**BÁO CÁO**

Chủ Đề : TÌM HIỂU CÁC LỚP VECTOR, LIST, ARRAYLIST

Thành viên nhóm:

Mai Anh Tuấn

Nguyễn Tiến Anh

MỤC LỤC

1. LỚP VECTOR 0
2. Khai báo lớp vector 1
3. Phương thức của lớp vector 3
4. LỚP LIST 18
5. Khai báo lớp list 18
6. Phương thức của lớp list 19
7. LỚP ARRAYLIST 53
8. SO SÁNH LỚP VECTOR, LIST VÀ ARRAYLIST 55
9. **LỚP VECTOR**

Vector có thể hiểu là một mảng có trình tự, giống như với danh sách liên kết hay một chuỗi thông thường nhưng “vector” khác với chuỗi hoăc mảng thông thường là chúng ta có thể thay đổi kích thước của nó. Và do nó giống với chuỗi thông thường nghĩa là nó cũng có thể truy cập trực tiếp đến các phần tử.

1. **Khai báo lớp vector:**

Để sử dụng vector, đầu tiên ta cần khai báo thư viện vector:

#include<vector>

**Vector 1 chiều:**

Từ ví dự đến  ví dụ 4 là cách khai báo thường dùng cho vector 1 chiều.

Ví dụ 1: Khai báo vector 1 chiều, rỗng, dữ liệu kiểu int có tên là firstVector:

vector <int> firstVector;

Ví dụ 2: Tạo vector có 4 phần tử int và 4 phần tử này đều có giá trị 100 có tên là secondVector:

vector <int> secondVector (4,100);

Ví dụ 3: Khai báo một vector kiểu int có tên là thirdVector sao chép từ đầu đến cuối secondVector:

vector <int> thirdVector (secondVector.begin(),secondVector.end()) ;

Lưu ý: begin(), end() là các phương thức cung cấp sẵn của đối tượng vector dùng để truy xuất phần tử đấu tiên và cuối cùng của vector.

Ví dụ 4: Tạo vector kiểu int tên là fourVector và sao chép tất cả phần tử  thirdVector

vector <int> fourList (thirdList) ;

Khai báo ở Ví dụ 4 có chức năng tương tự như ví dụ 3 nhưng đơn giản hơn.

**Vector 2 chiều:**

Ví dụ 5:  Tạo vector 2 chiều, phần tử rỗng, kiểu int có tên là v:

vector < vector <int> > v;

Ví dụ 6: khai báo vector kích thước 5×10 :

vector < vector <int> > v (5, 10) ;

Ví dụ 7: Khai báo vector 2 chiều, có 5 vector 1 chiều đều rỗng:

vector < vector <int> > v (5) ;

Ví dụ 8: Khai báo vector 5×10 với các phần tử khởi tạo giá trị là đều là 1

vector < vector <int> > v (5, vector <int> (10,1) ) ;

Khai báo vector 2 chiều SAI

vector <vector <int>> v ;

1. **Các phương thức của lớp vector**

**size()** : trả về số lượng phần tử.

Ví dụ: Tạo một vector có kích thước là 10 và in ra số lượng phần tử của vector này.

#include<iostream>

#include<vector>

Usingname space std;

Int main()

{

Vector<int> myVector(10);

Cout<<”The size of the vector is: “<<myVector size()<<endl;

Return 0;

}

Empty(): trả về true(1) nếu vector rỗng, ngược lại false(0).Ví dụ: kiểm tra vector có rỗng không, và in ra thông báo tương ứng.

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> myVector ;

if( myVector.empty()) {

cout<<"Vector is empty! "<<endl ;

}

else {

cout<<"Vector is not empty! "<<endl ;

}

return 0;

}

**operator [ ]** : Sử dụng [ ] để truy cập phần tử tương tự như đối với mảng thông thường.

Ví dụ: Tạo một vector 10 phần tử với giá các phần tử  lần lượt là  từ 0 đến 9. In ra các phần tử của vector.

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> myVector ;

for(int i = 0 ; i<10 ; i++) {

myVector.push\_back(i) ;

}

for(int i = 0 ; i<10 ; i++) {

cout<<myVector[i]<<" " ;

}

return 0;

}

Lưu ý: Để sử dụng “myVector[i]” thay vì sử dụng” myVector.push back(i)” thì phải khai báo vector “vector<int> myVector(10)” thay cho “vector<int> myVector”

**At (int i)**: truy cập phần tử thứ i trong vector.

Phương thức at() có chức năng tương tự như operator [].

Ví dụ: Tạo vector có 10 phần tử bao gồm 0, 10, 20, 30 , 40, 50, 60, 70, 80, 90. Sau đó, sử dụng phương thức at() để in giá trị các phần tử.

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> myVector;

// create a vector

for(int i = 0 ; i<10; i++) {

myVector.push\_back(i\*10) ; // 0 10 20 30 40 50 60 70 80 90

}

// print vector

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

cout<<myVector.at(i)<<" " ;

}

return 0;

}

### font(): Truy xuất phần tử đầu tiên của vector. Truy xuất ở đây bao gồm in hoặc chỉnh sửa giá trị.

### Ví dụ: Tạo vector có 5 phần tử có giá trị lần lượt là a,b,c,d,e. Thay đổi phẩn tử đầu tiền thành ‘A’ và in ra kết quả.

### #include <iostream>

### #include <vector>

### using namespace std;

### int main()

### {

### vector<char> myVector;

### // create a vector

### for(int i = 0 ; i<5; i++) {

### myVector.push\_back('a'+ i ) ; // a b c d e

### }

### // change vector

### cout<<"Before change, front is: "<<myVector.front()<<endl ;

### myVector.front() = 'A' ;

### cout<<"After change, front is: "<<myVector.front()<<endl ;

### return 0;

### }

### Back():Truy xuất phần tử cuối cùng của vector. Truy xuất ở đây bao gồm in hoặc chỉnh sửa giá trị.

### Ví dụ: Tạo vector có 5 phần tử có giá trị lần lượt là  A,B,C,D,E. Thay đổi phẩn tử cuối cùng thành ‘e’ và in ra kết quả.

