

MỘT SỐ KIẾN THỨC VỀ FOREACH, VECTOR, PAIR, TUPLE, SET & MAP TRONG C++

Contents

1. Vòng lặp for each trong C++11	3
1.1 Cú pháp vòng lặp for-each	3
1.2 Vòng lặp for-each và từ khóa auto	3
1.3 Vòng lặp for-each và tham chiếu.....	4
1.4 Vòng lặp for-each không làm việc với con trỏ mảng.....	4
1.5 Xem chỉ số phần tử hiện tại trong vòng lặp for-each?	5
2. Vector.....	5
2.1 Khai báo.....	5
2.2 push_back(...).	5
2.3 size()	5
2.4 Truy cập vào các phần tử	6
2.5 Duyệt các phần tử.....	7
2.6 Iterator.....	7
2.7 Khai báo vector với kích thước cố định.....	9
2.8 Lưu các kiểu dữ liệu khác	10
2.9 Vector Và Mảng 2 Chiều.....	11
3. Pair & Tuple.....	12
3.1. Kiểu Pair	12
3.1.1 Khai báo & khởi tạo.....	13
3.1.2 Toán Tử Với Pair	14
3.1.3 Một số hàm được sử dụng với pair	15
3.2 Kiểu tuple.....	18
3.2.1 Khởi tạo & khai báo.....	18
3.2.2 Operations on tuple.....	18
4. Set.....	23
4.1 Tính chất của set trong C++	23
4.2 Một số hàm cơ bản của set	24
4.2.1 insert() & size().....	24
4.2.2 Duyệt set	25
4.2.3 Hàm find()	26
4.2.4 Hàm count()	27
4.2.5 Hàm erase().....	28
4.2.6 Hàm lower_bound().....	28
4.2.7 Hàm upper_bound().....	29
Function of Set in C++ STL.....	30
4.3 Multiset trong C++.....	32

4.3.1 Hàm find()	33
4.3.2 Hàm count()	33
4.3.3 Hàm erase().....	34
4.3.4 Bài toán Sliding Window Maximum	35
List of Functions of Multiset	36
4.4 Unordered_set trong C++	38
std::unordered_set Member Methods	39
5. Map	40
5.1 Tính chất của map	40
5.2 Khởi tạo	41
5.3 Một số hàm cơ bản của map.....	42
5.3.1 insert() & size().....	42
5.3.2 clear()	43
5.3.3 empty().....	44
5.3.4 find()	44
5.3.5 count().....	45
5.3.6 erase()	45
5.3.7 lower_bound().....	46
5.3.8 upper_bound().....	47
List of all Functions of std::map	48
5.4 Duyệt map	50
5.5 Ứng dụng	53
5.6 Multimap trong C++	55
List of Functions of Multimap	57
5.7 Unordered_map trong C++	59
Methods on unordered_map	60

1. Vòng lặp for each trong C++11

- Phiên bản C++11 cung cấp một loại vòng lặp mới, có cú pháp đơn giản, dễ sử dụng hơn for loop, được gọi là **for-each loop** (Range-based for loop).

- **For-each loop** được sử dụng để lặp qua các phần tử trên 1 mảng (hoặc các cấu trúc danh sách khác như vectors, linked lists, trees, và maps).

1.1 Cú pháp vòng lặp for-each

```
for (element_declaration : array)
    statement(s);
```

Trong đó:

- **element_declaration**: là một khai báo biến (int a) hoặc tham chiếu có kiểu trùng với kiểu của các phần tử trong array. Thường sử dụng từ khóa auto.
- **array**: tên mảng hoặc cấu trúc danh sách cần lặp.
- **statement**: câu lệnh đơn hoặc khối lệnh.

Ví dụ: sử dụng for-each loop để truy cập vào từng phần tử của mảng:

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

int main()
{
    int arr[] = { 14, 3, 6, 27, 12 };
    for (int item: arr)
    {
        // biến item đại diện cho phần tử mảng ở mỗi vòng lặp
        cout << item << " ";
    }
    cout << endl;
    return 0;
}
```

Output: 14 3 6 27 12

- Trong chương trình trên, vòng lặp for-each sẽ lặp qua từng phần tử trong mảng **arr**, gán giá trị phần tử hiện tại vào biến **item**. Biến **item** nên có cùng kiểu dữ liệu với phần tử trong mảng **arr**.

Chú ý: Biến **element_declaration** đại diện cho phần tử của **array** ở vòng lặp hiện tại, **không phải** là chỉ số của **array**.

1.2 Vòng lặp for-each và từ khóa auto

- Vì biến **element_declaration** nên có cùng kiểu dữ liệu với phần tử trong mảng, nên cách tốt nhất là sử dụng từ khóa **auto** để khai báo. **Compiler** sẽ tự động xác định kiểu cho biến.

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

int main()
{
    int arr[] = { 14, 3, 6, 27, 12 };
    for (auto item: arr) // compiler tự động xác định kiểu cho item
    {
        cout << item << " ";
    }
    cout << endl;
    return 0;
}
```

Output: 14 3 6 27 12

1.3 Vòng lặp for-each và tham chiếu

Trong các ví dụ trên, vòng lặp **for-each** sử dụng các **biến giá trị** cho mỗi lần lặp:

```
int arr[] = { 14, 3, 6, 27, 12 };
for (auto item: arr) // item là 1 bản copy của phần tử hiện tại
{
    cout << item << " ";
}
```

- Với phương pháp này, các biến sẽ được sao chép giá trị của phần tử hiện tại trong mảng sau mỗi lần lặp. Điều này gây giảm hiệu suất và tốn vùng nhớ khi sao chép. Để khắc phục trường hợp này, ta sử dụng biến tham chiếu.

- Sử dụng biến tham chiếu cho vòng lặp for-each giúp CPU **truy cập trực tiếp** vào phần tử của mảng, không mất thời gian và vùng nhớ để **khởi tạo bản sao** cho biến. Tuy nhiên, tham chiếu có thể làm **thay đổi giá trị** của phần tử mảng.

```
int arr[] = { 14, 3, 6, 27, 12 };
for (auto &item: arr) // item là 1 biến tham chiếu đến phần tử hiện tại
{
    cout << item << " ";
}
```

- Trường hợp không muốn thay đổi giá trị của phần tử trong mảng, bạn có thể sử dụng **tham chiếu hằng (const reference)**.

```
int arr[] = { 14, 3, 6, 27, 12 };
for (const auto &item: arr) // item là 1 tham chiếu hằng đến phần tử hiện tại
{
    cout << item << " ";
}
```

Chú ý: Sử dụng **tham chiếu (reference)** hoặc **tham chiếu hằng (const reference)** cho biến khai báo trong vòng lặp for-each vì lý do hiệu suất.

1.4 Vòng lặp for-each không làm việc với con trỏ mảng

- Để lặp qua 1 mảng, vòng lặp **for-each** phải biết được kích thước của mảng đó. Con trỏ đến 1 mảng không cho biết kích thước của mảng đó, nên vòng lặp for-each sẽ không làm việc.

