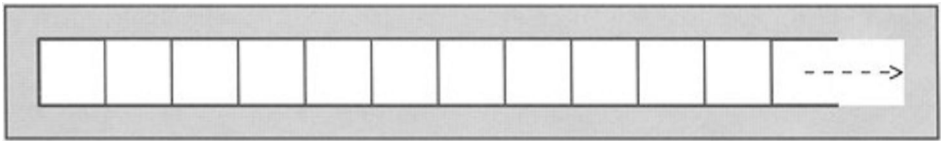
SỬ DỤNG STL VECTOR

I.Giới thiệu :

Lớp mảng động **vector<T>** có sẵn trong thư viện chuẩn STL của C++ cho phép định nghĩa một mảng động các phần tử kiểu T, vector có các tính chất sau:

- Không cần phải khai báo kích thước của mảng vector có thể tự động cấp phát bộ nhớ, bạn sẽ không phải quan tâm đến quản lý kích thước của nó.
- Vector còn có thể cho bạn biết số lượng các phần tử mà bạn đang lưu trong nó.
- Vector có các phương thức của stack.
- Hỗ trợ tất cả các thao tác cơ bản như chèn ,xóa, sao chép ..

II.Vì sao dùng vector :



Kiểu vector có thể coi là kiểu mảng trong lập trình C truyền thống. Mảng là tập hợp các giá trị cùng kiểu, được sắp xếp nối tiếp nhau. Các phần tử của mảng có thể được truy cập ngẫu nhiên qua chỉ số.
Vấn đề đặt ra: Nếu vector là mảng thì tại sao lại phải sử dụng vector khi bạn đã quá quen thuộc với mảng? Xin phân tích một số nhược điểm sau của mảng:

- Nếu bạn sử dụng mảng tĩnh: Mảng này luôn được khai báo với kích thước tối đa mà bạn có thể dùng dẫn đến tốn nhiều vùng nhớ thừa.
- Nếu bạn sử dụng mảng động: Bạn phải xin cấp phát bộ nhớ, làm việc với con trỏ. Con trỏ là khái niệm hay trong C, C++, nhưng nó là nguyên nhân của rất nhiều rắc rối trong lập trình.
- Không thuận tiện trong việc truyền tham số kiểu mảng vào hàm hay trả lại kiểu mảng từ hàm.
- Nhược điểm quan trọng nhất: Nếu bạn sử dụng mảng vượt chỉ số vượt quá kích thước đã khai báo, C++ sẽ không thông báo lỗi, điều này dẫn đến lỗi dây chuyền do các lệnh lỗi đã tác động đến các biến khác trong chương trình (trong Pascal bạn có thể kiểm tra tràn chỉ số mảng bằng dẫn biên dịch range check).

Vector là một container cung cấp khả năng sử dụng mảng mềm dẻo, có kiểm soát range check khi cần thiết, với kích thước tùy ý (mà không cần phải sử dụng con trỏ). Ngoài ra vector cho phép bạn chèn thêm hoặc xóa đi một số phần tử chỉ bằng 1 lệnh (không phải sử dụng vòng lặp như đối với mảng).

III.Cú pháp :

Để có thể dùng vector thì bạn phải thêm 1 header #include <vector> và phải có using std::vector; vì vector được định nghĩa trong STL(Standard Template Library).

Cú pháp của vector cũng rất đơn giản ví dụ :

```
vector<int> A ;
```

Câu lệnh trên định nghĩa 1 vector có kiểu int. Chú ý kiểu của vector được để trong 2 cái ngoặc nhọn. Vì kích thước của vector có thể nâng lên, cho nên không cần khai báo cho nó có bao nhiêu phần tử cũng được, hoặc nếu thích khai báo thì bạn cũng có thể khai báo như sau :

```
vector<int> A(10);
```

Câu lệnh trên khai báo A là 1 vector kiểu int có 10 phần tử. Tuy nhiên như đã nói ở trên, mặc dù size = 10, nhưng khi bạn add vào thì nó vẫn cho phép như thường.

Cũng có thể khởi tạo cho các phần tử trong vector bằng cú pháp đơn giản như sau :

```
vector<int> A(10, 2);
```

Trong câu lện trên thì 10 phần tử của vector A sẽ được khởi tạo bằng 2.