### #include <iostream>

### #include <vector>

### using namespace std;

### int main()

### {

### vector<char> myVector;

### // create a vector

### for(int i = 0 ; i<5; i++) {

### myVector.push\_back('A'+ i ) ; // A B C D E

### }

### // change vector

### cout<<"Before change, back is: "<<myVector.back()<<endl ;

### myVector.back() = 'e' ;

### cout<<"After change, back is: "<<myVector.back()<<endl ;

### return 0;

### }

### Push\_pack(const x): Thêm phần tử có giá trị x vào cuối vector.

### Phương thức push\_back(x) thường dùng để khởi tạo vector một cách “động”, có nghĩa là kích thước vector chưa xác định trước.

### Ví dụ: Viết chương trình cho người dùng nhập vào kích thước vector và nhập vào giá trị các phần tử. In ra các phần tử trong vector.

### #include <iostream>

### #include <vector>

### using namespace std;

### int main()

### {

### vector<int> myVector;

### int n; // size of vector

### cout<<"Size: " ;

### cin>>n ;

### // create vector

### int tempNumber ;

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

### cin>>tempNumber ;

### myVector.push\_back(tempNumber ) ;

### }

### // print vector

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

### cout<<myVector[i]<<" " ;

### }

### 

### return 0;

### }

### Pop\_pack():Loại bỏ phần tử cuối ra khỏi vector.

### Ví dụ: Viết chương trình cho người dùng nhập vào kích thước vector và nhập vào giá trị các phần tử. Sau đó, loại bỏ đi một nửa các phần tử ở cuối vector.  In ra các phần tử trong vector.

### #include <iostream>

### #include <vector>

### using namespace std;

### int main()

### {

### vector<int> myVector;

### int n; // size of vector

### cout<<"Size: " ;

### cin>>n ;

### // create vector

### int tempNumber ;

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

### cin>>tempNumber ;

### myVector.push\_back(tempNumber ) ;

### }

### // delete element

### for(int i = 0 ; i<n/2; i++) {

### myVector.pop\_back() ;

### }

### // print vector

### for(int i = 0 ; i<myVector.size(); i++) {

### cout<<myVector[i]<<" " ;

### }

### return 0;

### }

Lưu ý: Việc dùng pop\_back() chỉ dịch chuyển con trỏ end() về phía trước. Điều này có nghĩa là giá trị của các phần tử cuối vector sau khi dùng lệnh pop\_back() vẫn còn lưu trữ. Cụ thể, trong chương trình trên, ở vòng lặp for cuối cùng, nếu ta thay đổi:

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

thành:

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

thì ta thấy giá trị các phần tử ta nhập ban đâu vẫn còn lưu lại.

### insert(): chèn

Insert có thể dùng với 3 cách sau:

insert (iterator positon,const x): Chèn phần tử có giá trị x vào **trước** vị trí position.

insert (iterator positon,int n, const x): Chèn n phần tử có giá trị x vào**trước** vị trí position.

insert (iterator positon,iterator a, itertator b): Chèn vào **trước** vị trí position của vector hiện hành tất cả các phần tử trong nửa khoảng [a,b) của một vector khác.

Ví dụ 1: Tạo vector có 5 phần tử đều có giá trị là 100. Chèn phần tử có giá trị 200 vào vị trí thứ 2 của vector.

// inserting into a vector

#include <iostream>

#include <vector>

using namespace std ;

int main ()

{

// create vector

vector<int> myVector (5,100); // 100 100 100 100 100

// create iterator

vector<int>::iterator it;

it = myVector.begin();

// insert 200 into 2rd position of vector

myVector.insert(it+1,200) ; // 100 200 100 100 100 100

// print vector

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

cout<<myVector[i]<<" " ;

}

return 0;

}

Ví dụ 2: Tạo vector có 5 phần tử đều có giá trị là 100. Chèn 2 phần tử có giá trị 200 vào vị trí đầu tiên của vector.

// inserting into a vector

#include <iostream>

#include <vector>

using namespace std ;

int main ()

{

// create vector

vector<int> myVector (5,100); // 100 100 100 100 100

// insert tow 200 numbers into 1st position of vector

myVector.insert(myVector.begin(),2,200) ; // 100 200 200 100 100 100 100

// print vector

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

cout<<myVector[i]<<" " ;

}

return 0;

}

Ví dụ 3: Tạo vector thứ nhất có 5 phần tử đều có giá trị là 100. Tạo vector thứ hai có 3 phần tử đều có giá trị là 200. Chèn tất cả các phần tử của vector thứ hai vào vị trí thứ 2 của vector thứ nhất.

// inserting into a vector

#include <iostream>

#include <vector>

using namespace std ;

int main ()

{

// create vector

vector<int> myVector (5,100); // 100 100 100 100 100

vector<int> otherVector(3,200) ; // 200 200 200

// insert 2rd vector into 1st vector

myVector.insert(myVector.begin()+1, otherVector.begin(),otherVector.end()) ; // 100 200 200 100 100 100 100

// print vector

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

cout<<myVector[i]<<" " ;

}

return 0;

}

Lưu ý: Ngoài việc chèn vector này vào vector khác, ta có thể chèn một mảng các phần tử vào vector.

Ví dụ 4: tạo một vector có 5 phần tử đều có giá trị là 100. Tạo một mảng int có 3 phần tử lần lượt là 200, 300, 400. Chèn 2 phần tử đầu của mảng vào đầu vector.

// inserting into a vector

#include <iostream>

#include <vector>

using namespace std ;

int main ()

{

// create vector

vector<int> myVector (5,100); // 100 100 100 100 100

//create array

int myArray[3] = {200,300,400} ; // 200 300 400

// insert two element from array to vector

myVector.insert(myVector.begin(), myArray, myArray +2 ) ; // 100 200 200 100 100 100 100

// print vector

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

cout<<myVector[i]<<" " ;

}

return 0;

}

### Erase(): xóa 1 hoặc 1 dãy các phần tử trong vector.

erase (iterator position): Xóa phần tử ở vị trị position.

erase (iterator first, iterator last): Xóa tất cả các phần tử trong nửa khoảng [frist,last), có nghĩa là từ phẩn tử thứ first đến phần tử thứ (last-1).

