**Bài 1: Vector - Khái niệm - Sử dụng thư viện chuẩn STL cho C/C++**

Chúng ta đã biết trong lập trình C, chúng ta thường sử dụng mảng(array) được biết như là một mảng tĩnh.

Lợi ích của mảng tĩnh là việc truy xuất dễ dàng đến các phần tử trong mảng để thay đổi và chỉnh sửa các giá trị, nhưng không thể nói đến những khó khăn khi làm việc với mảng tĩnh như :

- Phải khai báo độ dài của mảng bằng giá trị cụ thể

- Muốn chèn, xoá, thay đổi độ dài của mảng khá phức tạp và tốn bộ nhớ

Vì thế hôm nay chúng ta sẽ tìm hiểu về vector nhé .

**Vector là gì ?**

Giống như là mảng (array), vector trong C++ là một đối tượng dùng để chứa các đối tượng khác, và các đối tượng được chứa này cũng được lưu trữ một cách liên tiếp trong vector

So với mảng, cách dùng của vector thì linh hoạt hơn nhiều, và đây là 1 số điểm nổi trội của vector so với mảng array :

* Bạn không cần phải khai báo kích thước của mảng ví dụ int A[100]..., vector có thể tự động nâng kích thước lên.
* Nếu bạn thêm 1 phần tử vào vector đã đầy rồi, thì vector sẽ tự động tăng kíck thước của nó lên để dành chỗ cho giá trị mới này.- Vector còn có thể cho bạn biết số lượng các phần tử mà bạn đang lưu trong nó.
* Dùng số phần tử âm vẫn được trong vector ví dụ A-10], A[-3], rất tiện trong việc cài đặt các giải thuật khác.

Dưới đây là ví dụ về Vector là cú pháp thêm phần tử vào vector :

#**include** <iostream>

#**include** <vector>

**using** **namespace** std;

**int** **main**(){

vector<**int**> list;

list.push\_back(1);

list.push\_back(2);

list.push\_back(3);

list.push\_back(4);

list.push\_back(5);

**for**(**int** i = 0 ; i < list.size(); i++){

cout << list[i]<<endl;

}

**return** 0;

}

Ở dòng thứ 2 chúng ta sử dụng thư việc vector để hỗ trợ trong việc thêm các phần tử vào list bằng câu lệnh *list.push\_back(value)*;

**Các ví dụ :**

chúng ta cùng làm qua một số ví dụ nhé :

**1. Tính tổng các phần tử trong list  :**

Chúng ta có một list gồm 5 phần tử [1,2,3,4,5] và tính tổng bằng cách sử dụng con trỏ để duyệt qua các phần tử trong list và tính tổng , code như sau :

#**include** <iostream>

#**include** <vector>

**using** **namespace** std;

**int** **main**(){

vector<**int**> list;

list.push\_back(1);

list.push\_back(2);

list.push\_back(3);

list.push\_back(4);

list.push\_back(5);

**int** sum = 0;

**for**(vector<**int**>::iterator it = list.begin(); it != list.end(); ++it){

sum += \*it;

}

cout << "Tong la : " << sum;

**return** 0;

}

Ở trên, vector<int>::iterator it có nghĩa là chúng ta đã khai báo một con trỏ trong vector và sử dụng con trỏ này để qua mỗi vòng lặp chúng ta lại cộng vào biến sum

sum += \*it : \*it để lấy giá trị của biến con trỏ

**2. Nhập vào mảng gồm n kí tự và sắp xếp mảng theo thứ tự tăng dần :**

#**include** <iostream>

#**include** <vector>

**using** **namespace** std;

**int** **main**(){

**int** n;

cout << "Nhap do dai cua mang : ";

cin >> n;

vector<**int**> list;

**int** index;

//Nhập mảng

**for**(**int** i = 0; i < n; i++){

cin >> index;

list.push\_back(index);

}

//Sắp xếp các phần tử của mảng

**int** idx = 0;

**for**(**int** i = 0; i < list.size()-1; i++){

idx = i;

**for**(**int** j = i + 1; j < list.size(); j++){

**if**(list[idx] > list[j]){

//Hoan vi tang dan

**int** hoanvi = list[idx];

list[idx] = list[j];

list[j] = hoanvi;

}

}

}

// Xuất mảng bằng cách sử dụng biến con trỏ

**for**(vector<**int**>::iterator it = list.begin(); it != list.end(); ++it){

cout << \*it << endl;

}

**return** 0;

}

Bài sau chúng ta sẽ tìm hiểu về một vài hàm thiết yếu của vector trong C++

**Bài 2: Các hàm thường dùng trong Vector - Sử dụng thư viện chuẩn STL cho C/C++**

**Đăng bởi: Admin | Lượt xem: 2895 | Chuyên mục: C/C++**

Để sử dụng Vector ta cần phải khai báo thư viện std:

#**include** <iostream>

#**include** <vector>

**using** **namespace** std;

Sau đó ta sẽ khai báo một vector :

vector<**int**> list ;

Câu lệnh trên có nghĩa là ta đã khai báo một vector kiểu int, bạn có thể thay đổi sang kiểu string, bool, float,...