Đồng thời ta cũng có thể khởi tạo cho 1 vector sẽ là bản sao của 1 hoặc 1 phần vector khác, ví dụ :

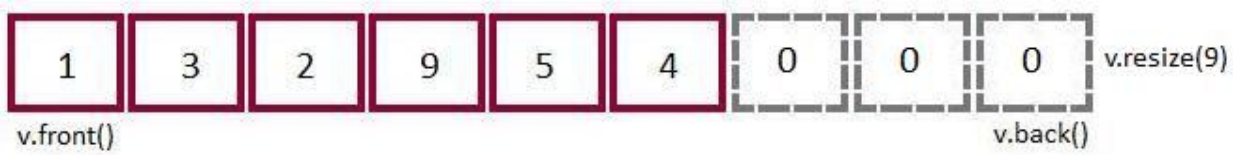
```
vector<int> A(10,2);
vector<int> B(A);
vector<int> C(A.begin(), A.begin() + 5 );
```

Để hiểu rõ hơn về vector, bạn hãy theo dõi ví dụ sau:

```
#include <iostream>     // Thư viện iostream phục vụ ghi dữ liệu ra màn h.nh
#include <vector>       // Thư viện vector, sử dụng kiểu vector
#include <conio.h>      // Thư viện conio (sử dụng hàm getchar() để dừng ct)
using namespace std;  // Sử dụng namespace std
int main()
{
        vector<int> V(3);          // V kiểu vector số nguyên (sử dụng giống mảng int[3])
        V[0] = 5;                  // Gán giá trị cho các phần tử của biến V
        V[1] = 6;                  // Sử dụng dấu móc [] hoàn toàn giống với mảng
        V[2] = 7;
        for (int i=0; i<V.size(); i++)      // Ghi giá trị các phần tử của V ra màn h.nh
                cout << V[i] << endl;       // Nếu sử dụng mảng, bạn phải có biến lưu kích thước
        getchar();                 // Dừng chương trình để xem kết quả
}
```

Ví dụ trên cho bạn thấy việc sử dụng vector rất đơn giản, hoàn toàn giống với mảng nhưng bộ nhớ được quản lý tự động, bạn không phải quan tâm đến giải phóng các vùng bộ nhớ đã xin cấp phát.

Trường hợp xác định kích thước mảng khi chương trình đang chạy, chúng ta dùng hàm dựng mặc định *(default constructor)* để khai báo mảng chưa xác định kích thước, sau đó dùng phương thức **resize()** để xác định kích thước của mảng khi cần. Chương trình sau đây nhập vào n từ (word) mỗi từ là một chuỗi kiểu string:




```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
int main()
{
        int iWordNum;
        vector<string> arrWords;
        cout << "Enter number of words = ";
        cin >> iWordNum;
        arrWords.resize(iWordNum);
        for (int i = 0; i < arrWords.size(); i++)
        {
                cout << "Enter word " << i << " = ";
                cin >> arrWords[i];
        }
        cout << "After entering data..." << endl;
        for (int i = 0; i < arrWords.size(); i++)
                cout << arrWords[i] << endl;
}
```

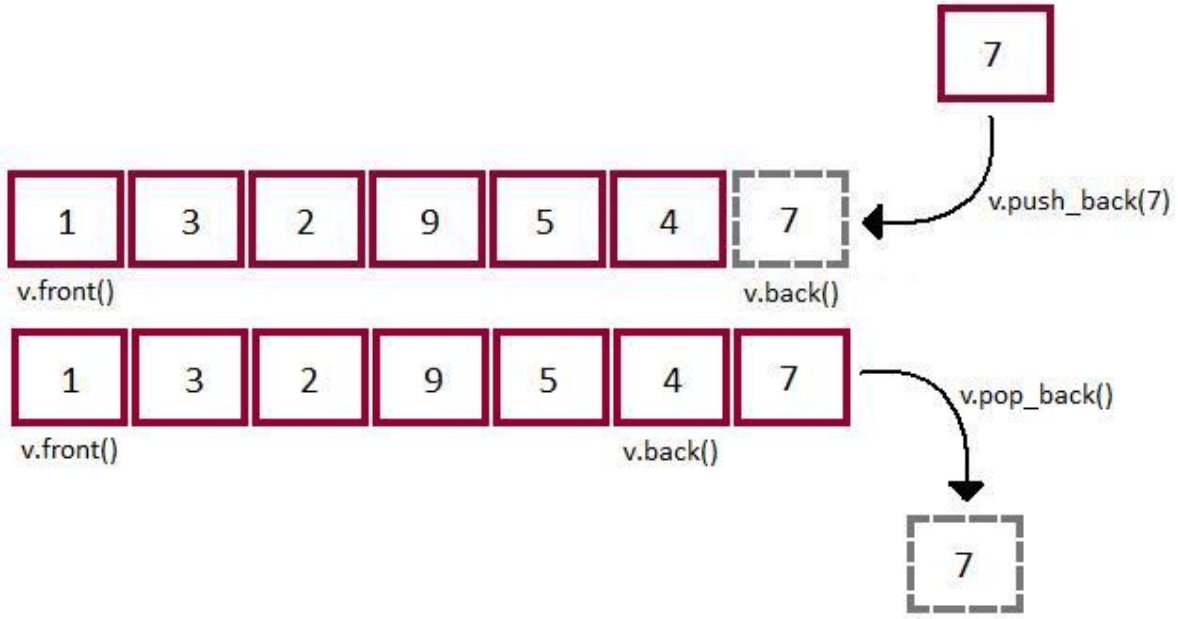
Output

```
Enter number of words = 3
Enter word 1 = hello
Enter word 1 = c++'s
```

```
Enter word 1 = world
After entering data...
hello
c++'s
world
Press any key to continue . . .
```

