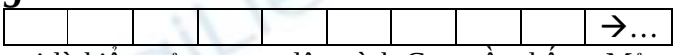
SỬ DỤNG STL VECTOR TRONG C++

Nguyễn Trí Hải
11520094
KHMT06

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?**
- 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
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 dùng vector thì cần thêm 1 header #include <vector> và phải có using std::vector; vì vector được định nghĩa trong STL (Standard Template Library)
Ví dụ:

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

Câu lệnh trên định nghĩa 1 vector có kiểu int. Kiểu vector được đặt trong 2 dấu ngoặc nhọn. Kích thước của vector có thể tự nâng lên nên không cần khai báo số phần tử hoặc nếu muốn khai báo thì bạn 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. Mặc dù size của nó bằng 10 nhưng sử dụng hơn vẫn được.
Ta có thể khởi tạo giá trị mặc định cho vector:

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

Câu lệnh trên khai báo 10 phần tử vector A 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 vè vector, ta xem đoạn code sau:

```
#include <iostream>       // Thư viện iostream phục vụ ghi dữ liệu ra màn hình
#include <vector>               // Thư viện vector, sử dụng kiểu vector
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 hình
            cout << V[i] << endl;
      // Nếu sử dụng mảng, bạn phải có biến lưu kích thước
}
```

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;
```

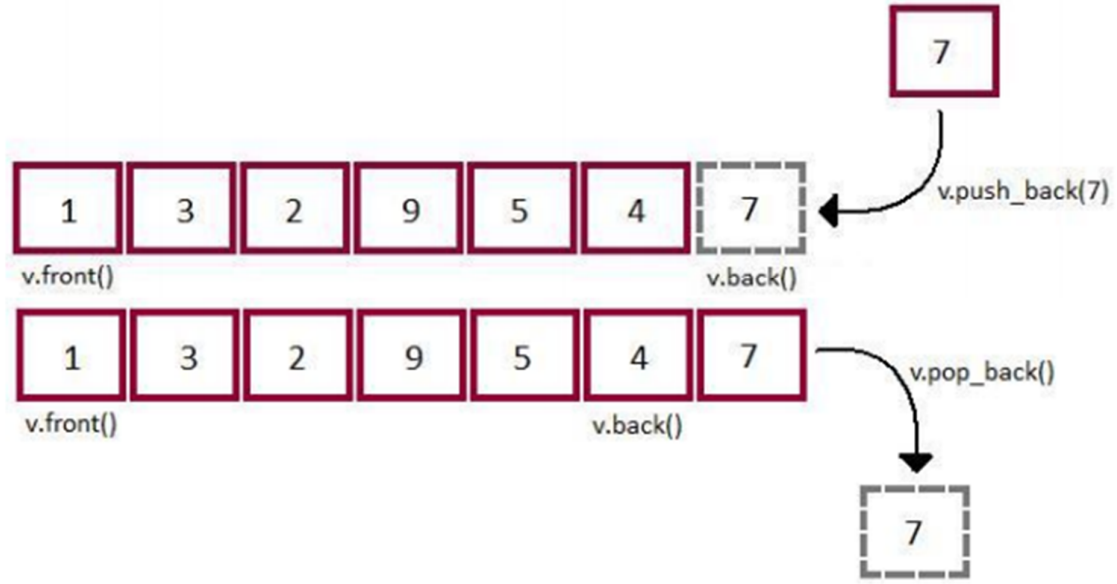
```
}
```

Ouput:

```
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 l ần 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 s ẽ 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;
```

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 lo ạt 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++