Bây giờ chúng ta sẽ tìm hiểu một số hàm cơ bản của vector nhé :

- Thêm một phần tử vào sau của vector :

list.push\_pack(value);

- Bỏ đi phần tử cuối cùng

list.pop\_back();

- Bỏ đi tất cả các phần tử của vector

list.clear();

- Lấy ra phần tử đầu tiên của vector

list.front();

- Lấy ra phần tử cuối cùng của vector

list.back();

- Lấy ra phần tử thứ n của vector( đếm từ vị trí 0)

list.at(n);

- Lấy độ dài của vector

list.size();

- Xem phần tử có trống hay không

**list**.**empty**();

- Chỉnh sửa lại độ dài của vector gồm n phần tử

list.resize(n);

**Bài tập :**

1. Cho trước một vector v, hãy xóa phần tử ở vị trí cuối cùng trong vector v.

Ví dụ:

* Với v = [1, 2 ,3, 4], thì vectorPop(v) = [1, 2, 3].
* Với v = [1, 2, 3], thì vectorPop(v) = [1, 2].

std::vector<**int**> **vectorPop**(std::vector<**int**> v)

{

v.pop\_back();

**return** v;

}

2. Cho vector v gồm các số nguyên.

Hãy tính tổng của phần tử đầu tiên và cuối cùng trong vector đó.

Ví dụ:

* Với v = [1, 2, 4, 9] thì VectorFrontBack(v) = 10.  
  Giải thích: 1 + 9 = 10.
* Với v = [1, 3, 4] thì VectorFrontBack(v) = 5.

**int** **vectorFrontBack**(std::vector<**int**> v)

{

**return** v.front() + v.back();

}

3. Cho trước một vector v, và số nguyên dương k (k ≤ v.size())

Hãy xóa k phần tử cuối cùng của vector v.

Ví dụ:

* Với v = [1, 2, 3, 4, 5, 6, 7] và k = 3 thì   
  vectorErase(v) = [1, 2, 3, 4].
* Với v = [1, 2, 3, 4] và k = 2 thì vectorErase(v) = [1, 2].

std::vector<**int**> **vectorErase**(std::vector<**int**> v, **int** k)

{

vector<**int**>:: iterator it = v.begin();

advance(it,v.size()-k);

v.erase(it,v.end());

**return** v;

}

# **Tổng hợp vector trong C++**

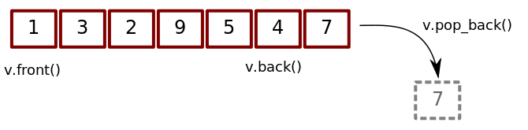
C++ là ngôn ngữ mạnh mẽ nhờ tính linh hoạt, gần gũi với ngôn ngữ máy. Bên cạnh đó còn có khả năng lập trình dựa trên mẫu có sẵn (template). Sức mạnh của C++ nằm ở STL (Standard Template Library) – một thư viện template dành cho [C++](https://topdev.vn/viec-lam-it/c-plus-kt19) tổng hợp các cấu trúc dữ liệu cũng như giải thuật và các vector trong C++ được xây dựng một cách tổng quát cho nhiều trường hợp.

Bộ thư viện này thực hiện toàn bộ các hoạt động ra vào của dữ liệu (iostream), quản lý mảng (vector) và hầu hết các tính năng của các cấu trúc dữ liệu cơ bản như (stack, queue, map, set…). Ngoài ra, STL còn bao gồm các thuật toán cơ bản: tính tổng, tìm min, max, sắp xếp (với đa dạng thuật toán sắp xếp), thay thế các phần tử, tìm kiếm (tìm kiếm bình thường và tìm kiếm dười dạng nhị phân). Toàn bộ những tính năng trên đều được cung cấp dưới dạng template nên việc sử dụng chúng cực kỳ tiện lợi và phù hợp với nhiều tình huống. Nhờ vậy, STL làm cho ngôn ngữ C++ trở nên “xịn xò” hơn.

Nhưng nói đến STL thì sẽ là bài tập rất dài nên hôm nay mình sẽ chỉ đề cập đến vector thôi.

## **Vậy Vector trong C++ là gì?**

Không giống như [array (mảng)](https://topdev.vn/blog/4-ung-dung-cua-array-from/), chỉ một số giá trị nhất định có thể được lưu trữ dưới một tên biến duy nhất. Vector trong C++ giống dynamic array (mảng động) nhưng có khả năng tự động thay đổi kích thước khi một phần tử được chèn hoặc xóa tùy thuộc vào nhu cầu của tác vụ được thực thi, với việc lưu trữ của chúng sẽ được vùng chứa tự động xử lý. Các phần tử vector được đặt trong contiguous storage (bộ nhớ liền kề) để chúng có thể được truy cập và duyệt qua bằng cách sử dụng iterator.