IV.Các phương thức:

Các phương thức của stack: push_back() và pop_back()




```
#include <iostream>
#include <vector>
using namespace std;
int main()
 {
	int i;
	vector<int> V;
	for (i=0; i<5; i++) // Lặp 5 lần, mỗi lần đưa thêm 1 số vào vector
		V.push_back(i); // Như vậy, vector có thể được sử dụng như stack
	cout << endl << "Mang ban dau:" << endl;
	for (i=0; i<V.size(); i++) // Ghi lại nội dung của mảng ra màn h.nh
		cout << V[i] << endl;
	V.pop_back( ); // Xóa phần tử vừa chèn vào đi
	cout << endl << "Xoa phan tu cuoi:" << endl;
	for (i=0; i<V.size(); i++) // In nội dung của vector sau khi xóa
		cout << V[i] << endl;
	return 0;
}
```

Với ví dụ trên, bạn có thể thấy ta có thể sử dụng vector như 1 stack:
- Không nên dùng toán tử [] để truy xuất các phần tử mà nó không tồn tại, nghĩa là ví dụ vector size = 10, mà bạn truy xuất 11 là sai. Để thêm vào 1 giá trị cho vector mà nó không có size trước hoặc đã full thì ta dùng hàm thành viên push_back(), hàm này sẽ thêm 1 phần tử vào cuối vector.
- Tương tự với thao tác xóa một phần tử ở cuối ra khỏi vector, bạn cũng chỉ cần sử dụng 1 lệnh: pop_back( )

Xóa tại vị trí bất kỳ, xóa trắng

```
#include <iostream>
#include <vector>
using namespace std;
template <class T>
void print(const vector<T>&v)
{
   for (int i=0; i < v.size(); i++)
      cout << v[i] << endl;
}
int main()
{
   char *chao[] = {"Xin", "chao", "tat", "ca", "cac", "ban"};
   int n = sizeof(chao)/sizeof(*chao);
   vector<char*> v(chao, chao + n);
   //đây là 1 cách khởi tạo vector


   cout << "vector truoc khi xoa" << endl;
   print(v);

   v.erase(v.begin()+ 2, v.begin()+ 5);
   //xóa từ phần tử thứ 2 đến phần tử thứ 5
   v.erase( v.begin()+1 );
   //xóa phần tử thứ 1
   cout << "vector sau khi xoa" << endl;
   print(v);

   v.clear();//Xóa toàn bộ các phần tử
   cout << "Vector sau khi clear co "
      << v.size() << " phan tu" << endl;
   return 0;
}
```

Output:

```
vector truoc khi xoa
Xin
chao
tat
ca
cac
ban
vector sau khi xoa
xin
ban
Vector sau khi clear co 0 phan tu
```

Phương thức chèn

```
iterator insert ( iterator position, const T& x );
void insert ( iterator position, size_type n, const T& x );
void insert ( iterator position, InputIterator first, InputIterator last );
```

Ví dụ:

```
// inserting into a vector
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
        vector<int> v1(4,100);
        v1.insert ( v1.begin()+3 , 200 );
        //chèn 200 vào trước vị trí thứ 3
        v1.insert ( v1.begin()+2 ,2,300);
```

```
        //chèn 2 lần 300 vào trước vị trí thứ 2
        vector<int> v2(2,400);
        int a [] = { 501, 502, 503 };
        v1.insert (v1.begin()+2, a, a+3);
        //chèn mảng a (3 phần tử) vào trước vị trí thứ 2
        v1.insert (v1.begin()+4,v2.begin(),v2.end());
        //chèn v2 vào trước vị trí thứ 4
        cout << "v1 contains:";
        for (int i=0; i < v1.size(); i++)
                cout << " " << v1[i];
        return 0;
}
```

Output:

```
v1 contains: 100 100 501 502 400 400 503 100 200 300 300 100
```

Một số hàm khác và chức năng

Những toán tử so sánh được định nghĩa cho vector: ==, <, <=, !=, >, >=

Tham chiếu back(), front()

```
template<class _TYPE, class _A>
reference vector::front( );
template<class _TYPE, class _A>
reference vector::back( );
```

Trả về tham chiếu đến phần tử đầu và cuối vector: v.front() ⇔ v[0] và v.back() ⇔ v[v.size()-1]

```
#include <iostream>
#include <vector>
using namespace std;

int main ()
{
   int a[] = {3,2,3,1,2,3,5,7};
   int n = sizeof(a)/sizeof(*a);
   vector<int> v(a, a+n);

   cout << "phan tu dau la " << v.front() << endl;
   cout << "phan tu cuoi la " << v.back() << endl;
   cout << "gan phan tu cuoi la 9 ..." << endl;
   v.back() = 9;
   cout << "gan phan tu dau la 100 ..." << endl;
   v.front() = 100;

   cout << "kiem tra lai vector: ";
   for (int i=0; i < v.size(); i++)
      cout << v[i] << " ";
   cout << endl;
   return 0;
}
```

