**Bài 3: List - Khái niệm cơ bản - Sử dụng thư viện chuẩn STL cho C/C++**

**Đăng bởi: Admin | Lượt xem: 7133 | Chuyên mục: C/C++**

**Khái niệm :**

List là một danh sách chứa các đối tượng (các nút(node) – lưu trữ các thông tin dữ liệu và địa chỉ của nút kế tiếp, nút trước đó) liên kết với nhau và cho phép chèn thêm hay xóa bất kì một đối tượng nào trong danh sách. Là một cấu trúc dữ liệu cơ bản để biết thêm các bạn tham khảo tại [danh sách liên kết đơn.](https://www.stdio.vn/articles/read/106/danh-sach-lien-ket-don)

**Ưu và nhược điểm :**

**Ưu điểm** : chèn và loại bỏ phần tử ở bất cứ vị trí nào trong container nếu biết được interator trỏ đến phần tử đó, (với độ phức tạp là O(1)). Nếu chưa biết interator của phần tử cần xóa hoặc ở vị trí cần chèn thì có thể tìm iterator đó thông qua begin() hoặc end().

**Nhược điểm :**khả năng truy cập tới phần tử, nó không thể truy xuất A[0] hay A[3] được mà phải thông qua vị trí đầu hoặc cuối của list. ( với độ phức tạp là O(n)).

\*\**Lưu ý* : end() không phải là iterator trỏ tới phần tử cuối cùng mà trỏ tới sau phần tử cuối cùng.

**Ví dụ cơ bản :**

Đầu tiên ta phải khai báo thư viện :

#**include**<list>

Một số cách khai báo list thường dùng là:

Khai báo list rỗng:

list<kiểu\_dữ\_liệu> tên\_list;

// ví dụ list<int> a;

Khai báo list khi biết trước size của nó:

list<kiểu\_dữ\_liệu> tên\_list(Size);

//ví dụ list<int> a(5); a =[0,0,0,0,0]

Khai báo list khi biết trước size và giá trị khởi tạo:

list<kiểu\_dữ\_liệu> tên\_list(Size,value);

//ví dụ list<int> a(3,2); a =[2,2,2]

Duyệt list:

Khi duyệt các phần tử trong list chúng ta phải làm quen với 1 kiểu dữ liệu là iterator, hiểu đơn giản thì đây là một con trỏ.

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

VD1 : Cho một số tự nhiên n. Hãy khởi tạo một list chứa lần lượt các số nguyên từ 1 đến n

list<**int**> **initList**(**int** n)

{

list<**int**> a;

**for** (**int** i = 1; i <= n; i++){

a.push\_back(i);

}

**return** a;

}

vector<**int**> **verifyFunction**(**int** n)

{

list<**int**> lst = initList(n);

vector<**int**> **res**(lst.begin(), lst.end());

**return** res;

}

VD2 : Cho một list line gồm các số nguyên. Hãy tính tổng phẩn tử đầu tiên và phần tử cuối cùng trong list đó, nếu list rỗng trả về -1, còn nếu list chỉ có một phần tử thì trả về phần tử đó.

*Gợi ý :*

* Với line = [1, 2, 3, 4], thì verifyFunction(line) = 5.
* Với line = [7], thì verifyFunction(line) = 7.
* Để lấy giá trị đầu tiên trong list, ta dùng hàm front().
* Để lấy giá trị cuối cùng trong list, ta dùng hàm back().
* Để kiểm tra list có rỗng hay không, ta dùng hàm empty() (hàm trả về true nếu list rỗng, ngược lại trả về false).
* Để lấy kích thước (số phần tử) của list, ta dùng hàm size().