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

void printArray(int arr[])
{
    for (const auto &item : arr) // lỗi biên dịch vì arr chỉ là 1 con trỏ
    {
        cout << item << " ";
    }
}

int main()
{
    int arr[] = { 14, 3, 6, 27, 12 };
    printArray(arr);

    return 0;
}
```

Chú ý: Vòng lặp for-each **không** làm việc với con trỏ đến một mảng.

1.5 Xem chỉ số phần tử hiện tại trong vòng lặp for-each?

Vòng lặp **for-each** **không** cung cấp phương thức trực tiếp nào để xem được chỉ số phần tử hiện tại. Với những yêu cầu liên quan đến chỉ số phần tử, bạn có thể sử dụng **vòng lặp for trong C++**

2. Vector

- **Vector** là một container có tính chất tương tự như 1 mảng động, nó tự thay đổi kích thước khi bạn thêm hay xóa các phần tử trong mảng.
- Ngoài ra nó hỗ trợ truy cập các phần tử trong mảng thông qua chỉ số như mảng 1 chiều.
- Vector có thể lưu các kiểu dữ liệu như int, long long, float, char, pair, hoặc thậm chí là một vector khác. Khi sử dụng vector bạn cần thêm thư viện "vector" vào chương trình của mình.
- Có thể hiểu đơn giản nó là một mảng động, có thể tự động tăng kích thước.

2.1 Khai báo

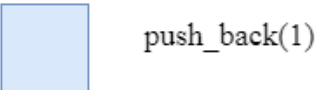
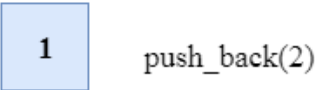
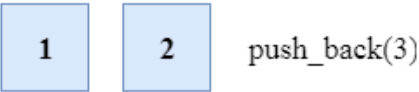
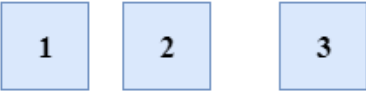
- Khai báo thư viện & namespace:

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

- Khai báo: `vector<kiểu_dữ_liệu>Tên_vector`

2.2 push_back(...)

- Đẩy một phần tử vào cuối vector hiện hành.

	<pre>#include <iostream> #include <vector> using namespace std; int main() { vector<int>v; v.push_back(1); v.push_back(2); v.push_back(3); }</pre>
	
	
	

2.3 size()

- Cho biết kích thước của vector hiện hành.

<pre>#include <iostream> #include <vector> using namespace std; int main() { vector<int>v; v.push_back(1); v.push_back(2); v.push_back(3); cout << v.size(); }</pre>	Kết quả: 3
---	-------------------

- Vector sẽ tự động cập nhật kích thước khi có sự biến động về số lượng phần tử.

<pre>#include <iostream> #include <vector> using namespace std; int main() { vector<int>v; v.push_back(1); v.push_back(2); v.push_back(3); cout << v.size() << endl; v.push_back(4); cout << v.size(); }</pre>	Kết quả: 3 4
---	---------------------------

2.4 Truy cập vào các phần tử

<pre>#include <iostream> #include <vector> using namespace std; int main() { vector<int>v; v.push_back(1); v.push_back(2); v.push_back(3); v.push_back(4); cout << "Phan tu dau tien trong vector: " << v[0] << endl; cout << "Phan tu cuoi cung trong vector: " << v[v.size() - 1] << endl; cout << "Phan tu cuoi cung trong vector: " << v.back() << endl; return 0; }</pre>	
---	--

Kết quả:

Phan tu dau tien trong vector: 1

Phan tu cuoi cung trong vector: 4

Phan tu cuoi cung trong vector: 4

- Có thể dùng iterator.

<pre>#include <iostream> #include <vector> using namespace std; int main() { vector<int>v; v.push_back(1); v.push_back(2); v.push_back(3); v.push_back(4); cout << v[2] << endl; cout << *(v.begin() + 2); return 0; }</pre>	Kết quả: 3 3
---	---------------------------

2.5 Duyệt các phần tử

<pre>#include <iostream> #include <vector> using namespace std; int main() { int i; vector<int>v; v.push_back(1); v.push_back(2); v.push_back(3); v.push_back(4); for (i = 0; i < v.size(); i++) { cout << v[i] << endl; } return 0; }</pre>	<pre>#include <iostream> #include <vector> using namespace std; int main() { vector<int>v; v.push_back(1); v.push_back(2); v.push_back(3); v.push_back(4); for (int i:v) { cout << i << endl; } return 0; }</pre>	Kết quả: 1 2 3 4
--	--	-------------------------------------

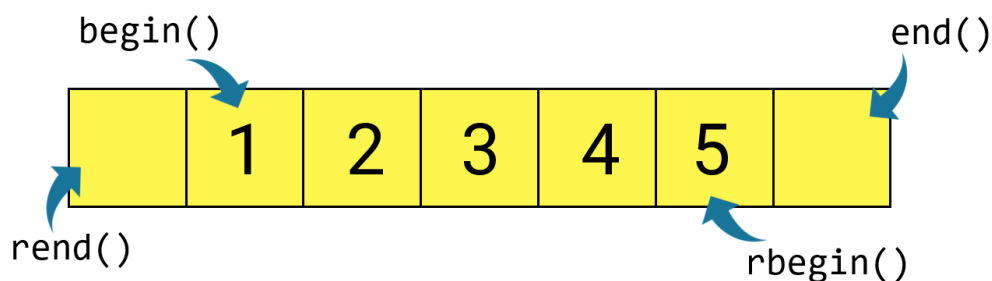
2.6 Iterator

- **Iterator** tương tự như con trỏ, nó trỏ tới các phần tử trong vector và là một kiến thức cực kỳ quan trọng. Nó giúp các bạn có thể sử dụng được các hàm, thuật toán trong thư viện STL

- Có 4 iterator chính trong vector mà bạn cần nắm được :

1. **begin()** : Iterator trỏ tới phần tử đầu tiên trong vector
2. **end()** : Iterator trỏ tới phần tử sau phần tử cuối cùng trong vector
3. **rbegin()** : Iterator ngược trỏ tới phần tử cuối cùng trong vector
4. **rend()** : Iterator ngược trỏ tới phần tử trước phần tử đầu tiên trong vector

```
vector<int> v = { 1, 2, 3, 4, 5 };
```



- **Cú pháp khai báo iterator :**

```
vector<data_type>::iterator iterator_name;
vector<data_type>reverse_iterator iterator_name;
```

- Tương tự như con trỏ thì khi bạn muốn truy cập vào phần tử mà iterator trong vector đang trỏ tới bạn cần giải tham chiếu bằng toán tử * .

Ví dụ duyệt các phần tử trong vector:

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

int main()
{
    vector<int>v;
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    v.push_back(4);
    for (vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
        cout << *it << endl;
    }
    return 0;
}
```

Kết quả:

1
2
3
4

Hoặc có thể dùng từ khóa auto:

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

int main()
{
    vector<int>v;
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    v.push_back(4);
    for (auto it = v.begin(); it != v.end(); ++it) {
        cout << *it << endl;
    }
    return 0;
}
```

- Các Toán Tử Với Iterator

Iterator hỗ trợ các toán tử ++, -- hoặc toán tử toán học để bạn có thể di chuyển iterator qua lại các phần tử trong vector.

Ví dụ 1 :

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

using namespace std;

int main(){
    vector<int> v = {100, 200, 300, 400, 500};
    vector<int>::iterator it = v.begin();
    cout << "v[0] = " << *it << endl;
    ++it; // it1 => v[1]
    cout << "v[1] = " << *it << endl;
    it += 2; // it1 => v[3]
    cout << "v[3] = " << *it << endl;
    --it; // it1 => v[2]
    cout << "v[2] = " << *it << endl;
}
```

Output :

v[0] = 100
v[1] = 200
v[3] = 400
v[2] = 300

Bạn cũng có thể tính khoảng cách giữa 2 iterator hoặc tìm ra chỉ số của phần tử mà iterator đang trỏ tới bằng cách sử dụng toán tử hoặc dùng hàm **distance()**

Ví dụ 2 :

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

using namespace std;

int main(){
    vector<int> v = {100, 200, 300, 400, 500};
    vector<int>::iterator it = v.begin();
    it += 3;
    cout << "Chỉ số của phần tử mà it đang trỏ tới : " << it - v.begin() << endl;
    cout << "Chỉ số của phần tử mà it đang trỏ tới : " << distance(v.begin(), it) << endl;
}
```

Output :