Output:

```
phan tu dau la 3
phan tu cuoi la 7
gan phan tu cuoi la 9 ...
gan phan tu dau la 100 ...
kiem tra lai vector: 100 2 3 1 2 3 5 9
```

```
Press any key to continue …
```

Hàm thành viên empty()

Để xác định vector có rỗng hay không ta dùng hàm thành viên empty(), hàm này trả về true nếu vector rỗng, và false ngược lại. Cú pháp :

```
if(v.empty() == true) {
      cout << "No values in vector \n";
}
```

- **capacity()** : Trả về số lượng phần tử tối đa mà vector được cấp phát, đây là 1 con số có thể thay đổi do việc cấp phát bộ nhớ tự động hay bằng các hàm như reserve() và resize()

Sự khác biệt giữa 2 hàm **size()** và **capacity()** :

```
#include<vector>
#include<iostream>
int main(int argc , char **argc)
{
    vector<int >so1,so2[10];
    so1.reserve(10);
    cout <<"Kich thuoc toi da:"<<so1.capacity();
    cout <<"\n Kich thuoc hien tai cua mang 2 "<<so2.size()<<endl;
    return 0 ;
}
```

- **reserve()**: cấp phát vùng nhớ cho vector, giống như **realloc()** của C và không giống **vector::resize(),** tác dụng của **reserve** để hạn chế vector tự cấp phát vùng nhớ không cần thiết.Ví dụ khi bạn thêm 1 phần tử mà vượt quá capacity thì vector sẽ cấp phát thêm, việc này lặp đi lặp lại sẽ làm giảm performance trong khi có những trường hợp ta có thể ước lượng được cần sử dụng bao nhiêu bộ nhớ.
Ví dụ nếu ko có **reserve()** thì capacity sẽ là 4 :

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

using namespace std;

int main()
{
   vector< int > my_vect;
   my_vect.reserve( 8 );
   my_vect.push_back( 1 );
   my_vect.push_back( 2 );
   my_vect.push_back( 3 );

   cout << my_vect.capacity() << "\n";
    return 0;
}
```

- **swap();** hoán đổi 2 container với nhau (giống việc hoán đổi giá trị của 2 biến kiểu số). Ví dụ : v1.swap(v2);

V.Kiểm tra tràn chỉ số mảng

Có một vấn đề chưa được đề cập đến từ khi ta làm quen với vector, đó là khả năng kiểm tra tràn chỉ số mảng (range check), để biết về khả năng này, chúng ta lại tiếp tục với một ví dụ mới:

```
#include <iostream>
#include <vector>
#include <conio.h>
```

```
using namespace std;
int main()
{
	try { // sử dụng try...catch để bẫy lỗi
		vector<long> V(3, 10); // Khởi tạo vector gồm 3 thành phần
		// Tất cả gán giá trị 10
		cout << "V[0]=" << V[0] << endl; // Đưa thành phần 0 ra màn hình
		cout << "V[1]=" << V[1] << endl; // Đưa thành phần 1 ra màn hình
		cout << "V[2]=" << V[2] << endl; // Đưa thành phần 2 ra màn hình
		cout << "V[3]=" << V[3] << endl; // Thành phần 3 (lệnh này hoạt động không
		// đúng vì V chỉ có 3 thành phần 0,1,2
		cout << "V[4]=" << V[4] << endl; // Thành phần 4 (càng không đúng)
		// Nhưng 2 lệnh trên đều không gây lỗi
		cout << "V[0]=" << V.at(0) << endl; // Không sử dụng [], dùng phương thức at
		cout << "V[1]=" << V.at(1) << endl; // Thành phần 1, OK
		cout << "V[2]=" << V.at(2) << endl; // Thành phần 2, OK
		cout << "V[3]=" << V.at(3) << endl; // Thành phần 3: Lỗi, chương trình dừng
		cout << "V[4]=" << V.at(4) << endl;
		getchar();
	} catch (exception &e) {
		cout << "Tran chi so ! " << endl;
	}
	return  0;
}
```

Trong ví dụ này, chúng ta lại có thêm một số kinh nghiệm sau:
- Nếu sử dụng cú pháp biến_vector[chỉ_số], chương trình sẽ không tạo ra lỗi khi sử dụng chỉ số mảng nằm ngoài vùng hợp lệ (giống như mảng thường). Trong ví dụ, chúng ta mới chỉ lấy giá trị phần tử với chỉ số không hợp lệ, trường hợp này chỉ cho kết quả sai. Nhưng nếu chúng ta gán giá trị cho phần tử không hợp lệ này, hậu quả sẽ nghiêm trọng hơn nhiều vì thao tác đó sẽ làm hỏng các giá trị khác trên bộ nhớ.
- Phương thức at(chỉ_số) có tác dụng tương tự như dùng ký hiệu [], nhưng có một sự khác biệt là thao tác này có kiểm tra chỉ số hợp lệ. Minh chứng cho nhận xét này trong ví dụ khi chương trình chạy đến vị trí lệnh V.at(3), lệnh này không cho ra kết quả mà tạo thành thông báo lỗi.

VI.Mảng 2 chiều với Vector

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

using namespace std;

int main()
{
	vector< vector<int> > matrix(3, vector<int>(2,0));
	//chu y viet > > de khong nham voi toan tu >>
	for(int x = 0; x < 3; x++)
	 for(int y = 0; y < 2; y++)
		matrix[x][y] = x*y;

	for(int x = 0; x < 3; x++)
	for(int y = 0; y < 2; y++)
		cout << matrix[x][y];
	cout << '\n';

	return 0;
}
```

