;

    vector<int> age;

}

## Gán giá trị

data[index] = value;

VD

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    vector<int> data(3);

    data[0]=1;

    data[1]=4;

    data[2]=88;

}

## Khởi tạo

### Khởi tạo vector trong C++ với các phần tử riêng biệt

std::vector<type> data {value1, value2, value3, ...};

Trong đó

* type là kiểu dữ liệu
* data là tên biến vector
* value là các giá trị của vector

VD

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

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

### Khởi tạo vector trong C++ với các phần tử giống nhau

std::vector<type> data(length, value);

Trong đó length là số phần tử(độ dài), và value là giá trị sử dụng để khai báo đồng loạt các phần tử của vector cần tạo.

VD

//Khởi tạo vector data kiểu int gồm 5 phần tử có giá trị bằng 8

std::vector<int> data(5, 8); //{8, 8, 8, 8, 8}

### Khai báo vector 2 chiều

vector<vector<type> > data {v1, v2, v3, ...};

Trong đó:

* data là tên biến vector 2 chiều
* v là các vector 1 chiều được sử dụng như phần tử của vector 2 chiều

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

VD

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    /\*Khai báo vector 2 chiều\*/

    vector<vector<string> > all\_user{

        {"Kiyoshi", "male", "Hanoi"},

        {"Honda", "male", "Tokyo"},

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

    return 0;

}

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

#include <iostream>

#include <vector>

using namespace std;

int main()

{

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

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

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

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

    /\*Khai báo vector 2 chiều\*/

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

    return 0;

}

## Truy cập phần tử

data[index];

Trong đó data là tên biến vector và index là vị trí của cần truy xuất.

#include <iostream>

#include <vector>

using namespace std;

int main()

{

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

    cout << user[0] <<endl;

    cout << user[1] <<endl;

    cout << user[2] <<endl;

    return 0;

}

Kiyoshi

male

Tokyo

## Duyệt vector

### Duyệt vector trong C++ bằng vòng lặp và index

VD

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    vector<int> data{5, 6, 88, -2};

    for (short i=0; i< data.size();i++){

        cout << data[i] <<endl;

    }

    return 0;

}

5

6

88

-2

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

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

for ( auto & x : v) {

// Xử lý

}

Trong đó:

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

VD

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    vector<int> data{5, 6, 88, -2};

    for (auto x: data) {

        cout << x << endl;

    }

}

5

6

88

-2

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

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

for (int x: data) {

    cout << x << endl;

}

### Duyệt vector trong C++ bằng iterator

Trong C++, các kiểu dữ liệu như vector, list, 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 vector trong C++ với cú pháp như sau:

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

// Xử lý

}

Trong đó:

* v là tên vector
* itr là tên iterator

VD

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    vector<int> data{5, 6, 88, -2};

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

        cout << \*itr << endl;

    }

}

5

6

88

-2

## Kích thước vector

### Lấy kích thước vector

vt.size();

Trong đó vt là vector cần kiểm tra.

Hàm size sẽ trả về kích thước của vector, cũng chính là số phần tử có trong vector đó.

VD

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    vector<int> data{5, 6, 88, -2};

    for (short i=0; i< data.size();i++){

        cout << data[i] <<endl;

    }

    return 0;

}

5

6

88

-2

### Thay đổi kích thước vector

vt.resize(n);

Trong đó vt là vector cần thay đổi kích thước, và n là kích thước cần thay đổi.

Nếu n nhỏ hơn kích thước(số phần tử) ban đầu của vector thì các phần tử bị thừa sẽ bị xóa đi.

Ngược lại nếu n lớn hơn kích thước(số phần tử) ban đầu của vector, các giá trị mặc định tùy thuộc vào kiểu dữ liệu sẽ được thêm vào vector cho đủ độ dài. Ví dụ nếu vector thuộc kiểu số như int hay double thì số 0 sẽ được thêm vào. Còn nếu vector thuộc kiểu chữ như char hay vector, giá trị trống sẽ được thêm vào.