```
Chỉ số của phần tử mà it đang trỏ tới : 3
Chỉ số của phần tử mà it đang trỏ tới : 3
```

2.7 Khai báo vector với kích thước cố định

- Mặc dù vector có thể tự xác định số lượng phần tử mà không cần khai báo trước, tuy nhiên nếu muốn ta có thể tạo vector có số lượng phần tử cố định (như mảng).

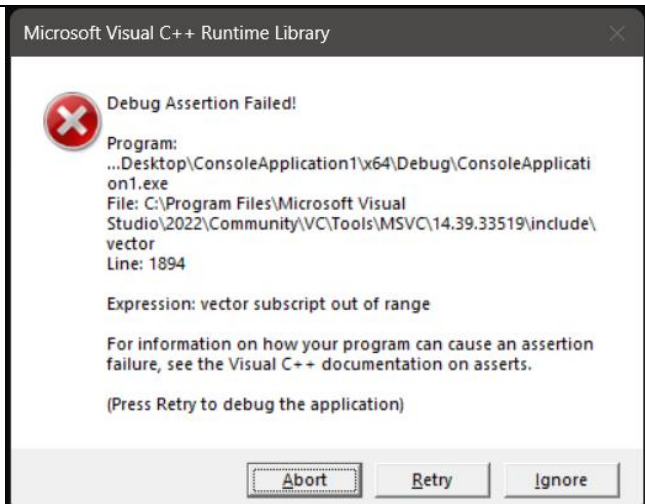
```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
    int n; //số lượng phần tử của vector
    cin >> n;
    vector<int>v(n); //Tương đương int v[n]
    for (int i = 0; i < n; i++) {
        cin >> v[i];
    }
    return 0;
}
```

- Nếu không khai báo số lượng phần tử trong vector mà dùng vòng for như trên để nhập dữ liệu của từng phần tử thì chương trình sẽ báo lỗi.

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

int main()
{
    int n; //số lượng phần tử của vector
    cin >> n;
    vector<int>v;
    for (int i = 0; i < n; i++) {
        cin >> v[i];
    }
    return 0;
}
```



- Cách giải quyết là ta sử dụng một biến tạm.

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

int main()
{
    int n; //số lượng phần tử của vector
    cin >> n;
    vector<int>v;
    for (int i = 0; i < n; i++) {
        int x; cin >> x;
        v.push_back(x);
    }
    return 0;
}
```

- Khai báo vector với số lượng phần tử cố định & các giá trị của các phần tử đều giống nhau.

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

int main()
{
    int n; //số lượng phần tử của vector
    cin >> n;
    vector<int>v(n,100); //Tương đương int
v[n]
    for (int i = 0; i < n; i++) {
        cout << v[i] << endl;
    }
    return 0;
}
```

Kết quả:

8
100
100
100
100
100
100
100
100

2.8 Lưu các kiểu dữ liệu khác

Ví dụ 1:

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

int main()
{
    vector <string> v;
    v.push_back("KTLT");
    v.push_back("DSA");
    for (int i = 0; i < v.size(); i++) {
        cout << v[i] << endl;
    }
    return 0;
}
```

Kết quả:

KTLT
DSA

Ví dụ 2: Tách từng từ trong một chuỗi

<pre>#include <iostream> #include <vector> #include <string> #include <sstream> using namespace std; int main() { string s = "ngon ngu lap trinh java"; stringstream ss(s); string tmp; vector<string>v; while (ss >> tmp) { v.push_back(tmp); } for (string x : v) { cout << x << endl; } return 0; }</pre>	Kết quả: ngon ngu lap trinh java
---	--

2.9 Vector Và Mảng 2 Chiều

Mỗi vector có thể sử dụng như mảng 1 chiều, nếu muốn sử dụng vector như mảng 2 chiều bạn cần sử dụng vector các vector.

- **Cách 1 :** Nhập từng dòng của mảng 2 chiều như 1 vector và thêm vào vector chính.

```
#include <iostream>
#include <vector>
using namespace std;
int main(){
    int n, m;
    cout << "Nhap hang, cot : ";
    cin >> n >> m;
    vector<vector<int>> v;
    for(int i = 0; i < n; i++){
        vector<int> row;
        for(int j = 0; j < m; j++){
            cout << "Nhap phan tu hang " << i + 1 << ", cot " << j + 1 << " : ";
            int tmp; cin >> tmp;
            row.push_back(tmp);
        }
        v.push_back(row);
    }
    cout << "\nMang 2 chieu vua nhap : \n";
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cout << v[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

- **Cách 2** : Khai báo sẵn vector có kích thước với hàng, cột được nhập từ bàn phím

* **Sử dụng cú pháp sau:**

```
using namespace std;
vector<vector<type>> data {v1, v2, v3, ...};
```

Trong đó:

- o data là tên biến vector 2 chiều
- o **vector<type>** là kiểu dữ liệu của vector bên ngoài (kiểu dữ liệu của vector bên ngoài lại là kiểu vector).
- o vi là các vector 1 chiều được sử dụng như phần tử của vector 2 chiều, i = 1,2,3...

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

using namespace std;

int main(){
    int n, m;
    cout << "Nhap hang, cot : ";
    cin >> n >> m;
    vector<vector<int>> v(n, vector<int>(m));
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cout << "Nhap phan tu hang " << i + 1 << ", cot " << j + 1 << " : ";
            cin >> v[i][j];
        }
    }
    cout << "\nMang 2 chieu vua nhap : \n";
    for(int i = 0; i < n; i++){
        for(int j = 0; j < m; j++){
            cout << v[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

3. Pair & Tuple

3.1. Kiểu Pair

- Kiểu dữ liệu Pair trong C++ sẽ rất hữu ích khi chúng ta cần lưu các cặp dữ liệu, chẳng hạn như lưu giá trị tọa độ Oxy, tọa độ nơi bắt đầu và kết thúc của một đường thẳng trong hệ tọa độ, thời gian bắt đầu và thời gian kết thúc của một tác vụ nào đó...

- pair nằm trong thư viện "utility" được sử dụng để kết hợp 2 kiểu dữ liệu với nhau, nó cung cấp cách lưu trữ 2 giá trị đi kèm nhau nhưng chỉ sử dụng 1 biến.

- Nhìn chung pair giúp lưu trữ dữ liệu theo cặp và có những đặc điểm sau :

- Phần tử thứ nhất của pair được gọi là **first**, phần tử thứ 2 của pair được gọi là **second**
- Bạn có thể so sánh, gán, copy pair như kiểu dữ liệu bạn thường gặp
- Để truy cập vào phần tử thứ nhất và thứ 2 trong pair ta sử dụng toán tử dấu chấm

3.1.1 Khai báo & khởi tạo

```
//Cách 1 : Giá trị của first và second là mặc định
pair<first_data_type, second_data_type> pair_name;

//Cách 2 : Giá trị của first là value1, second là value2
pair<first_data_type, second_data_type> pair_name = make_pair(value1, value2)

//Cách 3 : Giá trị của first là value1, second là value2
pair<first_data_type, second_data_type> pair_name(value1, value2)

//Cách 4 : Giá trị của first là value1, second là value2
pair<first_data_type, second_data_type> pair_name = {value1, value2}
```

Thành phần first và second của pair có thể là các kiểu dữ liệu thường gặp như int, float, double, long long, char... hay cũng có thể chính là một pair khác. Ngoài ra nó còn có thể là các kiểu phức tạp hơn như string, vector<>, set, object...

Ví dụ:

<pre>#include <iostream> #include <utility> using namespace std; int main() { pair<int, int>v; pair<string, int> v1; cout << v.first << endl; cout << v.second << endl; return 0; }</pre>	Kết quả: 0 0
<pre>#include <iostream> #include <utility> using namespace std; int main() { pair<int, int>v = make_pair(100,200); //pair<int, int>v = { 100,200 }; cout << v.first << endl; cout << v.second << endl; return 0; }</pre>	Kết quả: 100 200
<pre>#include <iostream> #include <utility> using namespace std; int main(){ pair<int, int> a = make_pair(28, 100); cout << a.first << ' ' << a.second << endl; pair<char, int> b = {'@', 28}; cout << b.first << ' ' << b.second << endl; pair<char, char> c('#', '\$'); cout << c.first << ' ' << c.second << endl; return 0; }</pre>	Output : 28 100 @ 28 # \$

- **Khai báo pair lồng nhau:** Chẳng hạn như khi cần khai báo 3 giá trị đi chung với nhau mô tả một cạnh của đồ thị có trọng số (đỉnh 1, đỉnh 2, trọng số).

```
#include <iostream>
#include <utility>

using namespace std;

int main(){
    pair<int, pair<char, int> > p1 = make_pair(28, make_pair('@', 100));
    cout << p1.first << endl;
    cout << p1.second.first << ' ' << p1.second.second << endl;

    pair<pair<int, int>, pair<int, int>> p2 = {{10, 20}, {30, 40}};
    cout << p2.first.first << ' ' << p2.first.second << endl;
    cout << p2.second.first << ' ' << p2.second.second << endl;
    return 0;
}
```

Output :

```
28
@ 100
10 20
30 40
```

* Phần đầu tiên là một pair, phần thứ hai là một giá trị kiểu int. Trong thành phần đầu tiên là một pair gồm có 2 thành phần nữa.