Ví dụ này minh họa việc sử dụng mảng 2 chiều, thực chất đây là một vector của vector. Mảng 2 chiều sử dụng biện pháp này có thể có kích thước khác nhau giữa các d.ng (ví dụ mảng 2 chiều là nửa trên của ma trận)

VII.Sử dụng iterator

Container (thùng chứa) : một kiểu cấu trúc dữ liệu dùng để lưu trữ thông tin. Ví dụ: mảng (array), list, vector, deque...

Container nào cũng có các phương thức sau đây

Phương thức	Mô tả
size()	Số lượng phần tử
empty ()	Trả về 1 nếu container rỗng, 0 nếu ngược lại.
max_size()	Trả về số lượng phần tử tối đa đã được cấp phát
==	Trả về 1 nếu hai container giống nhau
!=	Trả về 1 nếu hai v khác nhau
begin()	Trả về iterator đầu tiên của container
end()	Trả về iterator lặp cuối cùng của container
front()	Trả về tham chiếu đến phần tử đầu tiên của container
back()	Trả về tham chiếu đến phần tử cuối cùng của container
swap()	Hoán đổi 2 container với nhau (giống việc hoán đổi giá trị của 2 biến)

Trong thư viện STL thì người ta tích hợp lớp đối tượng Iterator (bộ lặp hay biến lặp) cùng với các container.Tư tưởng đó thể hiện như sau:

- Các đối tượng Iterator là các con trỏ đến các đối tượng của lớp lưu trữ:
  typedef__gnu_cxx::__normal_iterator <pointer,vector_type> iterator;
- Khai báo lớp Iterator như là 1 lớp nằm trong lớp lưu trữ.
- Xác định trong lớp lưu trữ các phương thức thành phần như:
  - begin() – trả lại con trỏ kiểu đối tượng Iterartor đến phần tử đầu tiên của nằm trong đối tượng lớp lưu trữ.
  - end() – trả lại con trỏ kiểu Iterator trỏ đến 1 đối tượng nào đó bên ngoài tập các phần tử được lưu trữ. Đối tượng bên ngoài nào đó có thể có các định nghĩa khác nhau.Trong trường hợp cụ thể như vector ta có thể hiểu là trỏ đến phần tử sau phần tử cuối cùng.
- Xác định trong lớp đối tượng kiểu Iterator các toán tử như sau:
  - ++p hoặc p++ : chuyển iterator p đến phần tử kế tiếp.
  - --p hoặc p-- : chuyển iterator p đến phần tử đằng trước nó.
  - *p : xác định giá trị của phần tử mà iterator p trỏ đến.

Như bạn biết, mảng và con trỏ có mối quan hệ chặt chẽ với nhau trong C++. Một mảng có thể được truy xuất thông qua con trỏ. Sự tương đương này trong STL là mối quan hệ giữa iterator và vector, mà tổng quát hơn là với container. Nó cung cấp cho chúng ta khả năng xử lý theo chu kì thông qua nội dung của container theo một cách giống như là bạn sử dụng con trỏ để tạo xử lý chu kỳ trong mảng.
Bạn có thể truy xuất đến các thành phần của một container bằng sử dụng một iterator:

```
<container>  coll;
for (<container>::iterator it = coll.begin(); it != coll.end(); ++it)
{
   *it;
   //……..
}
```

Dưới đây chúng ta xét 1 ví dụ làm việc với thư viện STL với lớp vector và con trỏ kiểu iterator như sau:

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

using std::vector;

void main()
{
   vector<int> v;

   for(int i = 10; i < 15; i++) v.push_back(i);

   vector<int>::iterator it = v.begin();
   while(it !=  v.end())
   {
      cout << *it << "   ";
      it++;
```

```
    }
}
```

Vì iterator định nghĩa bên trong các container – thế nào là "phần tử đầu", "phần tử cuối", "phần tử tiếp theo" … nên nó "chứa" thông tin cấu trúc phục vụ cho việc duyệt container. Nó che đi cấu trúc bên trong và cho phép ta viết các đoạn mã tổng quát để duyệt hay chọn phần tử trên các container khác nhau mà không cần biết cấu trúc của container đó ra sao.

Trong các reversible container như vector còn định nghĩa thêm reverse_iterator (cái tên nói lên chức năng: iterator nghịch đảo) là nested class với các "hằng" tương ứng:

```
vector<T>::reverse_iterator rbegin();
vector<T>::reverse_iterator rend();
```

Ví dụ : duyệt vector theo 2 chiều

```
#include <iostream>
#include <algorithm>
#include <stdlib.h>
#include <vector>
using namespace std;
int main()
{
	int A[] = {3,2,3,1,2,3,5,3};
	int n = sizeof(A)/sizeof(*A);
	vector<int> V;
	for (int i=0; i<n; i++){
		V.push_back(A[i]);

	vector<int>::iterator vi;
	cout << endl << "Duyet chieu xuoi" << endl;
	for (vi=V.begin(); vi!=V.end(); vi++)
		cout << *vi << endl;

	vector<int>::reverse_iterator rvi;
	cout << endl << "Duyet theo chieu nguoc" << endl;
	for (rvi=V.rbegin(); rvi!=V.rend(); rvi++)
		cout << *rvi << endl;

	getchar();
}
```

Chuyển đổi qua lại giữa reverse_iterator và iterator:

- Hàm thành viên base(): trả về một iterator trỏ đến phần tử hiện tại của reverse_iterator.
- Tạo reverse_iterator từ iterator: Contructor reverse_iterator(RandomAccessIterator i);

Ví dụ:

```
vector<int> v;
vector<int>::iterator it(v.begin());
vector<int>:: reverse_iterator ri(v.rbegin());
//goi contructor
assert(ri.base()==v.end()-1);
ri=v.begin();
//goi contructor
assert(ri.base()==it);
```

**Lệnh assert(); dùng để kiểm tra một biểu thức điều kiện.*

Iterator là 1 trong 4 thành phần chính của STL (container, iterator, algorithm và functor).Container và algorithm giao tiếp qua nó: nhiều hàm và algorithm trong STL nhận các đối số là iterator.Iterator gắn liền với tất cả các loại container, đây là khái niệm bạn cần nắm rất vững nếu muốn làm việc tốt với STL

Bảng: các hàm thành viên lớp vector

Hàm thành phần	Mô tả
template<class InIter> void assign(InIter start, InIter end);	Gán giá trị cho vector theo trình tự từ start đến end.
Template<class Size, class T) Void assign(Size num, const T &val = T());	Gán giá trị của val cho num phần tử của vector.
Reference at(size_type I); Const_reference at(size_type I) const;	Trả về một tham chiếu đến một phần tử được chỉ định bởi i.
Reference back(size_type I); Const_reference at(size_type I) const;	Trả về một tham chiếu đến phần tử cuôi cùng của vector.
Iterator begin(); Const _iterator begin() const;	Trả về một biến lặp chỉ định phần tử đầu tiên của vector.
Size_type capacity() const;	Trả về dung lượng hiện thời của vector. Đây là số lượng các phần tử mà nó có thể chứa trước khi nó cần cấp phát thêm vùng nhớ.
Void clear();	Xóa tất cả các phần tử trong vector.
Bool empty() const;	Trả về true nếu vector rỗng và trả về false nếu ngược lại.
Iterator end(); Const_iterator end() const	Trả về một biến lặp để kết thúc một vector.
iterator erase(iterator i);	Xóa một phần tử được chỉ bởi i. Trả về một biến lặp chỉ đến phần tử sau phần tử được xóa.
Iterator erase(iterator start, iterator end);	Xóa những phần tử trong dãy từ start đến end. Trả về một biến lặp chỉ đến phần tử sau cùng của vector.
Reference front(); Const_reference front() const;	Trả về một tham chiếu đến phần tử đầu tiên của vector.
Allocator_type get_allocator() const;	Trả về vùng nhớ được cấp phát cho vector.
Iterator insert(iterator I, const T&val=T());	Chèn val trực tiếp vào trước thành phần được chỉ định bởi i. biến lặp chỉ đến phần tử được trả về.
Void insert(iterator I, size_type num, const T& val);	Chèn num val một cách trực tiếp trước phần tử được chỉ định bởi i.
Template<class InIter> Void insert(iterator I, InIter start, InItr end);	Chèn chuỗi xác định từ start đến end trực tiếp trước một phần tử được chỉ định bởi i.
Size_type max_size() const;	Trả về số lượng phần tử lớn nhất mà vector có thể chứa.
Reference operator[](size_type i) const; Const_reference operator[](size_type i) const;	Trả về một tham chiếu đến phần tử được chỉ định bởi i.
Void pop_back();	Xóa phần tử cuối cùng trong vector.
Void push_back(cons T&val);	Thêm vào một phần tử có giá trị val vào cuối của vector.
Reverse_iterator rbegin(); Const_reverse_iterator rbegin() const;	Trả về biến lặp nghịch chỉ điểm kết thúc của vector.
Reverse_iterator rend(); Const_reverse_iterator rend() const;	Trả về một biến lặp nghịch chỉ điểm bắt đầu của vector.
Void reverse (size_type num);	Thiết lập kích thước của vector nhiều nhất là bằng num.
Void resize (size_type num, T val =T());	Chuyển đổi kích thước của vector được xác định bởi num. Nếu như kích thước của vector tăng lên thì các phần tử có giá trị val sẽ được thêm vào cuối vector.
Size_type size() const;	Trả về số lượng các phần tử hiện thời của

	trong vector.
Vois swap(vector<T, Allocator>&ob)	Chuyển đổi những phần tử được lưu trong vector hiện thời với những phần tử trong ob.

VIII.Dùng 1 số hàm cơ bản trong thư viện algorithm

Khởi tạo:

Bạn có thể sử dụng **lệnh fill** của thư viện <algorithm> để tô một vùng giá trị của 1 container (thường là 1 mảng, 1 vector)