VD

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& vt)

{

    for (auto x: vt) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main ()

{

    vector<int> vt{3, 1, 4,5};

    dump(vt);

    cout<< "Size before: "<<vt.size()<<endl;

    /\*Thay đổi kích thước nhỏ hơn ban đầu\*/

    vt.resize(2);

    dump(vt);

    cout<< "Size after: "<<vt.size()<<endl;

    /\*Thay đổi kích thước lớn hơn ban đầu\*/

    vt.resize(10);

    dump(vt);

    cout<< "Size after: "<<vt.size()<<endl;

    return 0;

}

3 1 4 5

Size before: 4

3 1

Size after: 2

3 1 0 0 0 0 0 0 0 0

Size after: 10

## Lấy phần tử

### Lấy phần tử đầu tiên

vt.front();

Trong đó vt là vector cần cắt ra phần tử ban đầu. Hàm front() sẽ trả về tham chiếu tới phần tử đầu tiên của vector, qua đó chúng ta có thể cắt lấy giá trị của phần tử đầu tiên này, hoặc là gán một phần tử khác để thay đổi nó.

Lưu ý chúng ta không dùng hàm front() cho vector trống.

VD

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& vt)

{

    for (auto x: vt) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main() {

    vector<int> vt{3, 1, 4, 1, 5, 7, 9};

    dump(vt);

    //Tạo tham chiếu tới vị trí phần tử đầu tiên của vector

    int front\_element = vt.front();

    cout << front\_element << endl;

    //Thay đổi phần tử đầu tiên

    vt.front() = 2;

    dump(vt);

    return 0;

}

3 1 4 1 5 7 9

3

2 1 4 1 5 7 9

### Lấy phần tử cuối cùng

vt.back();

Trong đó vt là vector cần cắt ra phần tử ban đầu. Hàm back() sẽ trả về tham chiếu tới phần tử cuối cùng của vector, qua đó chúng ta có thể cắt lấy giá trị của phần tử cuối cùng này, hoặc là gán một phần tử khác để thay đổi nó.

Lưu ý chúng ta không dùng hàm back() cho vector trống.

VD

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& vt)

{

    for (auto x: vt) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main() {

    vector<int> vt{3, 1, 4, 1, 5, 7, 9};

    dump(vt);

    //Tạo tham chiếu tới vị trí phần tử cuối cùng của vector

    int back\_element = vt.back();

    cout << back\_element << endl;

    //Thay đổi phần tử cuối cùng

    vt.back() = 2;

    dump(vt);

    return 0;

}

3 1 4 1 5 7 9

3

2 1 4 1 5 7 9

### Lấy một phạm vi từ vector

Sao chép từng phần tử trong phạm vi vào vector đích

VD

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& vt)

{

    for (auto x: vt) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main() {

    vector<int> vt{3, 1, 4, 1, 5, 7, 9};

    dump(vt);

    //Khai báo vector chứa kết quả

    vector<int> vt2;

    //Khai báo phạm vi cắt

    int start = 2, end = 4;

    //Truy cập vào phạm vi để lấy và lưu các phần tử vào vector kết quả

    for (short i = start; i<= end;i++){

        vt2.push\_back(vt[i]);

    }

    dump(vt2);

    return 0;

}

3 1 4 1 5 7 9

4 1 5

## Thêm chèn phần tử vào vector

### Thêm phần tử vào cuối vector bằng push\_back()

v.push\_back(value);

Trong đó v là tên vector, và value là giá trị sẽ được copy vào phần tử vừa được tạo ra. Nếu phần tử thêm vào có kiểu khác với kiểu của vector thì nó sẽ được chuyển về kiểu của vector trước khi được thêm vào.

Hàm push\_back thuộc kiểu void, do đó nó sẽ không trả giá trị. Bởi vậy nó sẽ thêm phần tử vào vị trí cuối cùng trong vector chứ không trả về chuỗi kết quả.

VD

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main() {

    vector<int> v{3, 1, 4, 1, 5, 7, 9};

    dump(v);

    v.push\_back(88);

    dump(v);

    return 0;

}

3 1 4 1 5 7 9

3 1 4 1 5 7 9 88

Lưu ý, nếu phần tử thêm vào có kiểu khác với kiểu của vector thì nó sẽ được chuyển về kiểu của vector trước khi được thêm vào. Ví dụ nếu thêm một ký tự vào một vector thuộc kiểu int, thì ký tự đó sẽ được chuyển về mã Unicode trước khi được thêm vào vector ban đầu như sau:

vector<int> v{3, 1, 4, 1, 5, 7, 9};

dump(v);

// 3 1 4 1 5 7 9

v.push\_back('a');

dump(v);

// 3 1 4 1 5 7 9 97

### Thêm phần tử vào cuối vector bằng emplace\_back()

v.emplace\_back(args);

Trong đó v là tên vector, và args là các thông tin được sử dụng trong hàm tạo value\_type tương ứng với kiểu giá trị để tạo nên phần tử trong vector.