Ví dụ 1: Tạo một vector có 10 phần tử có giá trị từ 1 đến 10. Xóa phần tử thứ 6 và in ra các phần tử còn lại.

#include <iostream>

#include <vector>

using namespace std ;

int main () {

vector<int> myvector;

// set some values (from 1 to 10)

for (int i=0; i<10; i++) {

myvector.push\_back(i+1); // 1 2 3 4 5 6 7 8 9 10

}

// erase the 6th element

myvector.erase (myvector.begin()+5);

// print vector

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

cout <<myvector[i]<<" " ;

}

return 0;

}

Ví dụ 2: Tạo một vector có 10 phần tử có giá trị từ 1 đến 10. Xóa  3 phần tử ở đầu vector.

#include <iostream>

#include <vector>

using namespace std ;

int main () {

std::vector<int> myvector;

// set some values (from 1 to 10)

for (int i=0; i<10; i++) {

myvector.push\_back(i+1); // 1 2 3 4 5 6 7 8 9 10

}

// erase the first 3 elements

myvector.erase (myvector.begin() , myvector.begin()+3);

// print vector

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

cout <<myvector[i]<<" " ;

}

return 0;

}

### swap(vector v): Hoán đổi vector v với vector hiện hành.

### Ví dụ: Cho hai vector, vector thứ nhất có 5 phần tử đều là số 1, vector thứ hai có 5 phần tử đều là số 2. Hoán đổi hai vector trên và in ra các phần tử của mỗi vector.

### #include <iostream>

### #include <vector>

### using namespace std;

### int main()

### {

### vector<int> firstVector (5, 1); // 1 1 1 1 1

### vector <int> secondVector (5,2) ; // 2 2 2 2 2

### // exchange 2 vector

### firstVector.swap(secondVector) ;

### // print vector 1

### for(int i = 0 ; i<firstVector.size(); i++) {

### cout<<firstVector[i]<<" " ;

### }

### cout<<endl ;

### // print vector 2

### for(int i = 0 ; i<secondVector.size(); i++) {

### cout<<secondVector[i]<<" " ;

### }

### return 0;

### }

### clear(): Xóa hoàn toàn một vector.

Ví dụ: Tạo một vector có 5 phần tử đều có giá trị là 5. In kích thước của vector trước và sau khi dùng phương thức clear().

#include <iostream>

#include <vector>

using namespace std;

int main()

{

vector<int> myVector (5,1); // 1 1 1 1 1

cout<<"Before clear vector, size of vector is : " << myVector.size()<< endl ;

myVector.clear() ; // resize vector

cout<<"After clear vector, size of vector is : " << myVector.size()<< endl ;

return 0;}

Lưu ý: phương thức clear() chỉ làm làm cho kích thước của vector về 0. Tuy nhiên, giá trị mà vector lưu trữ vẫn còn. Cụ thể, trong ví dụ trên, nếu ta thêm vào cuối chương trình câu lệnh:

for(int i = 0 ; i < 5 ; i++) {

cout<<myVector[i]<<" " ;

}thì ta thấy kết quả vẫn in ra được các giá trị các phần tử của vector.

1. **LỚP LIST**

### List là danh sách liên kết kép: Mỗi phần tử trong list có liên kết đến một phần tử trước nó và một phần tử phía sau nó.

### Do đó, list có các ưu điểm đó là chèn và loại bỏ phần tử ở bất cứ vị trí nào trong container với độ phức tạp O(1).

### Điểm yếu của list là khả năng truy cập tới phần tử thông qua vị trí với độ phức tạp O(n).

### Khai báo lớp list

### Để sử dụng list, ta cần khai báo thư viện :

#include <list>

Ví dụ 1: Khai báo list 1  có dữ liệu kiểu int có tên là firstList:

list <int> firstList;

Ví dụ 2: Tạo vector có 4 phần tử int và 4 phần tử này đều có giá trị 100 có tên là secondList:

list <int> secondList (4,100);

Ví dụ 3: Khai báo một list kiểu int có tên là thirdList sao chép từ đầu đến cuối secondList:

list <int> thirdList (secondList.begin(),secondList.end()) ;

Lưu ý: begin(), end() là các phương thức cung cấp sẵn của đối tượng list dùng để truy xuất phần tử đấu tiên và cuối cùng của list.

Ví dụ 4: Tạo list kiểu int tiên là fourList và sao chép tất cả phần tử  thirdList:

list <int> fourList (thirdList) ;

Nhận xét:

Khai báo ở ví dụ 4 có chức năng tương tự như ví dụ 3 nhưng đơn giản hơn.

1. **Các phương thức của lớp list**

### size():Trả về số lượng phần tử của list.

### Ví dụ: Tạo một list  có kích thước là 10 và in ra số lượng phần tử của list này.

### #include <iostream>

### #include <list>

### using namespace std;

### int main()

### {

### list<int> myList (10) ;

### cout << "The size of list is: " << myList .size()<< endl;

### return 0;

### }

### empty(): Trả về true(1) nếu list rỗng, ngược lại là false (0)

Ví dụ: Kiểm tra list có rỗng không và  in ra thông báo tương ứng.

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<int> myList ;

if( myList.empty()) {

cout<<"List is empty! "<<endl ;

}

else {

cout<<"List is not empty! "<<endl ;

}

return 0;

}

### front(): Truy xuất phần tử đầu tiên của list. Truy xuất ở đây bao gồm in hoặc chỉnh sửa giá trị.

Ví dụ: Tạo list có 5 phần tử có giá trị lần lượt là a,b,c,d,e. Thay đổi phần tử đầu tiên thành ‘A’ và in ra kết quả.