```
// fill algorithm example
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

void printint(const int &i)
{
   cout << i << endl;
}

int main ()
{
   vector<int> v(8);                 // v: 0 0 0 0 0 0 0 0
   fill(v.begin(),v.begin()+4,5);    // v: 5 5 5 5 0 0 0 0
   fill(v.begin()+3,v.end()-2,8);    // v: 5 5 5 8 8 8 0 0

   cout << "vector contains:";
   for_each( v.begin(), v.end(), printint );

   return 0;
}
```

```
template <class ForwardIterator, class Generator>
void generate(ForwardIterator first, ForwardIterator last, Generator gen);
```

Hàm generate sẽ "sinh" từng phần tử trong khoảng nào đấy của vector bằng cách gọi hàm được chỉ định (một hàm trả về cùng kiểu và không có đối số)

Ví dụ với hàm rand():

```
vector<int> V;
srand( time(NULL) );
//...
generate( V.begin(), V.end(), rand );
```

Copy iterator (tương tự memcpy() đối với pointer)

```
int a[] = {1, 2, 3, 4, 5, 6};
int n = sizeof(a)/sizeof(*a);
vector<int> v1(a, a+n);

vector<int> v2(n);
copy(v1.begin(), v1.end(), v2.begin());
//copy_n(v1.begin(), v1.size(), v2.begin());

for (int i=0; i<n; i++)
        cout << v2[i] << " ";
// The output is "1 2 3 4 5 6".
```

Đảo ngược (reverse) vector

Bạn có thể sử dụng lệnh reverse trong <algorithm> để đảo ngược container (ở đây là 1 vector)

```
// reverse algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
#include <conio.h>
using namespace std;
int main ()
{
	vector<int> a;
	// set some values:
	for (int i=1; i<10; ++i) a.push_back(i);	// 1 2 3 4 5 6 7 8 9
	reverse(a.begin(),a.end());			// 9 8 7 6 5 4 3 2 1
	// print out content:
	cout << "a contains:";
	int i, n = a.size();
	for (i=0; i<n; i++)
		cout << a[i] << " ";
	cout << endl;
	getchar();
}
```

Thay thế các giá trị (replace)

Lệnh **replace** tìm kiếm 1 giá trị trong container, ở đây là vector thay thế giá trị tìm được bởi giá trị mới.

```
// replace algorithm example
#include <iostream>
#include <algorithm>
#include <vector>
#include <conio.h>
using namespace std;
int main ()
{
	int myints[] = { 10, 20, 30, 30, 20, 10, 10, 20 };
	vector<int> a (myints, myints+8);		// 10 20 30 30 20 10 10 20
	replace(a.begin(), a.end(), 20, 99);		// 10 99 30 30 99 10 10 99
	cout << "a contains: ";
	int i, n;
	for (i=0, n=a.size(); i<n; i++)
	cout << a[i] << " ";
	getchar();
}
```

Thay thế các giá trị theo điều kiện (replace_if)

Các hàm có hậu tố _if như remove_if, replace_if, find_if, count_if … sử dụng 1 con trỏ hàm hoặc 1 functor để làm tiêu chí tìm kiếm (dạng của con trỏ hàm hoặc functor tùy vào mục đích sử dụng)

Lệnh **replace_if** cho phép tìm giá trị theo điều kiện do một hàm trả về. Để sử dụng lệnh này bạn phải khai báo 1 hàm có giá trị trả về là bool nhận tham số là giá trị của 1 element. Khi hàm trả về true, giá trị tương ứng sẽ bị thay thế bởi giá trị mới. Hàm kiểm tra nên khai báo inline để tốc độ nhanh hơn.