VD

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main() {

    vector<int> v{3, 1, 4, 1, 5, 7, 9};

    dump(v);

    v.emplace\_back(88);

    dump(v);

    return 0;

}

3 1 4 1 5 7 9

3 1 4 1 5 7 9 88

### Khác biệt giữa push\_back và emplace\_back

Hàm emplace\_back khác với hàm push\_back ở chỗ, nó không nhận trực tiếp giá trị và copy giá trị này vào phần tử được thêm, mà nó sẽ dùng giá trị này như là đối số trong hàm tạo nên phần tử.

Ví dụ, nếu chúng ta thêm phần tử vào vector bằng hàm push\_back:

vector<int> v{3, 1, 4, 1, 5, 7, 9};

v.push\_back(88);

Điều đó có nghĩa là giá trị 88 sẽ được sử dụng trực tiếp để copy vào một phần tử mới được tạo ở cuối vector.

Còn nếu chúng ta thêm phần tử vào vector bằng hàm emplace\_back:

vector<int> v{3, 1, 4, 1, 5, 7, 9};

v.emplace\_back(88);

Điều đó có nghĩa là giá trị 88 được sử dụng như đối số trong hàm tạo thuộc class std:In tương ứng với kiểu int của giá trị 88. Hàm tạo này sẽ tạo ra phần tử mới được tạo ở cuối vector.

Do hàm emplace\_back sẽ tạo trực tiếp phần tử bên trong vector mà không cần phải qua việc copy (di chuyển) giá trị vào trong vector, nên nó được cho là có tốc độ xử lý nhanh hơn hàm push\_back.

### Chèn 1 phần tử vào vị trí chỉ định trong vector C++ bằng hàm insert

v.insert(p, value);

Trong đó v là vector ban đầu, value là phần tử cần chèn, và p là trình lặp trỏ đến vị trí cần chèn trong vector.

Trong trường hợp cần chỉ đến vị trí index thứ n trong vector, chúng ta sẽ viết trình lặp p như sau:

v.insert(v.begin() + i, value);

Trong đó v.begin() chỉ đến vị trí đầu tiên trong vector, và i là index của vị trí phần tử cần chỉ đến.

VD

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main ()

{

  vector<int> v{3, 1, 4, 1, 5, 7, 9};

  v.insert(v.begin()+2, 55); //Thêm phần tử 55 vào vị trí thứ 2

  dump(v);

 v.insert(v.begin()+4, 22); //Thêm phần tử 22 vào vị trí thứ 4

  dump(v);

  return 0;

}

3 1 55 4 1 5 7 9

3 1 55 4 22 1 5 7 9

## Xoá phần tử

### Xóa phần tử cuối cùng trong vector C++ bằng pop\_back

Hàm pop\_back là một hàm thành viên trong class std:vector, có tác dụng xóa phần tử cuối cùng trong vector cũng như giảm độ dài của nó đi một đơn vị.

Cú pháp sử dụng hàm pop\_back để xóa phần tử cuối cùng trong vector như sau:

v.pop\_back();

Trong đó v là tên vector cần xóa phần tử cuối cùng.

VD

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main ()

{

    vector<int> v{3, 1, 4, 1, 5};

    cout << "Before erase" << endl;

    dump (v);

    v.pop\_back();

    cout << "After erase" << endl;

    dump (v);

    return 0;

}

Before erase

3 1 4 1 5

After erase

3 1 4 1

### Xóa 1 phần tử trong vector bằng vector erase c++

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

Cú pháp:

v.erase(p);

Trong đó v là vector ban đầu, và p là trình lặp trỏ đến vị trí cần xóa trong vector.

Trong trường hợp cần chỉ đến vị trí index thứ i trong vector, chúng ta sẽ viết trình lặp p như sau:

v.erase(v.begin() + i);

Trong đó v.begin() chỉ đến vị trí đầu tiên trong vector, và i là index của vị trí phần tử cần chỉ đến.

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main ()

{

    vector<int> v{3, 1, 4, 1, 5};

    v.erase(v.begin() + 1);//Xóa phần tử ở vị trí thứ 1

    dump(v);

    v.erase(v.begin() + 3);//Xóa phần tử ở vị trí thứ 3

    dump(v);

    return 0;

}

3 4 1 5

3 4 1

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

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

Trong đó v là chuỗi ban đầu, iterator\_first và iterator\_last là phạm vi cần xóa được chỉ định trong trình lặp trỏ đến vị trí cần xóa trong vector.