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<char> myList;

// create a list

for(int i = 0 ; i<5; i++) {

myList.push\_back('a'+ i ) ; // a b c d e

}

// change list

cout<<"Before change, front is: "<<myList.front()<<endl ;

myList.front() = 'A' ;

cout<<"After change, front is: "<<myList.front()<<endl ;

return 0;

}

#### back():Truy xuất phần tử cuối cùng của list. Truy xuất ở đây bao gồm in hoặc chỉnh sửa giá trị.

Ví dụ: Tạo list có 5 phần tử có giá trị lần lượt là  A,B,C,D,E. Thay đổi phẩn tử cuối cùng thành ‘e’ và in ra kết quả.

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<char> myList;

// create a list

for(int i = 0 ; i<5; i++) {

myList.push\_back('A'+ i ) ; // A B C D E

}

// change list

cout<<"Before change, back is: "<<myList.back()<<endl ;

myList.back() = 'e' ;

cout<<"After change, back is: "<<myList.back()<<endl ;

return 0;

}

### push\_back(const x):Thêm phần tử có giá trị x vào cuối list.

Phương thức push\_back(x) thường dùng để khởi tạo list một cách “động”, có nghĩa là kích thước list chưa xác định trước.

Ví dụ:Viết chương trình cho người dùng nhập vào kích thước list và nhập vào giá trị các phần tử. Lưu trữ các phần tử theo **đúng thứ tự** đã nhập. In ra các phần tử trong list.

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<int> myList;

int n; // size of list

cout<<"Size: " ;

cin>>n ;

// create list

int tempNumber ;

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

cin>>tempNumber ;

myList.push\_back(tempNumber ) ;

}

// print list

list<int>::iterator it ;

for( it = myList.begin() ; it!=myList.end(); it++) {

cout<<\*it<<" " ;

}

return 0;

}

Lưu ý: để in tất các phần tử trong list, ta dùng vòng lặp for với biến là iterator. Ta không thể dùng toán tử dấu ngoặc vuông “[ ]” như vector. Ví dụ, sử dụng truy xuất list như bên dưới là sai:

// WRONG way to access elements of list

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

cout<<myList[i]<<" " ;

}

### push\_front(const x): Thêm phần tử có giá trị x vào đầu list.

### Ví dụ: Khởi tạo list có các phần tử là kí tự lần lượt theo thứ tự là ‘B’,’C’,’D’,’E’. Thêm phần tử ‘A’ vào đầu danh sách. In các phần tử theo đúng thứ tự lưu trữ.

### #include <iostream>

### #include <list>

### using namespace std;

### int main()

### {

### list<char> myList;

### // create list

### for(int i = 0 ; i<4; i++) {

### myList.push\_back('B'+i ) ; // B C D E

### }

### // insert A into the head of list

### myList.push\_front('A');

### // print list

### list<char>::iterator it;

### for( it =myList.begin() ; it!=myList.end(); it++) {

### cout<<\*it<<" " ;

### }

### return 0;

### }

#### **pop\_back():** Loại bỏ phần tử cuối cùng ra khỏi list.

Ví dụ: Viết chương trình cho người dùng nhập vào kích thước list và nhập vào giá trị các phần tử. Sau đó, loại bỏ đi một nửa các phần tử ở cuối list.  In ra các phần tử trong list.

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<int> myList;

int n; // size of list

cout<<"Size: " ;

cin>>n ;

// create list

int tempNumber ;

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

cin>>tempNumber ;

myList.push\_back(tempNumber ) ;

}

// delete element

for(int i = 0 ; i<n/2; i++) {

myList.pop\_back() ;

}

// print list

list<int>::iterator it ;

for(it = myList.begin() ; it!= myList.end(); it++) {

cout<<\*it<<" " ;

}

return 0;

}

### pop\_front(): Loại bỏ phần tử đầu tiên ra khỏi list.

### Ví dụ: Tạo danh sách có 5 phần tử lần lượt theo thứ tự là ‘a’,’b’,’c’,’d’,’e’. Loại bỏ phần tử đâu tiên ra khỏi list và in ra màn hình các phần tử còn lại.

### #include <iostream>

### #include <list>

### using namespace std;

### int main()

### {

### list<char> myList;

### // create list

### for(int i = 0 ; i<5; i++) {

### myList.push\_back('a'+i ) ; // a b c d e

### }

### // remove a out of list

### myList.pop\_front();

### // print list

### list<char>::iterator it;

### for( it=myList.begin() ; it!=myList.end(); it++) {

### cout<<\*it<<" " ;

### }

### return 0;

### }

### insert

insert có thể dùng với 3 cách sau:

insert (iterator positon,const x): Chèn phần tử có giá trị x vào **trước** vị trí position.

insert (iterator positon,int n, const x): Chèn n phần tử có giá trị x vào**trước** vị trí position.

insert (iterator positon,iterator a, itertator b): Chèn vào **trước** vị trí position của list hiện hành tất cả các phần tử trong nửa khoảng [a,b) của một list khác.

Ví dụ 1: Tạo list có 5 phần tử đều có giá trị là 100. Chèn phần tử có giá trị 200 vào vị trí đầu tiên của list.

// inserting into a list

#include <iostream>

#include <list>

using namespace std ;

int main ()

{

// create list

list<int> myList (5,100); // 100 100 100 100 100

// create iterator

list<int>::iterator it = myList.begin() ;

it++ ; // it points now to 2rd postion

// insert 200 into 2st position of list

myList.insert(it,200) ; // 100 200 100 100 100 100

// print list

for(it= myList.begin(); it != myList.end() ; it++) {

cout<<\*it<<" " ;

}

return 0;

}

Lưu ý: Ta phải dùng lệnh it++ để tăng biến iterator it lên 1 đơn vị rồi sau đó mới dùng lệnh: myList.insert(it,200) để  chèn vào vị trí thứ 2. Ta không thể dùng lệnh myList.insert(it+1,200)  như đối với vector vì list không thể truy cập phần tử qua toán tử dấu ngoặc vuông “[]”.