### Vì sao nên dùng Vector

Nếu bạn đã phát chán việc quản lý mảng động qua [con trỏ trong C++](https://topdev.vn/blog/con-tro-va-ham-pointer-function/) hay chán phải tạo mảng mới, copy các phần tử qua mảng mới, rồi lại xóa mảng cũ mỗi khi bạn muốn resize kích thước mảng động trong C++. Thật thừa thải, tốn thời gian và đó là thời điểm ta nhận ra C++ còn có vector.

### Một số điểm nổi trội của Vector

– Bạn không cần phải khai báo kích thước của mảng ví dụ int A[100]…, vì vector có thể tự động nâng kích thước lên.

– Nếu bạn thêm 1 phần tử vào vector đã đầy rồi, thì nó sẽ tự động tăng kích thước của nó lên để dành chỗ cho giá trị mới này.

– Vector còn giúp cho bạn biết số lượng các phần tử mà bạn đang lưu trong đó.

– Dùng số phần tử âm vẫn được trong vector ví dụ A[-6], A[-9], rất tiện trong việc cài đặt các giải thuật.

**[Hướng dẫn lấy Date và Time trong C++](https://topdev.vn/blog/date-va-time-trong-c/" \t "_blank)**

**[Tổng hợp vector trong C++](https://topdev.vn/blog/vector-trong-c/" \t "_blank)**

Giới thiệu literals và operators”]

### Vector có thứ tự trong C++ không?

Không có vector nào không được sắp xếp trong C++. Các phần tử vector được đặt trong bộ nhớ liền kề để chúng có thể được truy cập và di chuyển qua các iterator. Trong vector, dữ liệu được chèn vào cuối. Việc chèn một phần tử vào cuối sẽ mất thời gian chênh lệch, vì đôi khi có thể cần mở rộng vector. Việc xóa phần tử cuối cùng chỉ mất thời gian không đổi vì không xảy ra thay đổi kích thước. Chèn và xóa ở đầu hoặc giữa vector là tuyến tính theo thời gian.

## **Các vector được lưu trữ trong C++ như thế nào?**

Để tạo một vector, bạn cần thực hiện theo cú pháp dưới đây:

**Cú pháp:**

#include <vector>

//...

vector<object\_type> variable\_name;

**Ví dụ:**

#include <vector>

int main()

{

    std::vector<int> my\_vector;

}

Vậy là chúng ta đã có một vector với mỗi phần tử có kiểu dữ liệu (object\_type) là int. Sau đó bạn có thể gắn giá trị cho vector như này:

vector<int> my\_vector = {1,2,3,4,5}

Hoặc bạn cũng có thể tạo một vector rồi gán giá trị của một vector khác cho nó bằng cách:

vector<int> my\_vector = {1,2,3,4,5};

vector<int> your\_vector = my\_vector;

### Cơ chế ngăn chặn rò rỉ bộ nhớ của Vector

Khi một biến **vector** rời khỏi phạm vi đoạn code mà chương trình đang chạy, nó sẽ tự động giải phóng những phần bộ nhớ mà nó kiểm soát (nếu cần). Điều này không chỉ tiện dụng (vì bạn không cần tự tay giải phóng bộ nhớ), mà nó còn giúp ngăn ngừa lỗi rò rỉ bộ nhớ (memory leaks).

**Xem hàm dưới đây:**

void doSomething(bool earlyExit)

{

int \*array = new int[3]{ 1, 3, 2 };

if (earlyExit) // thoát khỏi hàm

return;

delete[] array; // trường hợp hàm thoát sớm, array sẽ không bị xóa

}

Nếu biến earlyExit được gán là true, mảng array sẽ không bao giờ được giải phóng, và bộ nhớ sẽ bị rò rỉ.

Tuy nhiên, nếu biến array là một vector, điều này sẽ không xảy ra, bởi vì bộ nhớ sẽ được giải phóng ngay sau khi biến array nằm ngoài phạm vi đoạn code mà chương trình đang chạy (bất kể hàm có bị thoát ra sớm hay không). Điều này làm cho vector an toàn hơn nhiều so với việc bạn phải tự chú ý đến việc giải phóng bộ nhớ.

### Vector tự ghi nhớ độ dài của mình

Không giống như mảng động được tích hợp sẵn của C++, cái mà không biết được độ dài của mảng mà nó đang trỏ tới là bao nhiêu, **std::vectors** tự theo dõi độ dài của chính nó. Chúng ta có thể lấy được độ dài của vector thông qua hàm **size()**:

#include <iostream>

#include <vector>

void printLength(const std::vector<int>& array)

{

  std::cout << "The length is: " << array.size() << '\n';

}

int main()

{

  std::vector array { 9, 7, 5, 3, 1 };

  printLength(array);

  return 0;

}

**Output:**

The length is: 5

Tương tự với array, hàm size() sẽ trả về một giá trị thuộc kiểu nested type (kiểu dữ liệu lồng) là size\_type, nó là một số nguyên không dấu.