Tương tự như khi xóa 1 phần tử thì cú pháp sử dụng thực tế để xóa các phần tử trong một phạm vi chỉ định trong vector C++ bằng hàm erase sẽ là:

v.erase(v.begin() + start, v.begin() + end);

Trong đó start và end là vị trí index của phạm vi xóa trong vector.

VD

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main ()

{

    vector<int> v{3, 1, 4, 1, 5, 7, 9};

    dump(v);

    //Xóa phần tử trong phạm vi index từ 1 đến 3

    v.erase(v.begin() + 1, v.begin() + 3);

    dump(v);

    //Xóa phần tử trong phạm vi index từ 2 đến 5

    v.erase(v.begin() + 2, v.begin() +5);

    dump(v);

    return 0;

}

3 1 4 1 5 7 9

3 1 5 7 9

3 1

## Sao chép và hoán đổi vector

### Sao chép vector

std::vector<type> vt\_dest(vt\_src);

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

VD

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main()

{

    vector<int> vt\_src{3, 1, 4,5};

    cout << "Orginary vector" << endl;

    dump (vt\_src);

    /\*Sao chép vector\*/

    vector<int> vt\_dest(vt\_src);

    cout << "Copy vector" << endl;

    dump (vt\_dest);

    return 0;

}

Orginary vector

3 1 4 5

Copy vector

3 1 4 5

### Hoán đổi 2 vector

vt1.swap(vt2);

Trong đó vt1 và vt2 là 2 vector cần hoán đổi nội dung cho nhau.

VD

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main()

{

    vector<int> vt1{3, 1, 4, 1, 5};

    vector<int> vt2{9, 8, 7};

    cout << "Before swap" << endl;

    dump(vt1);

    dump(vt2);

    vt1.swap(vt2);

    cout << "After swap" << endl;

    dump(vt1);

    dump(vt2);

}

Before swap

3 1 4 1 5

9 8 7

After swap

9 8 7

3 1 4 1 5

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

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main()

{

    vector<int> vt1{3, 1, 4, 1, 5};

    vector<int> vt2{9, 8, 7};

    cout << "Before swap" << endl;

    dump(vt1);

    dump(vt2);

    swap(vt1,vt2);

    cout << "After swap" << endl;

    dump(vt1);

    dump(vt2);

}

Before swap

3 1 4 1 5

9 8 7

After swap

9 8 7

3 1 4 1 5

## Đảo ngược trong vector

### Đảo ngược trong vector C++ bằng std::reverse

std::reverse(vt.begin(), vt.end());

Trong đó vt là tên vector cần đảo ngược thứ tự các phần tử, còn vt.begin() và vt.end() lần lượt là các trình lặp có tác dụng như con trỏ chỉ đến vị trí đầu tiên và cuối cùng của vector.

Để sử dụng hàm std::reverse, chúng ta cần include header file algorithm vào đầu chương trình:

VD

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& vt)

{

    for (auto x: vt) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main ()

{

    vector<int> vt{3, 1, 4, 2, 5};

    cout << "Before reverse" << endl;

    dump (vt);

    reverse(vt.begin(), vt.end());

    cout << "After reverse" << endl;

    dump (vt);

    return 0;

}

Before reverse

3 1 4 2 5

After reverse

5 2 4 1 3

## Xắp xếp vector

### Sắp xếp vector tăng dần bằng hàm sort vector trong C++

sort vector trong C++ là một hàm thành viên trong class std:algorithm, có tác dụng sắp xếp vector trong C++ theo thứ tự tăng dần hoặc giảm dần. Phép sắp xếp vector bằng hàm sort trong C++ sẽ làm thay đổi vector ban đầu.

Để sử dụng hàm sort, chúng ta cần include header file algorithm.

Cú pháp:

std::sort(v.begin(), v.end() );

Trong đó v là vector cần sắp xếp theo thứ tự tăng dần. Các phương thức begin() và end() được dùng để lấy index đầu tiên và cuối cùng trong vector, tương ứng với phạm vi sẽ tiến hành sắp xếp.