Ví dụ 2: Tạo list có 5 phần tử đều có giá trị là 100. Chèn 2 phần tử có giá trị 200 vào vị trí thứ 3  của list.

// inserting into a list

#include <iostream>

#include <list>

using namespace std ;

int main ()

{

// create list

list<int> myList (5,100); // 100 100 100 100 100

// create iterator

list<int>::iterator it = myList.begin() ;

advance(it,2) ; // it points now to 3rd postion

// insert 200 into 3st position of list

myList.insert(it,2,200) ; // 100 100 200 200 100 100 100

// print list

for(it= myList.begin(); it != myList.end() ; it++) {

cout<<\*it<<" " ;

}

return 0;

}

Chú ý: Ta dùng lệnh advance(it,2) để tăng it lên 2 đơn vị.

void advance (iterator& it, int n): là phương thức con của đối tượng iterator và chỉ dùng phương thức này thao tác được với iterator. Ta không thể dùng lệnh it+=2 hoặc it=it+2.

Ví dụ 3: Tạo list thứ nhất có 5 phần tử đều có giá trị là 100. Tạo list thứ hai có 3 phần tử đều có giá trị là 200. Chèn tất cả các phần tử của list thứ hai vào vị trí thứ 2 của list thứ nhất.

// inserting into a list

#include <iostream>

#include <list>

using namespace std ;

int main ()

{

// create list

list<int> myList (5,100); // 100 100 100 100 100

list<int> otherList(3,200) ; // 200 200 200

// create iterator

list<int>::iterator it = myList.begin() ;

it++ ; // it points now to 2rd postion

// insert 2rd list into 1st list

myList.insert(it, otherList.begin(),otherList.end()) ; // 100 200 200 100 100 100 100

// print list

for(it= myList.begin(); it != myList.end() ; it++) {

cout<<\*it<<" " ;

}

return 0;

}

Lưu ý: Ngoài việc chèn list này vào list khác, ta có thể chèn một mảng các phần tử vào list.

#include <iostream>

#include <list>

using namespace std ;

int main ()

{

// create list

list<int> myList (5,100); // 100 100 100 100 100

//create array

int myArray[3] = {200,300,400} ; // 200 300 400

// insert two element from array to list

myList.insert(myList.begin(), myArray, myArray +2 ) ; // 100 200 200 100 100 100 100

// print list

list<int>::iterator it;

for(it= myList.begin(); it != myList.end() ; it++) {

cout<<\*it<<" " ;

}

return 0;

}

### Erase():Xóa phần tử ở vị trí position ra khỏi list

Phương thức erase có 2 cách dùng:

erase (iterator position): Xóa phần tử ở vị trị position

erase (iterator first, iterator last): Xóa tất cả các phần tử trong nửa khoảng [frist,last), có nghĩa là từ phẩn tử thứ first đến phần tử thứ (last-1).

Ví dụ 1: Tạo một list có 10 phần tử có giá trị từ 1 đến 10. Xóa phần tử thứ 6 và in ra các phần tử còn lại.

#include <iostream>

#include <list>

using namespace std ;

int main () {

list<int> myList;

// set some values (from 1 to 10)

for (int i=0; i<10; i++) {

myList.push\_back(i+1); // 1 2 3 4 5 6 7 8 9 10

}

// erase the 6th element

list<int>::iterator it;

it = myList.begin() ;

advance(it,5) ;

myList.erase (it);

// print list

for(it= myList.begin(); it != myList.end() ; it++) {

cout<<\*it<<" " ;

}

return 0;

}

Ví dụ 2: Tạo một list có 10 phần tử có giá trị từ 1 đến 10. Xóa  3 phần tử ở đầu list.

#include <iostream>

#include <list>

using namespace std ;

int main () {

list<int> myList;

// set some values (from 1 to 10)

for (int i=0; i<10; i++) {

myList.push\_back(i+1); // 1 2 3 4 5 6 7 8 9 10

}

// erase the first 3 elements

list<int>::iterator it ;

it = myList.begin() ;

advance (it,3);

myList.erase (myList.begin() ,it);

// print list

for (it = myList.begin(); it!= myList.end() ; it++) {

cout <<\*it<<" " ;

}

return 0;

}

### swap(list L): Hoán đổi list L với list hiện hành.

Ví dụ: Cho hai list, list thứ nhất có 5 phần tử đều là số 1, list thứ hai có 5 phần tử đều là số 2. Hoán đổi hai list trên và in ra các phần tử của mỗi list.

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<int> firstList (5, 1); // 1 1 1 1 1

list <int> secondList (5,2) ; // 2 2 2 2 2

// exchange 2 list

firstList.swap(secondList) ;

// create itertator

list<int>::iterator it ;

// print list 1

for( it = firstList.begin() ; it != firstList.end(); it++) {

cout<<\*it<<" " ;

}

cout<<endl ;

// print list 2

for(it = secondList.begin() ; it != secondList.end(); it++) {

cout<<\*it<<" " ;

}

return 0;

}

### clear():Xóa hoàn toàn một list.

Ví dụ: Tạo một list có 5 phần tử đều có giá trị là 5. In kích thước của list trước và sau khi dùng phương thức clear().

#include <iostream>

#include <list>

using namespace std;

int main()

{

list<int> myList (5,1); // 1 1 1 1 1

cout<<"Before clear list, size of list is : " << myList.size()<< endl ;

myList.clear() ; // resize list

cout<<"After clear list, size of list is : " << myList.size()<< endl ;

return 0;

}

Lưu ý: Khác với vector, phương thức clear() của list sẽ xóa đi hoàn toàn danh sách.

### Splice

splice (iterator positon,list& other): ghép list other vào vị trí position của list hiện hành, sau khi nối ghép, list other không còn phần tử nào.

splice (iterator positon,list& other, iterator first, iterator last): ghép các phần tử trong nửa khoảng [first,last) của list other và list hiện hành. Sau khi ghép nối, list other chỉ còn lại các phần tử trong nửa đoạn [ last, L.end() )

splice (iterator positon, list& other, iterator it ): ghép phần tử có vị trí là it của list other vào vị trí position của list hiện hành. Sau khi ghép nối, list other bị mất đi phần tử ở vị trí it.