**[Refresh trang web với Selenium webdriver](https://topdev.vn/blog/refresh-trang-web-voi-selenium-webdriver/" \t "_blank)**

### Các hàm của Vectors trong C ++

Vector trong STL cung cấp cho chúng ta nhiều chức năng hữu ích khác nhau.

1. **Modifiers**  
2. **Iterators**  
3. **Capacity**4. **Element access**

#### ****Modifiers****

1. **push\_back()**: Hàm đẩy một phần tử vào vị trí sau cùng của vector. Nếu kiểu của đối tượng được truyền dưới dạng tham số trong push\_back() không giống với kiểu của vector thì sẽ bị ném ra.

ten-vector.push\_back(ten-cua-phan-tu);

2. **assign()**: Nó gán một giá trị mới cho các phần tử vector bằng cách thay thế các giá trị cũ.

ten-vector.assign(int size, int value);

3. **pop\_back()**: Hàm pop\_back () được sử dụng để xóa đi phần tử cuối cùng một vector.

ten-vector.pop\_back();

4. **insert()**: Hàm này chèn các phần tử mới vào trước phần tử trước vị trí được trỏ bởi vòng lặp. Chúng ta cũng có thể chuyển một số đối số thứ ba, đếm số lần phần tử được chèn vào trước vị trí được trỏ.

ten-vector.insert(position, value);

5. **erase()**: Hàm được sử dụng để xóa các phần tử tùy theo vị trí vùng chứa

ten-vector.erase(position);

ten-vector.erase(start-position, end-position);

6. **emplace()**: Nó mở rộng vùng chứa bằng cách chèn phần tử mới vào

ten-vector.emplace(ten-vector.position, element);

7. **emplace\_back()**: Nó được sử dụng để chèn một phần tử mới vào vùng chứa vector, phần tử mới sẽ được thêm vào cuối vector

ten-vector.emplace\_back(value);

8. **swap()**: Hàm được sử dụng để hoán đổi nội dung của một vector này với một vector khác cùng kiểu. Kích thước có thể khác nhau.

ten-vector-1.swap(ten-vector-2);

9. **clear()**: Hàm được sử dụng để loại bỏ tất cả các phần tử của vùng chứa vector.

ten-vector.clear();

**Ví dụ:**

/ Modifiers in vector

#include <bits/stdc++.h>

#include <vector>

using namespace std;

int main()

{

    // Assign vector

    vector<int> vec;

    // fill the array with 12 seven times

    vec.assign(7, 12);

    cout << "The vector elements are: ";

    for (int i = 0; i < 7; i++)

        cout << vec[i] << " ";

    // inserts 24 to the last position

    vec.push\_back(24);

    int s = vec.size();

    cout << "nThe last element is: " << vec[s - 1];

     // prints the vector

    cout << "nThe vector elements after push back are: ";

    for (int i = 0; i < vec.size(); i++)

    cout << vec[i] << " ";

    // removes last element

    vec.pop\_back();

    // prints the vector

    cout << "nThe vector elements after pop\_back are: ";

    for (int i = 0; i < vec.size(); i++)

        cout << vec[i] << " ";

    // inserts 10 at the beginning

    vec.insert(vec.begin(), 10);

    cout << "nThe first element after insert command is: " << vec[0];

    // removes the first element

    vec.erase(vec.begin());

    cout << "nThe first element after erase command is: " << vec[0];

    // inserts at the beginning

    vec.emplace(vec.begin(), 5);

    cout << "nThe first element emplace is: " << vec[0];

    // Inserts 20 at the end

    vec.emplace\_back(20);

    s = vec.size();

    cout << "nThe last element after emplace\_back is: " << vec[s - 1];

    // erases the vector

    vec.clear();

    cout << "nVector size after clear(): " << vec.size();

    // two vector to perform swap

    vector<int> obj1, obj2;

    obj1.push\_back(2);

    obj1.push\_back(4);

    obj2.push\_back(6);

    obj2.push\_back(8);

    cout << "nnVector 1: ";

    for (int i = 0; i < obj1.size(); i++)

        cout << obj1[i] << " ";

    cout << "nVector 2: ";

    for (int i = 0; i < obj2.size(); i++)

        cout << obj2[i] << " ";

    // Swaps obj1 and obj2

    obj1.swap(obj2);

    cout << "nAfter Swap nVector 1: ";

    for (int i = 0; i < obj1.size(); i++)

        cout << obj1[i] << " ";

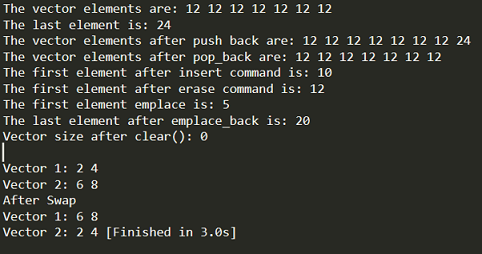
    cout << "nVector 2: ";

    for (int i = 0; i < obj2.size(); i++)

        cout << obj2[i] << " ";

}

**Output:**



Bạn có thể thấy cách sử dụng các hàm thuộc nhóm **Modifiers** mà chúng ta đã nghiên cứu ở trên qua ví dụ thực tiễn.