\*\* *Lưu ý :*Trong list không thể truy vấn đến các phần tử giống như mảng và vector, không thể sử dụng như a[0], a[1], a[2],

*Code mẫu :*

**int** **sumOfFirstAndLastElement**(list<**int**> linkedList)

{

**if** (linkedList.size() == 0) **return** -1;

**if** (linkedList.size() == 1) **return** linkedList.front();

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

}

**int** **verifyFunction**(vector<**int**> v)

{

list<**int**> **l**(v.begin(), v.end());

**return** sumOfFirstAndLastElement(l);

}

Bài sau chúng ta sẽ tiếp tục với bài tập về List để hiểu sâu hơn. Chúc các bạn học vui vẻ <3

**Bài 4: Các hàm thông dụng của List - Sử dụng thư viện chuẩn STL cho C/C++**

**Đăng bởi: Admin | Lượt xem: 3618 | Chuyên mục: C/C++**

**Hàm back():**

Hàm có chức năng tham chiếu đến phần tử cuối cùng trong danh sách chứa . Không giống như list.end() trả về con trỏ iterator phần tử cuối cùng, hàm này trả về một tham chiếu trực tiếp

Ví dụ :

Thêm vào list 10 phần tử và giá trị từ 10 về 0 và in ra giá trị của các phần tử trong list :

// list::back

#**include** <iostream>

#**include** <list>

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

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

{

list<**int**> mylist;

mylist.push\_back(10);

cout << mylist.back();

**while** (mylist.back() != 0)

{

mylist.push\_back(mylist.back() - 1);

cout << mylist.back()<< endl;

}

cout << "mylist contains:";

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

cout << ' ' << \*it;

cout << '\n';

system("pause");

**return** 0;

}

Kết quả :

mylist contains: 10 9 8 7 6 5 4 3 2 1 0

Các bạn chú ý vài điểm sau :

* Ở trên các bạn để ý sử dùng hàm push\_back(10) : khởi tạo list với 1 phần tử có giá trị bằng 10
* while (mylist.back() != 0) : có nghĩa khi giá trị của node bằng 0 thì vòng lặp while dừng lại
* mylist.push\_back(mylist.back() - 1) :  thêm phần tử có giá trị mylist.back() - 1 vào list
* Sau đó sử dụng vòng lặp for để duyệt danh sách và in ra các phần tử của danh sách đó

*Lưu ý* : Trong List chỉ có thể sử dụng con trỏ ierator để duyệt danh sách

**Hàm list.begin() và list.end():**

Trong khi duyệt danh sách chúng ta thường xuyên sử dụng l.begin() và l.end() để trỏ đến vị trí đầu và cuối của danh sách

Ví dụ sau đây là duyệt một list gồm 5 phần tử và in ra giá trị của 5 phần tử đó

// list::begin

#**include** <iostream>

#**include** <list>

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

{

**int** myints[] = { 75, 23, 65, 42, 13 };

std::list<**int**> **mylist**(myints, myints+5);

std::cout << "mylist contains:";

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

std::cout << ' ' << \*it;

std::cout << '\n';

system("pause");

**return** 0;

}

* mylist(myints, myints+5) : đổ mảng vào list với 5 phần tử, sử dụng allocator
* Trong vòng lặp for sử dụng mylist.begin() ,  mylist.begin() là điều kiện để bắt đầu và kết thúc vòng lặp
* Nếu duyệt ngược danh sách có thể dùng  mylist.rbegin() và  mylist.rend()

Kết quả :

mylist contains: 75 23 65 42 13

**Hàm list.clear():**

Hàm có tác dụng xóa tất cả các phần tử trong danh sách và biến danh sách có kích thước bằng 0

// clearing lists

#**include** <iostream>

#**include** <list>

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

{

std::list<**int**> mylist;

std::list<**int**>::iterator it;

mylist.push\_back (100);

mylist.push\_back (200);

mylist.push\_back (300);

std::cout << "mylist contains:";

**for** (it=mylist.begin(); it!=mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

mylist.clear();

mylist.push\_back (1101);

mylist.push\_back (2202);

std::cout << "mylist contains:";

**for** (it=mylist.begin(); it!=mylist.end(); ++it)

std::cout << ' ' << \*it;

std::cout << '\n';

**return** 0;

}

Kết quả :

mylist contains: 100 200 300

mylist contains: 1101 2202

**Hàm list.empty() :**

Hàm này có chức năng kiểm tra danh sách chứa có rỗng hay không.