Ví dụ 1: Tạo danh sách thứ nhất có 5 phần tử đều có giá trị là 100. Tạo danh sách thứ hai có 3 phần tử đều có giá trị là 200. Ghép danh sách thứ 2 vào đầu danh sách thứ nhất.

#include <iostream>

#include <list>

using namespace std ;

int main ()

{

// create list

list<int> myList1 (5,100); // 100 100 100 100 100

list<int> myList2(3,200) ; // 200 200 200

// create iterator

list<int>::iterator it = myList1.begin() ;

// splice list 2 into list 1 at 1st position of list 1

myList1.splice(it,myList2) ; //

// print list 1

for(it= myList1.begin(); it != myList1.end() ; it++) {

cout<<\*it<<" " ;

}

return 0;

}

Ví dụ 2: Tạo danh sách thứ nhất có 5 phần tử đều có giá trị là 100. Tạo danh sách thứ hai có 5 phần tử đều có giá trị là 200. Ghép 2 phần tử đầu củadanh sách thứ 2 vào đầu danh sách thứ nhất.

#include <iostream>

#include <list>

using namespace std ;

int main ()

{

// create list

list<int> myList1 (5,100); // 100 100 100 100 100

list<int> myList2(5,200) ; // 200 200 200 200 200

// create iterator

list<int>::iterator first = myList2.begin() ;

list<int>::iterator last = first;

advance(last,2) ;

// splice 2 first elements from list 2 into list 1 at 1st position of list 1

myList1.splice(myList1.begin(),myList2, first, last ) ; //

// print every list

list<int>::iterator it ;

cout<<"List 1: " ;

for(it= myList1.begin(); it != myList1.end() ; it++) {

cout<<\*it<<" " ;

}

cout<<endl;

cout<<"list 2: " ;

for(it= myList2.begin(); it != myList2.end() ; it++) {

cout<<\*it<<" " ;

}

return 0;

}

Ví dụ 3: Tạo list thứ nhất có các phần tử là a,b,c,d,e. Tạo list thứ hai có các phần tử là A,B,C,D,E. Chuyển phần tử C của list thứ hai vào vị trí cuối cùng của list thứ nhất.

#include <iostream>

#include <list>

using namespace std ;

int main ()

{

// create list 1

list<char> myList1 ;

for(int i = 0 ; i < 5 ; i++) {

myList1.push\_back('a'+i) ; // a b c d e

}

// create list 2

list<char> myList2 ;

for(int i = 0 ; i < 5 ; i++) {

myList2.push\_back('A'+i) ; // A B C D E

}

// create iterator it

list<char>::iterator it = myList2.begin() ;

advance(it,2) ;

// slice 3rd element from list 2 into list 1

myList1.splice(myList1.end(),myList2, it ) ;

// print every list

cout<<"List 1: " ;

for(it= myList1.begin(); it != myList1.end() ; it++) {

cout<<\*it<<" " ;

}

cout<<endl;

cout<<"list 2: " ;

for(it= myList2.begin(); it != myList2.end() ; it++) {

cout<<\*it<<" " ;

}

return 0;

}

### remove (const val): Loại bỏ tất cả phần tử có giả trị bằng val trong list.

Ví dụ: Tạo một danh sách có các phần tử lần lượt là 10,15,15,20,30,40. Loại bỏ phần tử có giá trị bằng 15 ra khỏi danh sách.

// remove from list

#include <iostream>

#include <list>

using namespace std ;

int main ()

{

// create list

int myArray[]= {10,15,15,20,30,40};

std::list<int> myList (myArray,myArray+6);

// remove number 15

myList.remove(15);

// print list

cout << "myList contains: ";

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

cout <<\*it<<" ";

}

return 0;

}

### remove\_if (predicate pred): Loại bỏ tất các phần tử trong list nếu pred trả về giá trị  true. Predicate có thể là một hàm (function) hoặc một class.

Ví dụ 1: Tạo một list có các phần tử lần lượt là 1, 10, 2, 20, 3, 30, 4, 40. Loại bỏ các phần tử có 1 kí tự ra khỏi list.

// list::remove\_if

#include <iostream>

#include <list>

using namespace std;

// a predicate implemented as a function:

bool single\_digit (const int& value) {

return (value<10);

}

int main ()

{

int myArray[]= {1,10,2,20,3,30,4,40};

std::list<int> mylist (myArray,myArray+8); // 1 10 2 20 3 30 4 40

// remove with predicate is as a function

mylist.remove\_if (single\_digit); // 10 20 30 40

// print list

cout << "mylist contains:";

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

cout << \*it<<" ";

}

return 0;

}

Ví dụ 2: Tạo một list có các phần tử lần lượt là 1, 2, 3, 4, 5, 6, 7, 8, 9. Loại bỏ các phần tử lẻ ra khỏi list.

// list::remove\_if

#include <iostream>

#include <list>

using namespace std;

// a predicate implemented as a class:

struct is\_odd {

bool operator() (const int& value) {

return (value%2)==1;

}

};

int main ()

{

int myArray[]= {1,2,3,4,5,6,7,8,9};

std::list<int> mylist (myArray,myArray+9); // 1 2 3 4 5 6 7 8 9

// remove with predicate is as a function

mylist.remove\_if (is\_odd()); // 2 4 6 8

// print list

cout << "mylist contains:";

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

cout << \*it<<" ";

}

return 0;

}

Lưu ý: khi sử dụng predicate như là một class, thì khi truyền tham số cho hàm remove\_if ta phải thêm dấu ngoặc đơn () như trong lệnh : mylist.remove\_if (is\_odd());

### sort

có hai hình thức sử dụng sort là :

sort(): Sắp xếp các phần tử của list theo chiều tăng dần.

sort (compare comp) : Sắp xếp các phần tử của list tăng dần theo tiêu chí so sánh comp. Biểu thức compare comp có thể là một hàm trả về giá trị bool.