3.1.2 Toán Tử Với Pair

- Có thể sử dụng các toán tử so sánh, gán với kiểu pair này.

Ví dụ 1. Toán tử gán sẽ gán giá trị của first và second của 2 pair cho nhau. Trong ví dụ này mình demo với kiểu string - chuỗi ký tự trong C++

```
#include <iostream>
#include <utility>

using namespace std;

int main(){
    pair<string, int> p = make_pair("28tech", 28);
    pair<string, int> p1 = p;
    cout << p1.first << ' ' << p1.second << endl;
    return 0;
}
```

Output:

```
28tech 28
```

Ví dụ 2: So sánh 2 pair sẽ so sánh giá trị first sau đó so sánh giá trị second

```
#include <iostream>
#include <utility>

using namespace std;

int main(){
    pair<int, int> p1 = {10, 20};
    pair<int, int> p2 = {10, 21};
    pair<int, int> p3 = {5, 35};
    cout << boolalpha << (p1 == p2) << endl;
    cout << boolalpha << (p1 != p2) << endl;
    cout << boolalpha << (p1 < p2) << endl;
    cout << boolalpha << (p1 > p3) << endl;
    return 0;
}
```

Output:

```
false
true
true
true
```

3.1.3 Một số hàm được sử dụng với pair

a/ swap: This function swaps the contents of one pair object with the contents of another pair object. The pairs must be of the same type.

Ví dụ: swap 2 pair với nhau bằng hàm swap

```
#include <iostream>
#include <utility>

using namespace std;

int main(){
    pair<int, int> p1 = {10, 20};
    pair<int, int> p2 = {30, 40};
    cout << "Ban dau : \n";
    cout << "p1 = {" << p1.first << ", " << p1.second << "}\n";
    cout << "p2 = {" << p2.first << ", " << p2.second << "}\n";
    p1.swap(p2);
    cout << "Sau khi swap : \n";
    cout << "p1 = {" << p1.first << ", " << p1.second << "}\n";
    cout << "p2 = {" << p2.first << ", " << p2.second << "}\n";
    return 0;
}
```

Output:

```
Ban dau :
p1 = {10, 20}
p2 = {30, 40}
Sau khi swap :
p1 = {30, 40}
p2 = {10, 20}
```

b/ tie(): Hàm này hoạt động tương tự như trong [tuples](#). Nó tạo ra một bộ tham chiếu giá trị lvalue cho các đối số của nó, tức là để “giải nén” các giá trị bộ (hoặc cặp ở đây) thành các biến riêng biệt. Từ khóa “ignore” bỏ qua một phần tử bộ cụ thể khỏi việc “giải nén”.

Syntax:

```
tie(int &, int &) = pair1;
```

```
// CPP code to illustrate tie() in Pair
#include <bits/stdc++.h>
using namespace std;

// Driver Code
int main()
{
    pair<int, int> pair1 = { 1, 2 };
    int a, b;
    tie(a, b) = pair1;
    cout << a << " " << b << "\n";

    pair<int, int> pair2 = { 3, 4 };
    tie(a, ignore) = pair2;

    // prints old value of b
    cout << a << " " << b << "\n";

    // Illustrating pair of pairs
    pair<int, pair<int, char> > pair3 = { 3, { 4, 'a' } };
    int x, y;
    char z;

    // tie(x,y,z) = pair3; Gives compilation error
    // tie(x, tie(y,z)) = pair3; Gives compilation error
    // Each pair needs to be explicitly handled
    tie(x,ignore) = pair3;
    tie(y, z) = pair3.second;
    cout << x << " " << y << " " << z << "\n";
}

// contributed by sarthak_eddy.
```

Output

```
1 2
3 2
3 4 a
```

Code to illustrate Functions in Pair:

```
// CPP program to illustrate pair in STL
#include <iostream>
#include <string>
#include <utility>
using namespace std;

int main()
```



```

{
    pair<string, int> g1;
    pair<string, int> g2("Quiz", 3);
    pair<string, int> g3(g2);
    pair<int, int> g4(5, 10);

    g1 = make_pair(string("Geeks"), 1);
    g2.first = ".com";
    g2.second = 2;

    cout << "This is pair g" << g1.second << " with "
         << "value " << g1.first << "." << endl
         << endl;

    cout << "This is pair g" << g3.second << " with value "
         << g3.first
         << "This pair was initialized as a copy of "
         << "pair g2" << endl
         << endl;

    cout << "This is pair g" << g2.second << " with value "
         << g2.first << "\nThe values of this pair were"
         << " changed after initialization." << endl
         << endl;

    cout << "This is pair g4 with values " << g4.first
         << " and " << g4.second
         << " made for showing addition. \nThe "
         << "sum of the values in this pair is "
         << g4.first + g4.second << "." << endl
         << endl;

    cout << "We can concatenate the values of"
         << " the pairs g1, g2 and g3 : "
         << g1.first + g3.first + g2.first << endl
         << endl;

    cout << "We can also swap pairs "
         << "(but type of pairs should be same) : " << endl;
    cout << "Before swapping, "
         << "g1 has " << g1.first << " and g2 has "
         << g2.first << endl;
    swap(g1, g2);
    cout << "After swapping, "
         << "g1 has " << g1.first << " and g2 has "
         << g2.first;

    return 0;
}

```

Output

This is pair g1 with value Geeks.

This is pair g3 with value QuizThis pair was initialized as a copy of pair g2

This is pair g2 with value .com

The values of this pair were changed after initialization.

This is pair g4 with values 5 and 10 made for showing addition.

The sum of the values in this pair is 15.

We can concatenate the values of the pairs g1, g2 and g3 : GeeksQuiz.com

We can also swap pairs (but type of pairs should be same) :

Before swapping, g1 has Geeks and g2 has .com

After swapping, g1 has .com and g2 has Geeks

Time complexity: O(1).

Auxiliary space: O(1).

3.2 Kiểu tuple

- A tuple is an object that can hold a number of elements. The elements can be of different data types. The elements of tuples are initialized as arguments in order in which they will be accessed.

3.2.1 Khởi tạo & khai báo

```
#include <tuple>
tuple<kiểu_1, kiểu_2, ..., kiểu_N> Tên_tuple;
```

- Để khởi tạo giá trị, ta dùng một trong các cách sau:

+ Cách 1:

```
// Declaring tuple
tuple <char, int, float> geek;

// Assigning values to tuple using make_tuple()
geek = make_tuple('a', 10, 15.5);
```

+ Cách 2:

```
// Initializing tuple
tuple <char, int, float> geek(20, 'g', 17.5);
```

+ Cách 3:

```
tuple <int, int, int> t{ 20, 10, 12 };
```

- Có thể tạo tuple với nhiều kiểu dữ liệu khác nhau (như pair).

3.2.2 Operations on tuple

1. get<index>(tên_tuple) : get() is used to access the tuple values and modify them, it accepts the index and tuple name as arguments to access a particular tuple element.

2. make_tuple(danh_sách_các_giá_trị) : make_tuple() is used to assign tuple with values. The values passed should be in order with the values declared in tuple.

```
// C++ code to demonstrate tuple, get() and make_pair()
#include<iostream>
#include<tuple> // for tuple
using namespace std;
int main()
{
    // Declaring tuple
    tuple <char, int, float> geek;

    // Assigning values to tuple using make_tuple()
    geek = make_tuple('a', 10, 15.5);

    // Printing initial tuple values using get()
    cout << "The initial values of tuple are : ";
    cout << get<0>(geek) << " " << get<1>(geek);
    cout << " " << get<2>(geek) << endl;

    // Use of get() to change values of tuple
    get<0>(geek) = 'b';
    get<2>(geek) = 20.5;

    // Printing modified tuple values
    cout << "The modified values of tuple are : ";
    cout << get<0>(geek) << " " << get<1>(geek);
    cout << " " << get<2>(geek) << endl;

    return 0;
}
```

The initial values of tuple are : a 10 15.5

The modified values of tuple are : b 10 20.5

3. tuple_size- It returns the number of elements present in the tuple.

```
//C++ code to demonstrate tuple_size
#include<iostream>
#include<tuple> // for tuple_size and tuple
using namespace std;
int main()
{
    // Initializing tuple
    tuple <char, int, float> geek(20, 'g', 17.5);

    // Use of size to find tuple_size of tuple
    cout << "The size of tuple is : ";
    cout << tuple_size<decltype(geek)>::value << endl;

    return 0;
}
```

The size of tuple is : 3

• **decltype(geek):**

- decltype là một từ khóa trong C++ được sử dụng để truy vấn kiểu của một biểu thức. Trong trường hợp này, decltype(geek) sẽ lấy kiểu của biến geek. Vì geek là một tuple có kiểu tuple<char, int, float>, nên decltype(geek) sẽ tương đương với tuple<char, int, float>.