**[Lập trình viên mobile cần học những gì?](https://topdev.vn/blog/lap-trinh-vien-mobile-can-hoc-nhung-gi/" \t "_blank)**

#### ****Iterators****

1. **begin()**: đặt iterator đến phần tử đầu tiên trong vectorten-vector.begin();
2. **end()**: đặt iterator đến sau phần tử cuối cùng trong vectorten-vector.end();
3. **rbegin()**: đặt reverse iterator (trình lặp đảo) đến phần tử cuối cùng trong vector (reverse begin). Nó di chuyển từ phần tử cuối cùng đến phần tử đầu tiên

ten-vector.rbegin();

1. **rend()**: đặt reverse iterator (trình lặp đảo) đến phần tử đầu tiên trong vector (reverse end)

ten-vector.rend();

1. **cbegin()**: đặt constant iterator (trình vòng lặp) đến phần tử đầu tiên trong vectorten-vector.cbegin();
2. **cend()**: đặt constant iterator (trình vòng lặp) đến phần tử cuối cùng trong vectorten-vector.cend();
3. **crbegin()**: đặt constant reverse iterator (trình lặp đảo liên tục) đến phần tử cuối cùng trong vector (reverse begin). Nó di chuyển từ phần tử cuối cùng đến phần tử đầu tiênten-vector.cbregin();
4. **crend()**: đặt constant reverse iterator (trình lặp đảo liên tục) đến phần tử đầu tiên trong vector (reverse end)ten-vector.crend();

**Ví dụ 1:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    vector<int> vec1;

    for (int i = 1; i <= 10; i++)

        vec1.push\_back(i);

    cout << "Understanding begin() and end() function: " << endl;

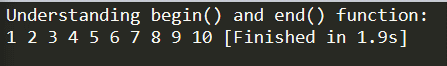
    for (auto i = vec1.begin(); i != vec1.end(); ++i)

        cout << \*i << " ";

    return 0;

}

**Output:**



Trong ví dụ trên, chúng ta có thể thấy cách sử dụng hàm begin() và end(). Đầu tiên, chúng ta xác định một vector là vec1, chúng đẩy lùi các giá trị trong đó từ 1 đến 10 bằng cách sử dụng vòng lặp for. Sau đó, chúng in các giá trị của các vector của chúng tôi bằng cách sử dụng vòng lặp for, chúng tôi sử dụng hàm begin() và end() để chỉ định điểm đầu và điểm cuối của vòng lặp for của chúng tôi.

**Ví dụ 2:**

// C++ program to illustrate the

// iterators in vector

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    vector<int> g1;

   for (int i = 1; i <= 5; i++)

       g1.push\_back(i);

   cout << "Output of begin and end: ";

for (auto i = g1.begin(); i != g1.end(); ++i)

       cout << \*i << " ";

   cout << "\nOutput of cbegin and cend: ";

   for (auto i = g1.cbegin(); i != g1.cend(); ++i)

       cout << \*i << " ";

   cout << "\nOutput of rbegin and rend: ";

   for (auto ir = g1.rbegin(); ir != g1.rend(); ++ir)

       cout << \*ir << " ";

   cout << "\nOutput of crbegin and crend : ";

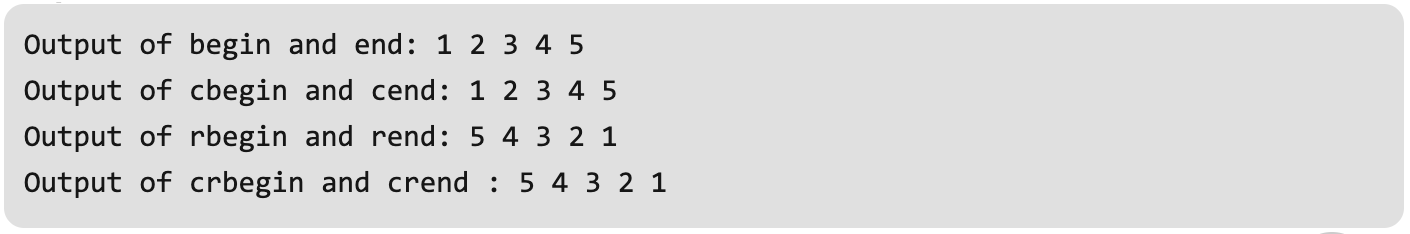
   for (auto ir = g1.crbegin(); ir != g1.crend(); ++ir)

       cout << \*ir << " ";

   return 0;

}

**Output:**



**Ví dụ 3:**

// CPP program to illustrate working of crbegin()

// crend()