VD

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main()

{

    vector<int> data{3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};

    cout << "Before sort" << endl;

    dump (data);

    /\*Sắp xếp vector theo thứ tự tăng dần\*/

    sort(data.begin(), data.end() );

    cout << "After sort" << endl;

    dump (data);

    return 0;

}

Before sort

3 1 4 1 5 9 2 6 5 3 5

After sort

1 1 2 3 3 4 5 5 5 6 9

### Sắp xếp vector giảm dần bằng hàm sort vector trong C++

Để sắp xếp vector giảm dần, chúng ta sử dụng hàm sort vector trong C++ với cú pháp sau đây:

std::sort(v.begin(), v.end(), std::greater<type>() );

Sự khác biệt duy nhất với cú pháp khi sắp xếp vector theo thứ tự tăng dần bằng hàm sort đó là đối số std::greater<type> có tác dụng chuyển từ sắp xếp tăng dần thành giảm dần, trong đó type là kiểu dữ liệu sử dụng trong vector.

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

void dump(vector<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main()

{

    vector<int> data{3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};

    cout << "Before sort" << endl;

    dump (data);

    sort(data.begin(), data.end(), std::greater<int>() );

    cout << "After sort" << endl;

    dump (data);

    return 0;

}

Before sort

3 1 4 1 5 9 2 6 5 3 5

After sort

9 6 5 5 5 4 3 3 2 1 1

## Tính tổng các phần tử

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

Cú pháp:

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

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

Để sử dụng hàm std::accumulate, chúng ta cần include header file numeric.

VD

#include <iostream>

#include <numeric>

#include <vector>

using namespace std;

int main ()

{

    vector<int> vt{3, 1, 4, 2, 5};

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

    cout << "SUM = " << sum << endl;

    return 0;

}

SUM = 15

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

VD

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    vector<int> vt{3, 1, 4, 2, 5};

    int sum = 0;

    for (auto x: vt) {

        sum += x;

    }

    cout << "SUM = " << sum << endl;

    return 0;

}

SUM = 15

## Vector trống

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

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

vt.empty();

Trong đó vt là vector cần kiểm tra.

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

VD

#include <iostream>

#include <vector>

using namespace std;

int main ()

{

    vector<int> vt;

    if( vt.empty() )

        std::cout << "empty.\n";

    else

        std::cout << "not empty.\n";

    vector<int> dq2{1, 2, 3};

    if( dq2.empty() )

        std::cout << "empty.\n";

    else

        std::cout << "not empty.\n";

}

empty.

not empty.

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

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

vt.clear();

Trong đó vt là vector cần làm trống.

Lưu ý hàm vector clear sẽ làm trống vector chỉ định bằng việc xóa đi tất cả phần tử của nó. Tuy nhiên thì hàm vector clear sẽ không giải phóng bộ nhớ sử dụng cho việc lưu trữ dữ liệu đã dùng.

Trong trường hợp muốn xóa phần tử và giải phóng bộ nhớ, hãy dùng hàm shrink\_to\_fit() để thay thế.

VD

#include <iostream>

#include <vector>

using namespace std;

//Tạo hàm xuất vector

void dump(vector<int>& v)

{

    for (auto x: v) {

        cout << " "<< x ;

    }

    cout << endl;

}

int main ()

{

    vector<int> vt{3, 1, 4};

    dump(vt);

    vt.clear();

    dump(vt);

}

3 1 4

### Giải phóng bộ nhớ trong C++ bằng hàm shrink\_to\_fit

Thông thường một vùng chứa vector có thể được cấp phát nhiều bộ nhớ hơn mức cần thiết để chứa các phần tử hiện tại của nó, nhằm tạo không gian trong mảng động giúp việc xử lý thêm chèn vector nhanh hơn.

Và hàm shrink\_to\_fit được tạo ra nhằm giúp chúng ta khi muốn reset lại vùng bộ nhớ và xóa đi các vùng cấp phát thừa. Lưu ý là hàm chỉ xóa các bộ nhớ cấp phát thừa, chứ không làm giảm kích thước tối thiểu của vùng chứa vector.

Hàm shrink\_to\_fit hay được sử dụng sau hàm clear(), sau khi bạn đã xóa tất cả các phần tử và cần phải giải phóng cả bộ nhớ đã dùng để chứa vector đó.

Cú pháp của hàm shrink\_to\_fit trong C++ như sau:

vt.shrink\_to\_fit();

Trong đó vt là vector cần tinh chỉnh và làm giảm bộ nhớ.

#include <iostream>

#include <vector>

using namespace std;

int main ()

{

    std::vector<int> vt{3, 1, 4};

    vt.clear();

    vt.shrink\_to\_fit();

}

# Tham khảo

Lập trình căn bản:

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