• **tuple_size<decltype(geek)>:**

- `tuple_size` là một template class trong thư viện `<tuple>` của C++ dùng để lấy số lượng phần tử trong một tuple. `tuple_size<decltype(geek)>` sẽ trả về một đối tượng của `tuple_size` mà kiểu của nó là `tuple<char, int, float>`.
- **`tuple_size<decltype(geek)>::value`:**
 - `::value` là một hằng số tĩnh (static constant) được định nghĩa trong `tuple_size` để giữ giá trị số lượng phần tử của tuple. Trong trường hợp này, vì `geek` là một tuple chứa 3 phần tử (`char, int, float`), nên `tuple_size<tuple<char, int, float>>::value` sẽ bằng 3.

* Từ khóa `decltype` (decltype keyword)

- Cũng tương tự với từ khóa **auto**, từ khóa **decltype** giúp chương trình tự động xác định kiểu dữ liệu cho biến. Nhưng cách sử dụng từ khóa **decltype** có một chút khác biệt so với cách sử dụng từ khóa **auto**.

- Để phân biệt:

- Từ khóa **auto** xác định kiểu dữ liệu dựa trên phân khởi tạo của biến.
- Từ khóa **decltype** xác định kiểu dữ liệu từ 1 biến hoặc 1 biểu thức khác.

Vì thế, khi sử dụng từ khóa **decltype**, chúng ta phải sử dụng kèm với 1 đối tượng cụ thể (1 biến, 1 biểu thức hoặc 1 đối tượng của class nào đó...).

- Cách sử dụng từ khóa **decltype**:

```
decltype(<object or expression>) <variable_name> [= <initial_value>];
```

- Giá trị khởi tạo (phần đặt trong ngoặc vuông) là không bắt buộc vì từ khóa **decltype** đã xác định được kiểu dữ liệu bằng cách lấy kiểu dữ liệu của đối tượng (object) hoặc biểu thức (expression).

```
int32_t i_value;
decltype(i_value) what_is_this;

cout << typeid(what_is_this).name() << endl; //int
```

- Trong đoạn chương trình trên, mình khai báo 1 biến có tên là `i_value` với kiểu dữ liệu `int32_t`. Sau đó, mình dùng từ khóa **decltype** để lấy ra kiểu dữ liệu của biến `i_value` và dùng nó cho biến mà mình muốn sử dụng.

- Thử so sánh kiểu dữ liệu của 2 biến:

```
int32_t i_value;
decltype(i_value) what_is_this;

if (typeid(i_value) == typeid(what_is_this))
    cout << "i_value and what_is_this have the same data type" << endl;
else
    cout << "Are you kidding me?" << endl;
```

Bởi vì từ khóa **decltype** lấy kiểu dữ liệu của đối tượng trước đó để khai báo cho đối tượng sau, nên hai đối tượng này luôn có cùng kiểu dữ liệu.

4. swap() :- The swap(), swaps the elements of the two different tuples.

```
//C++ code to demonstrate swap()
#include<iostream>
#include<tuple> // for swap() and tuple
using namespace std;
int main()
{

    // Initializing 1st tuple
    tuple<int, char, float> tup1(20, 'g', 17.5);

    // Initializing 2nd tuple
    tuple<int, char, float> tup2(10, 'f', 15.5);

    // Printing 1st and 2nd tuple before swapping
    cout << "The first tuple elements before swapping are : ";
    cout << get<0>(tup1) << " " << get<1>(tup1) << " "
         << get<2>(tup1) << endl;
    cout << "The second tuple elements before swapping are : ";
    cout << get<0>(tup2) << " " << get<1>(tup2) << " "
         << get<2>(tup2) << endl;

    // Swapping tup1 values with tup2
    tup1.swap(tup2);

    // Printing 1st and 2nd tuple after swapping
    cout << "The first tuple elements after swapping are : ";
    cout << get<0>(tup1) << " " << get<1>(tup1) << " "
         << get<2>(tup1) << endl;
    cout << "The second tuple elements after swapping are : ";
    cout << get<0>(tup2) << " " << get<1>(tup2) << " "
         << get<2>(tup2) << endl;

    return 0;
}
```

The first tuple elements before swapping are : 20 g 17.5
The second tuple elements before swapping are : 10 f 15.5
The first tuple elements after swapping are : 10 f 15.5
The second tuple elements after swapping are : 20 g 17.5

5. tie() : The work of tie() is to unpack the tuple values into separate variables. There are two variants of tie(), with and without “ignore”, the “ignore” ignores a particular tuple element and stops it from getting unpacked.

```
// C++ code to demonstrate working of tie()
#include<iostream>
#include<tuple> // for tie() and tuple
using namespace std;
int main()
{
    // Initializing variables for unpacking
    int i_val;
    char ch_val;
    float f_val;

    // Initializing tuple
    tuple <int, char, float> tup1(20, 'g', 17.5);

    // Use of tie() without ignore
    tie(i_val, ch_val, f_val) = tup1;

    // Displaying unpacked tuple elements without ignore
    cout << "The unpacked tuple values (without ignore) are : ";
    cout << i_val << " " << ch_val << " " << f_val;
    cout << endl;

    // Use of tie() with ignore
    // ignores char value
    tie(i_val, ignore, f_val) = tup1;

    // Displaying unpacked tuple elements with ignore
    cout << "The unpacked tuple values (with ignore) are : ";
    cout << i_val << " " << f_val;
    cout << endl;

    return 0;
}
```

The unpacked tuple values (without ignore) are : 20 g 17.5

The unpacked tuple values (with ignore) are : 20 17.5

6. tuple_cat() :- This function concatenates two tuples and returns a new tuple.

```
// C++ code to demonstrate working of tuple_cat()
#include<iostream>
#include<tuple> // for tuple_cat() and tuple
using namespace std;
int main()
{
    // Initializing 1st tuple
    tuple <int, char, float> tup1(20, 'g', 17.5);

    // Initializing 2nd tuple
    tuple <int, char, float> tup2(30, 'f', 10.5);

    // Concatenating 2 tuples to return a new tuple
    auto tup3 = tuple_cat(tup1, tup2);

    // Displaying new tuple elements
    cout << "The new tuple elements in order are : ";
    cout << get<0>(tup3) << " " << get<1>(tup3) << " ";
    cout << get<2>(tup3) << " " << get<3>(tup3) << " ";
    cout << get<4>(tup3) << " " << get<5>(tup3) << endl;

    return 0;
}
```

The new tuple elements in order are : 20 g 17.5 30 f 10.5

4. Set

Set trong C++ là một container cực kỳ mạnh mẽ và hiệu quả, đây là một kiến thức cơ bản trong C++ mà người học cần nắm vững. Khi nắm được set trong C++ rồi thì việc học set trong Java, PHP, JS hay Python cũng rất dễ dàng và nhanh chóng.