Ví dụ: Tạo list chứ các phần tử từ 1 đến 9. Sắp xếp lại các phần tử theo chiều từ 9 đến 1.

// list::sort

#include <iostream>

#include <list>

using namespace std;

int main ()

{

int myArray[]= {9,8,7,6,5,4,3,2,1};

std::list<int> mylist (myArray,myArray+9); //9 8 7 6 5 4 3 2 1

// ascend sort

mylist.sort() ;

// print list

cout << "mylist contains:";

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

cout << \*it<<" ";

}

return 0;

}

Ví dụ 2: Tạo list chứa các phần tử kiểu string bao gồm “AAAA”,”BBB”,”CC”,”D”. Sắp xếp lại các phần tử trong list theo chiều tăng dần độ dài của chuỗi.

// list::sort

#include <iostream>

#include <list>

using namespace std;

bool compare\_string (const string& first, const string& second)

{

if(first.length() < second.length())

return true;

else

return false ;

}

int main ()

{

//create list

string myArray[]= {"AAAA","BBB","CC","D"};

std::list<string> mylist (myArray,myArray+4);

// ascend sort depend on length

mylist.sort(compare\_string) ;

// print list

cout << "mylist contains:";

for (list<string>::iterator it = mylist.begin(); it!=mylist.end(); it++) {

cout << \*it<<" ";

}

return 0;

}

### reverse():Đảo ngược lại các phần tử của list.

Ghi chú: Ta có thể dùng phương thức reverse() phối hợp với phương thức sort() để sắ xếp các phần tử theo chiều giảm dần. Bởi vì mật định hàm sort() là sắp xếp tăng dần do đó sau khi dùng hàm sort(), ta dùng thêm hàm reverse() để lại ngược lại các phần tử.

Ví dụ: Tạo list có các phần tử lần lượt là 0,1,2,3,4,5,9,8,6,5. Sắp xếp lại các phần tử theo chiều giảm dần.

// list::reverse

#include <iostream>

#include <list>

using namespace std;

int main ()

{

int myArray[]= {0,1,2,3,4,5,9,8,6,5};

std::list<int> mylist (myArray,myArray+10); //0 1 2 3 4 5 9 8 6 5

mylist.sort() ;// 0 1 2 3 4 5 6 7 8 9

//reverse list

mylist.reverse() ; // 9 8 7 6 5 4 3 2 1 0

// print list

cout << "mylist contains:";

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

cout << \*it<<" ";

}

return 0;

}

### unique

unique(): Loại bỏ các phần tử thích hợp sao cho các phần tử còn lại trong list chỉ xuất hiện đúng duy nhất 1 lần.

unique (BinaryPredicate binary\_pred): Loại bỏ các phần tử thích hợp sao cho các phần tử còn lại trong list chỉ xuất hiện đúng duy nhất 1 lần dựa theo tiêu chí của hàm binary\_pred. Biểu thức binary\_pred có thể là hàm (function) hoặc lớp (class).

Lưu ý: Các phần tử trong list phải được sắp xếp, nếu không sắp xếp thì phương thức unique sẽ cho kết quả sai.

Ví dụ 1: Tạo list bao gồm các phần tử có giá trị lần lượt là 1, 2, 3, 4, 1, 2, 3, 4. Loại bỏ các phần tử sau cho các phần tử chỉ xuất hiện 1 lần.

// list::unique

#include <iostream>

#include <list>

using namespace std;

int main ()

{

int myArray[]= {1,2,3,4,1,2,3,4};

list<int> mylist (myArray,myArray+8); //1 2 3 4 1 2 3 4

// sort elements in list

mylist.sort() ;// 1 1 2 2 3 3 4 4

//reverse list

mylist.unique() ; // 1 2 3 4

// print list

cout << "mylist contains: ";

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

cout << \*it<<" ";

}

return 0;

}

Ví dụ 2: Cho một list bao gồm các 8 số thập phân sau: 1.1, 2.1, 3.1, 4.1, 1.2, 2.2, 3.2, 4.2. Giả sử các số thập phân có phần nguyên giống nhau thì được xem là một số duy nhất. Hãy loại bỏ các phần tử sau cho các phần tử chỉ xuất hiện 1 lần.

// list::unique

#include <iostream>

#include <list>

using namespace std;

bool same\_integral\_part (double first, double second)

{

return ( int(first)==int(second) );

}

int main ()

{

double myArray[]= {1.1,2.1,3.1,4.1,1.2,2.2,3.2,4.2};

list<double> mylist (myArray,myArray+4); //1.1 2.1 3.1 4.1 1.2 2.2 3.2 4.2

// sort elements in list

mylist.sort() ;// 1.1 1.2 2.1 2.2 3.1 3.2 4.1 4.2

//reverse list

mylist.unique(same\_integral\_part) ; // 1 2 3 4

// print list

cout << "mylist contains: ";

for (list<double>::iterator it = mylist.begin(); it!=mylist.end(); it++) {

cout << \*it<<" ";

}

return 0;

}

### merge

Có hai cách sử dụng chính như sau:

**merge (list& L):**Ghép list L vào sau vị trí cuối cùng list hiện hành. Sau khi ghép nối, list L không còn phần tử nào.

**merge (list& L, Compare comp):** Ghép các phần tử của list L vào list hiện hành với vị trí dựa theo biểu thức comp. Biểu thức comp là một hàm trả về giá trị bool.

Ví dụ 1: Tạo list thứ nhất có 5 phần tử đều có giá trị 100, list thứ hai có các 3 phần tử đều có giá trị 200. Hãy di chuyển tất cả các phần tử của list thứ hai vào sau list thứ nhất.