Ví dụ

// list::empty

#**include** <iostream>

#**include** <list>

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

{

std::list<**int**> mylist;

**int** **sum** (0);

**for** (**int** i=1;i<=10;++i) mylist.push\_back(i);

**while** (!mylist.empty())

{

sum += mylist.front();

mylist.pop\_front();

}

std::cout << "total: " << sum << '\n';

**return** 0;

}

Ở đây ta duyệt vòng lặp và thêm vào danh sách 10 phần tử với giá trị từ 1 đến 10, sau đó ta sử dụng !mylist.empty() là điều kiện vòng lặp hoạt động, nếu danh sách rỗngòng lặp dừng và mỗi vòng lặp thì sẽ thực hiện 2 nhiệm vụ :

1. sum += mylist.front() : cộng giá trị đầu của danh sách
2. mylist.pop\_front() :  xóa phần tử đầu của danh sách đến khi danh sách rỗng

Kết quả :

total: 55

**Hàm list.earase() :**

Hàm có chức năng xóa một phần tử hoặc nhiều phần tử trong danh sách

// erasing from list

#**include** <iostream>

#**include** <list>

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

{

std::list<**int**> mylist;

std::list<**int**>::iterator it1,it2;

// set some values:

**for** (**int** i=1; i<10; ++i) mylist.push\_back(i\*10);

// 10 20 30 40 50 60 70 80 90

it1 = it2 = mylist.begin(); // ^^

advance (it2,6); // ^ ^

++it1; // ^ ^

it1 = mylist.erase (it1); // 10 30 40 50 60 70 80 90

// ^ ^

it2 = mylist.erase (it2); // 10 30 40 50 60 80 90

// ^ ^

++it1; // ^ ^

--it2; // ^ ^

mylist.erase (it1,it2); // 10 30 60 80 90

// ^

std::cout << "mylist contains:";

**for** (it1=mylist.begin(); it1!=mylist.end(); ++it1)

std::cout << ' ' << \*it1;

std::cout << '\n';

**return** 0;

}

Một số chú ý :

* Ở đây IT1 và IT2 là các con trỏ
* Hàm advance có ý nghĩa dịch vị trí của con trỏ đến vị trí nào đó
* it1 = mylist.erase (it1) : Xóa giá trị tại ví trí con trỏ it1

Kết quả :

mylist contains: 10 30 60 80 90

**Hàm list.remove():**

Đưa vào giá trị phần tử muốn xóa và danh sách sẽ xóa đi tất cả các phần tử có giá trị đưa vào

// remove from list

#**include** <iostream>

#**include** <list>

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

{

**int** myints[]= {17,89,7,14};

std::list<**int**> **mylist** (myints,myints+4);

mylist.remove(89);

std::cout << "mylist contains:";

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

std::cout << ' ' << \*it;

std::cout << '\n';

**return** 0;

}

Ở đây mylist.remove(89) xóa hết tất cả các phần tử có giá trị 89 nằm trong danh sách

Kết quả :

mylist contains: 17 7 14

**Hàm list.resize():**

Có 2 trường hợp thay đổi kích thước mảng như sau :

* Nếu n nhỏ hơn kích thước hiện tại của danh sách, danh sách sẽ giữ lại n phần tử trong danh sách và loại bỏ tất cả các phần tử phía sau
* Nếu n lớn hơn kích thước danh sách thì kích thước danh sách sẽ được mở rộng ra đúng n phần tử và giá trị của các phần tử bạn có thể đưa vào hoặc mặc định bằng 0

// resizing list

#**include** <iostream>

#**include** <list>

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

{

std::list<**int**> mylist;

// set some initial content:

**for** (**int** i = 1; i<10; ++i) mylist.push\_back(i);

mylist.resize(5);

mylist.resize(8, 100);

mylist.resize(12);

std::cout << "mylist contains:";

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

std::cout << ' ' << \*it;

std::cout << '\n';

system("pause");

**return** 0;

}

Kết quả:

mylist contains: 1 2 3 4 5 100 100 100 0 0 0 0