4.1 Tính chất của set trong C++

- **Set** là một container được xây dựng sẵn, bạn cũng có thể gọi nó là cấu trúc dữ liệu hay thư viện cũng được.
- The **std::set** class is the part of C++ Standard Template Library (STL) and it is defined inside the **<set>** header file.
- Đây là một cấu trúc dữ liệu đã được xây dựng sẵn, để sử dụng bạn cần thêm thư viện set vào chương trình của mình.
- Set được cài đặt bởi **cấu trúc dữ liệu cây nhị phân tìm kiếm (binary search tree)**.
- Sau đây là những tính chất quan trọng của set mà bạn cần ghi nhớ :
 1. Các phần tử trong set có giá trị khác nhau, **không có 2 phần tử có cùng giá trị**
 2. Các phần tử trong set được **tự động sắp xếp theo thứ tự tăng dần**
 3. Tìm kiếm phần tử trong set chỉ mất độ phức tạp **$O(\log N)$**
 4. **Set không thể truy cập phần tử thông qua chỉ số như mảng hay vector, string.**
- Set đặc biệt phù hợp với những bài toán liên quan tới việc **loại bỏ giá trị trùng nhau hoặc tìm kiếm nhanh**.
- Cú pháp khai báo set trong C++ :

```
#include <set>
set<data_type> set_name;
```

4.2 Một số hàm cơ bản của set

4.2.1 insert() & size()

- Để thêm 1 phần tử vào trong set bạn sử dụng hàm **insert()**, hàm này có độ phức tạp là $O(\log N)$ và lưu ý rằng nếu trong set của bạn đã tồn tại giá trị nào đó thì bạn không thể thêm nó vào nữa vì set không thể lưu giá trị trùng nhau.

- Hàm **size()** trả về số lượng phần tử trong set.

Ví dụ 1:

```
#include <iostream>
#include <algorithm>
#include <set>

using namespace std;

int main(){
    set<int> se; // {}
    se.insert(3); // {3}
    se.insert(1); // {1, 3}
    se.insert(1); // {1, 3}
    se.insert(2); // {1, 2, 3}
    se.insert(3); // {1, 2, 3}
    se.insert(2); // {1, 2, 3}
    se.insert(4); // {1, 2, 3, 4}
    cout << "So luong phan tu trong set : \n";
    cout << se.size() << endl;
}
```

Output :

So luong phan tu trong set :

4

Nhận xét : Các phần tử trong set không trùng nhau và được sắp xếp theo thứ tự tăng dần

Ví dụ 2 :

```
#include <iostream>
#include <algorithm>
#include <set>

using namespace std;

int main(){
    set<string> se; // {}
    se.insert("tech28"); // {tech28}
    se.insert("c++"); // {c++, tech28}
    se.insert("c++"); // {c++, tech28}
    se.insert("tech28"); // {c++, tech28}
    se.insert("stl"); // {c++, stl, tech28}
    se.insert("stl"); // {c++, stl, tech28}
    se.insert("stl"); // {c++, stl, tech28}
    cout << "So luong phan tu trong set : \n";
    cout << se.size() << endl;
}
```

Output :

Số lượng phần tử trong set :

3

4.2.2 Duyệt set

- Để duyệt set bạn có thể dùng range-based for loop hoặc duyệt thông qua iterator, tương tự như vector hay các container khác trong thư viện STL thì set cũng có [iterator](#)

Ví dụ 1: Duyệt set

```
#include <iostream>
#include <algorithm>
#include <set>

using namespace std;

int main(){
    set<int> se; // {}
    se.insert(3); // {3}
    se.insert(1); // {1, 3}
    se.insert(1); // {1, 3}
    se.insert(2); // {1, 2, 3}
    se.insert(3); // {1, 2, 3}
    se.insert(2); // {1, 2, 3}
    se.insert(4); // {1, 2, 3, 4}
    cout << "Duyệt set bang range-based for loop : \n";
    for(int x : se){
        cout << x << ' ';
    }
    cout << "\nDuyệt set bang iterator : \n";
    for(set<int>::iterator it = se.begin(); it != se.end(); ++it){
        cout << *it << " ";
    }
}
```

Output :

Duyệt set bang range-based for loop :

1 2 3 4

Duyệt set bang iterator :

1 2 3 4

- Truy cập vào các giá trị đặc biệt (nhỏ nhất, lớn nhất) trong set thông qua iterator :

- begin() : Iterator trở tới phần tử đầu tiên trong set
- rbegin() : Iterator ngược trở tới phần tử cuối cùng trong set

Ví dụ 2 :

```
#include <iostream>
#include <algorithm>
#include <set>

using namespace std;

int main(){
    set<int> se; // {}
    se.insert(3); // {3}
    se.insert(1); // {1, 3}
    se.insert(1); // {1, 3}
    se.insert(2); // {1, 2, 3}
    se.insert(3); // {1, 2, 3}
    se.insert(2); // {1, 2, 3}
    se.insert(4); // {1, 2, 3, 4}
    cout << "Phan tu nho nhat : " << *se.begin() << endl;
    cout << "Phan tu lon nhat : " << *se.rbegin() << endl;
}
```

Output :

Phan tu nho nhat : 1

Phan tu lon nhat : 4

- **Chú ý** : Iterator trong set chỉ hỗ trợ toán tử ++ hoặc -- chứ không hỗ trợ toán tử += và -= như iterator của vector

Ví dụ 3:

```
#include <set>
#include <iostream>
#include <string>

using namespace std;

int main() {
    int n; cin >> n;
    set<string> s;
    cin.ignore();
    for (int i = 0; i < n; i++) {
        string str;
        getline(cin, str);
        s.insert(str);
    }
    cout << s.size();
    return 0;
}
```

Kết quả:

5
python
laptrinh
java
python
laptrinh
3

4.2.3 Hàm find()

- Hàm **find()** có chức năng tìm kiếm một phần tử trong set, hàm này rất hiệu quả và dễ dùng.
- Hàm này sẽ trả về **iterator tới phần tử nếu tìm thấy**, ngược lại sẽ **trả về iterator end()** của set
- Độ phức tạp: $O(\log N)$
- Khi gặp bài toán cần tìm kiếm nhanh bạn có thể nghĩ tới set như một giải pháp tối ưu.

Mã nguồn :

```
#include <iostream>
#include <algorithm>
#include <set>

using namespace std;

int main(){
    set<int> se = {3, 1, 2, 4, 5, 6}; // {1, 2, 3, 4, 5, 6}
    if(se.find(3) != se.end()){
        cout << "3 xuất hiện trong set\n";
    }
    else{
        cout << "3 không xuất hiện trong set\n";
    }
    if(se.find(28) != se.end()){
        cout << "28 xuất hiện trong set\n";
    }
    else{
        cout << "28 không xuất hiện trong set\n";
    }
}
```

Output :

```
3 xuất hiện trong set
28 không xuất hiện trong set
```

4.2.4 Hàm count()

- Hàm **count()** trả về *số lần xuất hiện của phần tử trong set*, do đó hàm này sẽ có giá trị trả về là 0 hoặc 1 tùy theo giá trị đếm có xuất hiện trong set hay không.
- Bạn có thể sử dụng hàm count() thay cho hàm find() vì độ phức tạp của chúng là như nhau.
- Độ phức tạp : $O(\log N)$

Mã nguồn :

```
#include <iostream>
#include <algorithm>
#include <set>
using namespace std;
int main(){
    set<int> se = {3, 1, 2, 4, 5, 6}; // {1, 2, 3, 4, 5, 6}
    if(se.count(3) != 0){
        cout << "3 xuất hiện trong set\n";
    }
    else{
        cout << "3 không xuất hiện trong set\n";
    }
    if(se.count(28) != 0){
        cout << "28 xuất hiện trong set\n";
    }
    else{
        cout << "28 không xuất hiện trong set\n";
    }
}
```

Output :

```
3 xuất hiện trong set
28 không xuất hiện trong set
```

4.2.5 Hàm `erase()`

- Hàm `erase()` giúp bạn có thể xóa một giá trị ra khỏi set, ngoài ra bạn cũng có thể *xóa thông qua iterator* trỏ tới giá trị đó trong set.
- Lưu ý nếu bạn ***cố xóa 1 phần tử không nằm trong set sẽ gây lỗi***, nên trước khi xóa hãy đảm bảo giá trị bạn xóa có mặt trong set.
- Độ phức tạp : $O(\log N)$