// list::merge

#include <iostream>

#include <list>

using namespace std;

int main ()

{

list<int> myList1(5,100) ;

list<int> myList2(3,200) ;

myList1.merge(myList2) ;

cout << "myList1 contains: ";

for (list<int>::iterator it = myList1.begin(); it!=myList1.end(); it++) {

cout << \*it<<" ";

}

cout <<endl<< "myList2 contains: ";

for (list<int>::iterator it = myList2.begin(); it!=myList2.end(); it++) {

cout <<\*it<<" ";

}

return 0;

}

Ví dụ 2: Tạo list thứ nhất bao gồm 3 phần tử 3.1,  2.2 , 2.9. Tạo list thứ hai bao gồm 4 phần tử 3.7, 7.1, 1.4, 2.1. Hãy trộn các phần tử của list thứ hai vào list thứ nhất dựa theo biểu thức comp như sau:

bool mycomparison (double first, double second)

{

return ( (first)>(second) );

}

Hãy in các phần tử trong danh sách thứ nhất sau khi trộn.

// list::merge

#include <iostream>

#include <list>

using namespace std ;

// compare only integral part:

bool mycomparison (double first, double second)

{

return ( (first)>(second) );

}

int main ()

{

list<double> first, second;

first.push\_back (3.1);

first.push\_back (2.2);

first.push\_back (2.9); // 3.1 2.2 2.9

second.push\_back (3.7);

second.push\_back (7.1);

second.push\_back (1.4);

second.push\_back (2.1); // 3.7 7.1 1.4 2.1

first.merge(second,mycomparison);

cout << "first contains:";

for (list<double>::iterator it=first.begin(); it!=first.end();it++) {

cout <<\*it<<" " ;

}

return 0;

}

1. **LỚP ARRAYLIST**

ArrayList là lớp cài đặt giao diện List. Lớp ArrayList hỗ trợ các mảng động mà có thể tăng kích cỡ nếu cần.

Các mảng Java chuẩn là có độ dài cố định. Sau khi các mảng được tạo, chúng không thể tăng hoặc giảm kích cỡ, nghĩa là bạn phải có bao nhiêu phần tử mà một mảng sẽ giữ.

ArrayList được tạo với một kích cỡ ban đầu. Khi kích cỡ này bị vượt, collection tự động được tăng. Khi các đối tượng bị gỡ bỏ, ArrayList có thể bị giảm kích cỡ.

Lớp này được ưa chuộng trong java vì chức năng và tính linh hoạt mà nó cung cấp. Hầu hết các lập trình viên đều chọn ArrayList thay thế cho cách dùng Array truyền thống.  
Kiểu dữ liệu mạng truyền thống Array bị giới hạn với kích thước xác định. Nếu một số phần tử trong mảng bị xóa đi thì bộ nhớ chứa các phần tử đó không được thu hồi. Ngược lại kích thước của ArrayList có thể tăng hoặc giảm tùy thích.

**Các phương thức thường dùng:**

**add( Object o):** Phương thức thêm phần tử vào mảng

CODE: obj.add(“Xin chao!”);

Dòng lệnh này sẽ thêm chuỗi "Xin chao" vào cuối mảng.

**add(int index, Object o):** Phương thức thêm phần tử vào chỉ mục đã cho

CODE: obj.add(2, “Tam biet”);

Dòng lệnh này thêm chuỗi "Tam biet" vào mảng ở chỉ mục số 2(tức là vị trí thứ 3 vì thứ tự trong mảng tính từ 0).

**remove(Object o):** Xóa phần tử trong mảng

CODE: obj.remove(“Sinh vien 3”);

Dòng lệnh này xóa phần tử chuỗi "Sinh viên 3” trong mảng.

**remove(int index):** Xóa phần tử theo chỉ mục

CODE: obj.remove(3);

Dòng lệnh này xóa phần tử ở chỉ mục số 3 trong mảng(tức là vị trí thứ 4 trong mảng).

**set(int index, Object o)**: Phương thức cập nhận phần tử trong mảng, nó thay thế phần tử ở chỉ mục đã cho bằng phần tử mới.

CODE: obj.set(2, "Sinh viên X");

Dòng lệnh này thay thế phần tử ở chỉ mục 2 bằng chuỗi "Sinh viên X".

**int indexOf(Object o):** Phương thức lấy chỉ mục của một phần tử, nếu kết quả trả về -1 thì phần tử không tồn tại trong mảng.

CODE: int pos=obj.indexOf(“Sinh vien”);

Dòng lệnh này trả về chỉ mục của phần tử trong mảng.

**Object get(int index):** Phương thức lấy phần tử ở chỉ mục đã cho trong mảng.

CODE: String str= obj.get(2);

Dòng lệnh này trả về phần tử ở chỉ mục số 2

**int size():** Phương thức xác định kích thước mảng.

CODE: int numberofitems = obj.size();

**boolean contains(Object o):** Phương thức kiểm tra phần tử có tồn tại trong mảng hay không.

CODE: obj.contains(“Sinh vien 4”);

Dòng lệnh này sẽ trả về true nếu phần tử "Sinh viên 4" có trong mảng, trả về false nếu không.

**clear():** Phương thức xóa tất cả các phần tử trong mảng.

CODE: obj.clear();

Dòng lệnh này sẽ xóa hết các phần tử trong mảng, mảng trở thành mảng rỗng.

1. **SỰ GIỐNG VÀ KHÁC NHAU CỦA CÁC LỚP VECTOR, LIST, ARRAYLIST**

Vector có cơ chế đồng bộ hóa, array list thì không.

Vector không hỗ trợ hàm sort.

Array list nhanh hơn vector.

Các phương thức trong Vector là đồng bộ, sử dụng nhiều trong muti – threaded và môi trường xử lí đồng thời.

Vector có thể trả về liệt kê, array list thì không.

So sánh với  dãy các phần tử chứa khác (array, vector), lists biểu diễn tốt hơn trong việc chèn, thêm, xóa.

So với cơ sở chứa chuỗi tiêu chuẩn khác (array, vector), danh sách thực hiện nói chung tốt hơn trong việc chèn , thêm và di chuyển các phần tử trong bất kỳ vị trí nào trong dãy chứa, và do đó, một số thuật toán sẽ hiệu quả hơn, vd như thuật toán sắp xếp.

Duyệt danh sách dùng iterator tương tự như với vector.