#include <iostream>

#include <vector>

using namespace std;

int main ()

{

// initializing vector with values

 vector<int> vect = {10, 20, 30, 40, 50};

// for loop with crbegin and crend

 for (auto i = vect.crbegin(); i != vect.crend(); i++)

   cout << ' ' << \*i;  //printing results

 cout << '\n';

 return 0;

}

**Output:**

50 40 30 20 10

#### ****Capacity****

1. **size()**: hàm sẽ trả về số lượng phần tử đang được sử dụng trong vectorten-vector.size();
2. **max\_size()**: hàm trả về số phần tử tối đa mà vector có thể chứaten-vector.max\_size();
3. **capacity()**: hàm trả về số phần tử được cấp phát cho vector nằm trong bộ nhớten-vector.capacity();
4. **resize(n)**: Hàm này thay đổi kích thước vùng chứa để nó chứa đủ n phần tử. Nếu kích thước hiện tại của vector lớn hơn n thì các phần tử phía sau sẽ bị xóa khỏi vector và ngược lại nếu kích thước hiện tại nhỏ hơn n thì các phần tử bổ sung sẽ được chèn vào phía sau vectorten-vector.resize(int n, int value);
5. **empty()**: Trả về liệu vùng chứa có trống hay không, nếu trống thì trả về True, nếu có phần tử thì trả về Falseten-vector.empty();
6. **shrink\_to\_fit()**: Giảm dung lượng của vùng chứa để phù hợp với kích thước của nó và hủy tất cả các phần tử vượt quá dung lượngten-vector.shrink\_to\_fit();
7. **reserve(n)**: hàm cấp cho vector số dung lượng vừa đủ để chứa n phần tửten-vector.reserve(n);

**Ví dụ:**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

    vector<int> vec1;

    for (int i = 1; i <= 10; i++)

        vec1.push\_back(i);

    cout << "Size of our vector: " << vec1.size();

    cout << "nCapacity of our vector: " << vec1.capacity();

    cout << "nMax\_Size of our vector: " << vec1.max\_size();

    // resizes the vector size to 4

    vec1.resize(4);

    // prints the vector size after resize()

    cout << "nSize of our vector after resize: " << vec1.size();

    // checks if the vector is empty or not

    if (vec1.empty() == false)

        cout << "nVector is not empty";

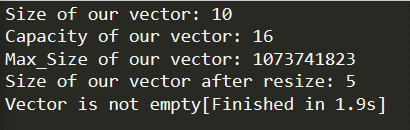
    else

        cout << "nVector is empty";

    return 0;

}

**Output:**

****

**Chú ý:** Khi thay đổi kích thước mảng vector, các giá trị của phần tử hiện có cần giữ nguyên, thì các phần tử mới được khởi tạo bằng giá trị mặc định của kiểu dữ liệu mảng.

Chúng ta có thể thấy cách hoạt động của các hàm capacity như đã thảo luận ở trên.

#### ****Element access****

1. **at(g)**: Trả về một tham chiếu đến phần tử ở vị trí ‘g’ trong vectorten-vector.at(position);
2. **data()**: Trả về một con trỏ trực tiếp đến (memory array) bộ nhớ mảng được vector sử dụng bên trong để lưu trữ các phần tử thuộc sở hữu của nóten-vector.data();
3. **front()**: hàm dùng để lấy ra phần tử đầu tiên của vectorten-vector.front();
4. **back()**: hàm dùng để lấy ra phần tử cuối cùng của vectorten-vector.back();

**Ví dụ:**

// C++ program to illustrate the

// element accesser in vector

#include <bits/stdc++.h>

using namespace std;

int main()

{

    vector<int> g1;

    for (int i = 1; i <= 10; i++)

        g1.push\_back(i \* 10);

    cout << "\nReference operator [g] : g1[2] = " << g1[2];

    cout << "\nat : g1.at(4) = " << g1.at(4);

    cout << "\nfront() : g1.front() = " << g1.front();

    cout << "\nback() : g1.back() = " << g1.back();

    // pointer to the first element

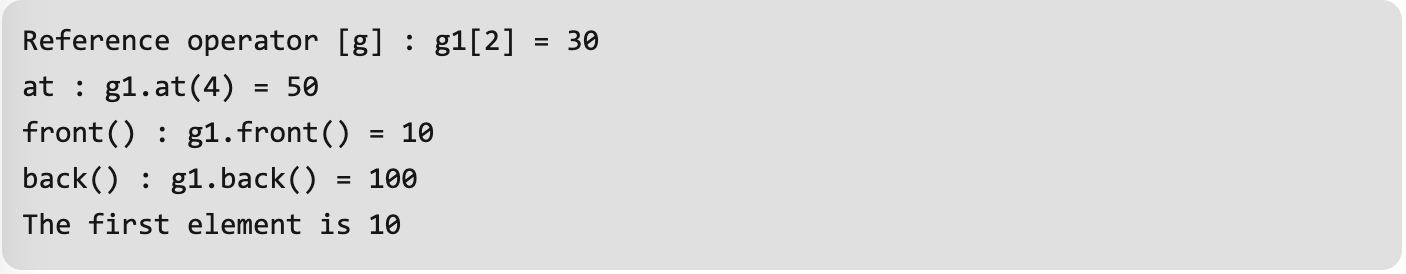
    int\* pos = g1.data();

    cout << "\nThe first element is " << \*pos;

    return 0;