Mã nguồn :

```
#include <iostream>
#include <algorithm>
#include <set>

using namespace std;

int main(){
    set<int> se = {3, 1, 2, 4, 5, 6}; // {1, 2, 3, 4, 5, 6}
    //xoa thong qua gia tri
    se.erase(3);
    cout << "Set sau khi xoa 3 :\n";
    for(int x : se){
        cout << x << " ";
    }
    //xoa thong qua iterator
    set<int>::iterator it = se.find(5);
    if(it != se.end()){
        se.erase(it);
    }
    cout << "\nSet sau khi xoa 5 : \n";
    for(int x : se){
        cout << x << " ";
    }
}
```

Output :

```
Set sau khi xoa 3 :
1 2 4 5 6
Set sau khi xoa 5 :
1 2 4 6
```

4.2.6 Hàm `lower_bound()`

- Hàm `lower_bound()` trả về iterator tới phần tử có *giá trị nhỏ nhất lớn hơn hoặc bằng giá trị cần tìm kiếm*.
- Trong trường hợp tất cả các phần tử trong set *không có phần tử nào lớn hơn hoặc bằng giá trị mà bạn tìm kiếm* thì hàm này **trả về iterator `end()`** của set.
- Độ phức tạp : $O(\log N)$

Mã nguồn :

```
#include <iostream>
#include <algorithm>
#include <set>

using namespace std;

int main(){
    set<int> se = {1, 2, 3, 5, 7, 9, 10};
    set<int>::iterator it1 = se.lower_bound(6);
    if(it1 == se.end()){
        cout << "Tat ca phan tu trong set < 6\n";
    }
    else{
        cout << *it1 << " la gia tri nho nhat >= 6\n";
    }
    set<int>::iterator it2 = se.lower_bound(28);
    if(it2 == se.end()){
        cout << "Tat ca phan tu trong set < 28\n";
    }
    else{
        cout << *it2 << " la gia tri nho nhat >= 28\n";
    }
}
```

Output :

```
7 la gia tri nho nhat >= 6
Tat ca phan tu trong set < 28
```

4.2.7 Hàm upper_bound()

- Tương tự như lower_bound(), hàm **upper_bound()** trả về iterator tới phần tử có giá trị nhỏ nhất **lớn hơn giá trị cần tìm kiếm**.
- Trong trường hợp tất cả các phần tử trong set không có phần tử nào lớn hơn giá trị mà bạn tìm kiếm thì hàm này trả về **iterator end()** của set.
- Độ phức tạp : $O(\log N)$

Mã nguồn :

```
#include <iostream>
#include <algorithm>
#include <set>

using namespace std;

int main(){
    set<int> se = {1, 2, 3, 5, 7, 9, 10};
    set<int>::iterator it1 = se.upper_bound(5);
    if(it1 == se.end()){
        cout << "Tat ca phan tu trong set <= 5\n";
    }
    else{
        cout << *it1 << " la gia tri nho nhat > 5\n";
    }
    set<int>::iterator it2 = se.upper_bound(10);
    if(it2 == se.end()){
        cout << "Tat ca phan tu trong set <= 10\n";
    }
    else{
        cout << *it2 << " la gia tri nho nhat > 10\n";
    }
}
```

Output :

```
7 la gia tri nho nhat > 5
Tat ca phan tu trong set <= 10
```

Function of Set in C++ STL

Function	Description
<u>begin()</u>	Returns an iterator to the first element in the set.
<u>end()</u>	Returns an iterator to the theoretical element that follows the last element in the set.
<u>rbegin()</u>	Returns a reverse iterator pointing to the last element in the container.
<u>rend()</u>	Returns a reverse iterator pointing to the theoretical element right before the first element in the set container.
<u>crbegin()</u>	Returns a constant iterator pointing to the last element in the container.
<u>crend()</u>	Returns a constant iterator pointing to the position just before the first element in the container.

Function	Description
<u>cbegin()</u>	Returns a constant iterator pointing to the first element in the container.
<u>cend()</u>	Returns a constant iterator pointing to the position past the last element in the container.
<u>size()</u>	Returns the number of elements in the set.
<u>max_size()</u>	Returns the maximum number of elements that the set can hold.
<u>empty()</u>	Returns whether the set is empty.
<u>insert(const g)</u>	Adds a new element 'g' to the set.
<u>iterator insert (iterator position, const g)</u>	Adds a new element 'g' at the position pointed by the iterator.
<u>erase(iterator position)</u>	Removes the element at the position pointed by the iterator.
<u>erase(const g)</u>	Removes the value 'g' from the set.
<u>clear()</u>	Removes all the elements from the set.
<u>key_comp()</u> / <u>value_comp()</u>	Returns the object that determines how the elements in the set are ordered ('<' by default).
<u>find(const g)</u>	Returns an iterator to the element 'g' in the set if found, else returns the iterator to the end.
<u>count(const g)</u>	Returns 1 or 0 based on whether the element 'g' is present in the set or not.
<u>lower_bound(const g)</u>	Returns an iterator to the first element that is equivalent to 'g' or definitely will not go before the element 'g' in the set.
<u>upper_bound(const g)</u>	Returns an iterator to the first element that will go after the element 'g' in the set.
<u>equal_range()</u>	The function returns an iterator of pairs. (key_comp). The pair refers to the range that includes all the elements in the container which have a key equivalent to k.

Function	Description
<u>emplace()</u>	This function is used to insert a new element into the set container, only if the element to be inserted is unique and does not already exist in the set.
<u>emplace_hint()</u>	Returns an iterator pointing to the position where the insertion is done. If the element passed in the parameter already exists, then it returns an iterator pointing to the position where the existing element is.
<u>swap()</u>	This function is used to exchange the contents of two sets but the sets must be of the same type, although sizes may differ.
<u>operator=</u>	The '=' is an operator in C++ STL that copies (or moves) a set to another set and set::operator= is the corresponding operator function.
<u>get_allocator()</u>	Returns the copy of the allocator object associated with the set.

4.3 Multiset trong C++

- **Multiset** trong C++ cũng được cài đặt bằng cấu trúc dữ liệu cây nhị phân tìm kiếm (binary search tree) tương tự như set, đặc điểm khác biệt nhất so với set đó là nó có thể lưu trữ các giá trị trùng nhau.

- Các tính chất của multiset :

- Các phần tử trong multiset có thứ tự mặc định **theo thứ tự tăng dần**
- Các phần tử trong multiset **có thể trùng nhau**
- Các phần tử trong multiset **không thể thay đổi, chỉ có thể xóa hoặc thêm vào**
- Multiset **không hỗ trợ truy cập phần tử thông qua chỉ số**

Ví dụ :

```
#include <iostream>
#include <algorithm>
#include <set>
using namespace std;
int main(){
    multiset<int> ms;
    ms.insert(3);
    ms.insert(1);
    ms.insert(3);
    ms.insert(2);
    ms.insert(2);
    ms.insert(3);
    ms.insert(4);
    cout << "Số phần tử trong multiset : " << ms.size() << endl;
    cout << "Các phần tử trong multiset :\n";
    for(int x : ms){
        cout << x << ' ';
    }
}
```


Output :

So phan tu trong multiset : 7

Cac phan tu trong multiset :

1 2 2 3 3 3 4

- Các hàm của multiset tương tự như set tuy nhiên có sự khác nhau về 3 hàm là **find()**, **count()** và **erase()**.

4.3.1 Hàm find()

- Hàm này sẽ trả về *iterator tới vị trí đầu tiên của phần tử* trong multiset nếu giá trị tìm kiếm xuất hiện, ngược lại sẽ trả về iterator end()

- Ví dụ trong mã nguồn dưới đây bạn sẽ thấy hàm find() trả về vị trí đầu tiên của số 3 trong multiset.

Ví dụ 1 :

```
#include <iostream>
#include <algorithm>
#include <set>

using namespace std;

int main(){
    multiset<int> ms = {1, 1, 1, 2, 2, 3, 3, 3, 3, 3, 4};
    multiset<int>::iterator it = ms.find(3);
    if(it != ms.end()){
        cout << "3 xuất hiện trong multiset\n";
        cout << "Vị trí đầu tiên của 3 trong multiset : " <<
distance(ms.begin(), it) << endl;
    }
    else{
        cout << "3 không xuất hiện trong multiset\n";
    }
}
```

Output :

3 xuất hiện trong multiset

Vị trí đầu tiên của 3 trong multiset : 5

4.3.2 Hàm count()

- Hàm count() trong set chỉ trả về giá trị là 0 hoặc 1 nhưng do multiset có thể lưu giá trị trùng nhau nên hàm count() trong multiset có thể *trả về 0 hoặc một số khác 0*, tương ứng số lần xuất hiện của giá trị đó trong multiset.