```
// replace_if example
#include <iostream>
#include <algorithm>
#include <vector>
#include <conio.h>
using namespace std;
```

```
inline bool SoLe(int i) { return ((i%2)==1); }
int main ()
{
	vector<int> a;
	// set some values:
	for (int i=1; i<10; i++) a.push_back(i);	// 1 2 3 4 5 6 7 8 9
	replace_if(a.begin(), a.end(), SoLe, 0);	// 0 2 0 4 0 6 0 8 0
	cout << "a contains: ";
	int i, n;
	for (i=0, n=a.size(); i<n; i++)
		cout << a[i] << " ";
	getchar();
}
```

Xóa với remove và remove_if

Ví dụ 1:

```
//remove
int A[] = {3,1,4,1,5,9};
int N = sizeof(A)/sizeof(*A);
vector<int> V(A, A+N);

copy(V.begin(), V.end(), ostream_iterator<int>(cout, " "));
//The output is "3 1 4 1 5 9".
cout << endl;

vector<int>::iterator new_end =
   remove(V.begin(), V.end(), 1);

V.resize(new_end - V.begin());

copy(V.begin(), V.end(), ostream_iterator<int>(cout, " "));
// The output is "3 4 5 9".
```

Ví dụ 2:

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

bool IsOdd(int x)
{
   return x%2;
}

int main()
{
   int a[] = {3,1,4, 8, 5, 2, 9};
   int n = sizeof(a)/sizeof(*a);
   vector<int> vec(a, a+n);
   vector<int>::iterator new_end =
        remove_if( vec.begin(), vec.end(), IsOdd );
   vec.erase(new_end, vec.end());
   copy(vec.begin(), vec.end(), ostream_iterator<int>(cout, " "));
   // The output is "4 8 2".
   return 0;
}
```

Các hàm có hậu tố _copy như remove_copy, remove_if_copy, replace_copy, replace_if_copy, reverse_copy sử dụng tương tự nhưng tạo ra và thao tác trên bản sao container

Sắp xếp container (ở đây là vector)

```
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

void printint(const int &i)
 {
        cout << i << endl;
}

int main()
 {
        int A[] = {3,2,3,1,2,3,5,3};
        int n = sizeof(A)/sizeof(*A);
        vector<int> V(A, A+n);

        cout << endl << "Danh sach ban dau" << endl;
        for_each( V.begin(), V.end(), printint );

        V.sort();
        vector<int>::iterator vi;
        cout << endl << "Danh sach sau khi sap xep" << endl;
        for_each( V.begin(), V.end(), printint );

        getchar();
}
```

Trong ví dụ trên ta học được cách sử dụng hàm for_each() và sort() để thao tác với 1 container.

Tìm kiếm tuyến tính

```
// lineal search
#include <iostream.h>
#include <algorithm>
#include <vector>
using namespace std;

bool IsOdd(int x)
{
   return x%2;
}

int main()
{
        int a[] = {2,4,2,6,9,1,2,3,2,3,4,5,6,4,3,2,1};
        int n = sizeof(a)/sizeof(*a);
        vector<int> v(a, a+n);

        int first = find(a, a+n, 1) - a;
        //các hàm thuật toán hoàn toàn có thể thao tác trên mảng & con trỏ thường
        cout << "[array] so thu tu cua phan tu dau tien = 1: " << first << endl;

        first = find(v.begin(),v.end(), 1) - v.begin();
        cout << "[vector] so thu tu cua phan tu dau tien = 1: " << first << endl;

        //find_if
          vector<int>::iterator it;
          it = find_if(v.begin(),v.end(), IsOdd );
          first = it - v.begin();
          cout << "Phan tu le dau tien la " << *it << " o vi tri thu " << first << endl;
```

```
	return 0;
}
```

Ngoài ra còn có nhiều hàm tìm kiếm khác: hàm search() dùng để so khớp 1 chuỗi liên tiếp các phần tử cho trước, hàm search_n tìm kiếm với số lần lặp xác định, hàm find_end tìm kết quả cuối cùng, find_first_not_of(), find_last_not_of() …

Đếm & tìm min max

```
//counting
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

bool IsOdd(int x)
{
   return x%2;
}

int main()
{
	int a[] = {3,2,3,1,2,4,5,3};
	int n = sizeof(a)/sizeof(*a);
	vector<int> v(a, a+n);

	cout << "So luong so 3 trong mang: " << count(v.begin(),v.end(), 3) << endl;
	cout << "So luong so le trong mang: " << count_if(v.begin(),v.end(), IsOdd) << endl;
	cout << "So nho nhat trong mang: " << *min_element(v.begin(),v.end()) << endl;
	cout << "So lon nhat trong mang: " << *max_element(v.begin(),v.end()) << endl;;

	return 0;
}
```

Hàm chuyển đổi hàng loạt transform()

Một trong những giải thuật thú vị hơn là transform(), phần tử được sửa đổi từng cái trong một phạm vi theo một chức năng mà bạn cung cấp.

```
	#include <vector>
	#include <algorithm>
	using namespace std;

	template <class T>
	T inc(T x)
	//hàm dùng để chuyển đổi
	{
		return x+1;
	}
	int main()
	{
		int a[] = {3,2,3,1,2,3,5,3};
		int n = sizeof(a)/sizeof(*a);
		vector<int> v(a, a+n);
		transform(v.begin(), v.end(), v.begin(), inc<int> );

		copy(v.begin(), v.end(), ostream_iterator<int>(cout, " "));
		// The output is "4 3 4 2 3 4 6 4".

		return 0;
	}
```

Ngoài ra còn nhiều hàm thuật toán khác có thể tham khảo trong tài liệu reference của c++