}

**Output:**



## **Thực hiện nén nhiều giá trị bools**

Vector còn có một [thủ thuật hay ho](https://topdev.vn/blog/bookmark-vai-thu-thuat-giup-ban-lam-chu-mot-ngon-ngu-lap-trinh/) khác là sử dụng một cài đặt đặc biệt dành cho vector kiểu bool mà nó có thể nén 8 giá trị booleans vào trong chỉ một byte (amazing!!) Quá trình này đã được cài sẵn, tất cả những gì bạn cần chỉ là sử dụng nó như một tính năng hỗ trợ khi lập trình. Thủ thuật này cũng sẽ không ảnh hưởng gì đến cách mà bạn sử dụng vector.

#include <vector>

#include <iostream>

int main()

{

  std::vector<bool> array { true, false, false, true, true };

  std::cout << "The length is: " << array.size() << '\n';

  for (int i : array)

      std::cout << i << ' ';

    std::cout << '\n';

    return 0;

}

**Output:**

The length is: 5

1 0 0 1 1

## **Kết**

Với những cú pháp mẫu và ví dụ thực tiễn trên, chúng ta kết thúc phần tìm hiểu về vector trong C++. Tôi hy vọng bạn đã phân biệt được các hàm khác nhau của vector và nắm được cách hoạt động của từng hàm. Vì vector hỗ trợ rất tốt trong việc thao tác với mảng động, đảm bảo an toàn và dễ dàng hơn. Bạn nên sử dụng vector trong hầu hết các trường hợp đụng tới mảng động.

<https://topdev.vn/blog/vector-trong-c/>

# **Vector trong C++**

4426

**Bài viết được sự cho phép của tác giả Khiêm Lê**

## **Vector là gì và tại sao nên dùng vector?**

Bạn đã bao giờ phát chán việc quản lý mảng động qua con trỏ trong C++ chưa? Hay việc mỗi khi bạn muốn resize kích thước mảng động trong C++, bạn phải tạo mảng mới, copy các phần tử qua mảng mới, rồi lại xóa mảng cũ đi. Quá phiền phức, tốn quá nhiều thời gian, vì thế mà C++ còn cung cấp cho chúng ta một kiểu nữa đó chính là kiểu vector. Vậy thì vector là gì?

**[Bảng băm trong C++](https://topdev.vn/blog/bang-bam-trong-c/" \t "_blank)**

**[C Token là gì? Cú pháp trong lập trình C/C++](https://topdev.vn/blog/c-token-la-gi-cu-phap-trong-lap-trinh-c-c/" \t "_blank)**

Vector là một chuỗi các phần tử có cùng kiểu dữ liệu, cũng giống như mảng bình thường trong C++. Vậy thì tại sao phải dùng nó?

Đầu tiên, vector có thể tự tăng kích thước của nó mỗi khi ta thực hiện thêm một phần tử vào vector. Thứ 2, vector có thể tự giải phóng bộ nhớ khi ta thực hiện xong đoạn code và thoát ra khỏi scope chứa vector đó, việc này nhầm tránh rò rỉ bộ nhớ khi ta quên delete[] như con trỏ. Thứ ba là vector cung cấp các hàm cần thiết để chúng ta cho thể thao tác với mảng một cách dễ dàng.

## **Cú pháp khai báo và các hàm cơ bản**

Với quá nhiều lợi ích của vector thì việc gì chúng ta phải xử dụng mảng động để quản lý cho mất công đúng không nào! Hãy cùng tìm hiểu về cách sử dụng vector ngay sau đây.

### Khai báo vector

Để sử dụng vector bạn cần thêm thư viện vector trước. Cú pháp khai báo một vector như sau:

#include <vector>

//...

vector<[kiểu\_dữ\_liệu]> tên\_vector;

Vậy là chúng ta đã có một vector với mỗi phần tử có kiểu dữ liệu là [kiểu\_dữ\_liệu]. Bạn có thể gán giá trị trực tiếp cho vector ngay khi khởi tạo như ví dụ sau:

vector<int> arr = {1, 2, 3};

// tạo một vector có ba phần tử là 1, 2, 3

Bạn cũng có thể khởi tạo một vector và gán giá trị một vector khác cho nó như sau:

vector<int> A = {1, 2, 3};

vector<int> B = A;

// B cũng sẽ mang giá trị của A là {1, 2, 3}

### Truy xuất phần tử của vector

Truy xuất phần tử trong vector có thể được thực hiện qua hai cách là qua phương thức at() hoặc qua dấu [] như mảng thông thường.

vector<int> arr = {1, 2, 3};

cout << arr.at(2); // được 3 vì vị trí số 2 chính là 3

// Hoặc

cout << arr[2]; // ta cũng được tương tự như trên

// gán phần tử

arr.at(0) = 0; // gán lại phần tử 0 bằng 0

// hoặc

arr[1] = 9; // gán lại phần tử 1 bằng 7

Cả hai cách đều có công dụng như nhau vậy thì sự khác nhau giữa chúng là gì? Chính là khả năng kiểm tra phạm vi. Đối với phương thức at(), bạn có thể dùng try catch để bắt lỗi như sau:

vector<int> arr = {1, 2, 3};

try

{

cout << arr.at(-1);

}

catch (const std::out\_of\_range& oor)

{

std::cerr << "Out of Range error: " << oor.what() << '\n';

}

### Lấy phần tử đầu và cuối của vector

Để lấy phần tử đầu và cuối ta sử phương thức hàm font() và back().

vector<int> arr = {1, 2, 3};

cout << arr.front(); // được phần tử đầu là 1

cout << arr.back(); // được phần tử cuối là 3

### Thêm phần tử vào cuối vector

Để thêm phần tử vào cuối vector ta sử dụng phương thức push\_back().

vector<int> arr = {1, 2, 3};

arr.push\_back(4);

// Mảng sau khi thêm là {1, 2, 3, 4}

Lưu ý sau khi thêm vào thì size của vector sẽ tăng lên 1 và nếu size vượt quá capacity thì capacity cũng sẽ tự động được tăng lên. (size và capacity sẽ được trình bày bên dưới).

### Loại bỏ phần tử cuối vector

Để loại bỏ đi phần tử cuối cùng của vector, ta sử dụng phương thức pop\_back().

vector<int> arr = {1, 2, 3};

arr.pop\_back(4);

// Mảng sau khi loại bỏ phần tử cuối là {1, 2, 3}

Lưu ý sau khi loại bỏ phần tử cuối thì size sẽ tự giảm xuống 1 nhưng capacity không thay đổi (size và capacity sẽ được trình bày bên dưới).

### Số phần tử của vector

Để biết được số phần tử của vector ta sử dụng phương thức size().

vector<int> arr = {1, 2, 3};

cout << arr.size(); // ta sẽ được 3 vì vector có 3 phần tử

### Kích thước của vector

Để biết được kích thước của vector ta dùng phương thức capacity().

vector<int> arr = {1, 2, 3};

cout << arr.capacity(); // ta sẽ được 3 vì vector có kích thước là 3 phần tử

Có nhiều bạn bị nhầm giữa số lượng phần tử (size) và kích thước vector (capacity). Kích thước là số lượng phần tử nhiều nhất mà vector đó từng chứa còn số lượng phần tử là số lượng phần tử mà vector đó hiện tại đang chứa. Các bạn có thể xem ví dụ sau để hiểu thêm:

vector<int> arr = {1, 2, 3};

cout << arr.size(); // được 3 vì vector đang chứa 3 phần tử

cout << arr.capacity(); // được 3 vì vector chứa nhiều nhất là 3 phần tử

arr.pop\_back(); // lấy đi phần tử cuối

cout << arr.size(); // được 2 vì vector đang chứa 2 phần tử

cout << arr.capacity(); // vẫn được 3 vì vector này từng chứa nhiều nhất 3 phần tử

### Thay đổi số phần tử của vector

Bạn có thể dùng phương thức resize() để thay đổi số phần tử của vector.

vector<int> arr = {1, 2, 3};

cout << arr.size(); // được 3 vì vector đang giữ 3 phần tử

arr.resize(1);

cout << arr.size(); // được 1 vì vector đã được resize về 1

Lưu ý là sau khi resize thì vector sẽ chỉ lấy đủ số phần tử đã set còn tất cả những phần tử sau sẽ bỏ đi.

vector<int> arr = {1, 2, 3};

cout << arr.size(); // được 3 vì vector đang giữ 3 phần tử

arr.resize(1);

cout << arr.size(); // được 1 vì vector đã được resize về 1

// Xét ví dụ sau

cout << arr.at(2);

cout << arr[2];

// cả hai đều lỗi vì truy cập phần tử lớn hơn size của vector

### Xóa hết phần tử của vector

Để xóa tất cả các phần tử của vector ta dùng phương thức clear(). Sau khi xóa thì size cũng sẽ trở về 0.

vector<int> arr = {1, 2, 3};

cout << arr.size(); // được 3

arr.clear();

cout << arr.size(); // được 0 vì đã xoá hết phần tử của vector

## **Lời kết**

Trên đây là bài viết vector trong C++ của mình, nếu có sai sót dì các bạn có thể comment bên dưới video để giúp mình sửa sai nha. Cảm ơn các bạn đã theo dõi bài viết!

https://topdev.vn/blog/vector-trong-c-2/