Ví dụ 2:

```
#include <iostream>
#include <algorithm>
#include <set>

using namespace std;

int main(){
    multiset<int> ms = {1, 1, 1, 2, 2, 3, 3, 3, 3, 3, 4};
    cout << "Số lần xuất hiện của 3 : " << ms.count(3) << endl;
    cout << "Số lần xuất hiện của 28 : " << ms.count(28) << endl;
    cout << "Số lần xuất hiện của 1 : " << ms.count(1) << endl;
}
```

Output :

```
So lan xuất hiện của 3 : 5
So lan xuất hiện của 28 : 0
So lan xuất hiện của 1 : 3
```

4.3.3 Hàm erase()

- Khi sử dụng hàm erase() bạn cần lưu ý nếu bạn *xóa thông qua giá trị thì nó sẽ xóa hết mọi phần tử có cùng giá trị đó*, ngược lại nếu bạn chỉ muốn *xóa 1 phần tử có giá trị đó trong multiset thì bạn cần xóa thông qua iterator*.

- Tương tự như trong set bạn cần đảm bảo rằng *giá trị bạn xóa tồn tại trong multiset*.

Ví dụ 3 : Xóa phần tử thông qua giá trị

```
#include <iostream>
#include <algorithm>
#include <set>

using namespace std;

int main(){
    multiset<int> ms = {1, 1, 1, 2, 2, 3, 3, 3, 3, 3, 4};
    cout << "Multiset ban dau : \n";
    for(int x : ms){
        cout << x << " ";
    }
    ms.erase(3);
    cout << "\nMultiset sau khi xoa 3 : \n";
    for(int x : ms){
        cout << x << " ";
    }
}
```

Output :

```
Multiset ban dau :
1 1 1 2 2 3 3 3 3 3 4
Multiset sau khi xoa 3 :
1 1 1 2 2 4
```

Ví dụ 4 : Xóa thông qua iterator

```
#include <iostream>
#include <algorithm>
#include <set>
using namespace std;
int main(){
    multiset<int> ms = {1, 1, 1, 2, 2, 3, 3, 3, 3, 3, 4};
    cout << "Multiset ban dau : \n";
    for(int x : ms)
        cout << x << " ";
    multiset<int>::iterator it = ms.find(3);
    ms.erase(it);
    cout << "\nMultiset sau khi xoa 3 : \n";
    for(int x : ms)
        cout << x << " ";
}
```

Output :

Multiset ban dau :

1 1 1 2 2 3 3 3 3 3 4

Multiset sau khi xoa 3 :

1 1 1 2 2 3 3 3 3 4

4.3.4 Bài toán Sliding Window Maximum

You are given an array of integers `nums`, there is a *sliding window of size k* which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

Return *the max sliding window*.

Example 1:

Input: `nums = [1,3,-1,-3,5,3,6,7]`, `k = 3`

Output: `[3,3,5,5,6,7]`

Explanation:

Window position	Max
-----	-----
[1 3 -1] -3 5 3 6 7	3
1 [3 -1 -3] 5 3 6 7	3
1 3 [-1 -3 5] 3 6 7	5
1 3 -1 [-3 5 3] 6 7	5
1 3 -1 -3 [5 3 6] 7	6
1 3 -1 -3 5 [3 6 7]	7

Example 2:

Input: `nums = [1]`, `k = 1`

Output: `[1]`

```
#include <set>
#include <iostream>
#include <string>
using namespace std;
int main() {
    int n, k;
    cin >> n >> k; // n = 8, k = 3
    int* a = new int[n]; //mảng chứa đề bài

    for (int i = 0; i < n; i++){
        cin >> a[i];
    }

    multiset<int> ms;

    for (int i = 0; i < k; i++) {
        ms.insert(a[i]);
    }

    for (int i = k; i < n; i++) {
        cout << *ms.rbegin() << endl;
        //do multiset đã sắp xếp thứ tự 3 số tăng dần, nên max = số cuối cùng trong ms.
        ms.erase(ms.find(a[i-k]));
        ms.insert(a[i]);
    }
    cout << *ms.rbegin() << endl; //in phần tử cuối cùng của ms
    delete []a;
    return 0;
}
```

Kết quả:

8 3

1 3 -1 -3 5 3 6 7

3

3

5

5

6

7

List of Functions of Multiset

Function	Definition
<u>begin()</u>	Returns an iterator to the first element in the multiset.
<u>end()</u>	Returns an iterator to the theoretical element that follows the last element in the multiset.
<u>size()</u>	Returns the number of elements in the multiset.
<u>max_size()</u>	Returns the maximum number of elements that the multiset can hold.
<u>empty()</u>	Returns whether the multiset is empty.
<u>pair insert(const g)</u>	Adds a new element 'g' to the multiset.
<u>iterator insert (iterator position, const g)</u>	Adds a new element 'g' at the position pointed by the iterator.
<u>erase(iterator position)</u>	Removes the element at the position pointed by the iterator.
<u>erase(const g)</u>	Removes the value 'g' from the multiset.
<u>clear()</u>	Removes all the elements from the multiset.
<u>key_comp()</u> / <u>value_comp()</u>	Returns the object that determines how the elements in the multiset are ordered ('<' by default).
<u>find(const g)</u>	Returns an iterator to the element 'g' in the multiset if found, else returns the iterator to end.
<u>count(const g)</u>	Returns the number of matches to element 'g' in the multiset.

Function	Definition
<u>lower_bound(const g)</u>	Returns an iterator to the first element that is equivalent to 'g' or definitely will not go before the element 'g' in the multiset if found, else returns the iterator to end.
<u>upper_bound(const g)</u>	Returns an iterator to the first element that will go after the element 'g' in the multiset.
<u>multiset::swap()</u>	This function is used to exchange the contents of two multisets but the sets must be of the same type, although sizes may differ.
<u>multiset::operator=</u>	This operator is used to assign new contents to the container by replacing the existing contents.
<u>multiset::emplace()</u>	This function is used to insert a new element into the multiset container.
<u>multiset::equal_range()</u>	Returns an iterator of pairs. The pair refers to the range that includes all the elements in the container which have a key equivalent to k.
<u>multiset::emplace_hint()</u>	Inserts a new element in the multiset.
<u>multiset::rbegin()</u>	Returns a reverse iterator pointing to the last element in the multiset container.
<u>multiset::rend()</u>	Returns a reverse iterator pointing to the theoretical element right before the first element in the multiset container.
<u>multiset::cbegin()</u>	Returns a constant iterator pointing to the first element in the container.
<u>multiset::cend()</u>	Returns a constant iterator pointing to the position past the last element in the container.
<u>multiset::crbegin()</u>	Returns a constant reverse iterator pointing to the last element in the container.
<u>multiset::crend()</u>	Returns a constant reverse iterator pointing to the position just before the first element in the container.
<u>multiset::get_allocator()</u>	Returns a copy of the allocator object associated with the multiset.

4.4 Unordered_set trong C++

- **Unordered_set** là một container trong thư viện STL và có thể sử dụng trong các chuẩn C++ 11 tới các phiên bản mới hơn.

- **Thư viện chứa unordered_set là "unordered_set"**, bạn cần thêm vào chương trình để có thể sử dụng.

- Unordered_set được cài đặt bởi cấu trúc dữ liệu **bảng băm** - hash table vì thế rất tối ưu trong việc tìm kiếm giá trị của phần tử.

- Sự khác biệt lớn nhất giữa Unordered_set và set chính là thứ tự các phần tử trong container.

Các tính chất của unordered_set :

- Các phần tử trong unordered_set đôi một **khác nhau**
- unordered_set không **duy trì bất kỳ thứ tự nào giữa các phần tử** mà nó chứa

Ngoài sự khác biệt về thứ tự, unordered_set khác với set và multiset ở độ phức tạp của các hàm tìm kiếm, chèn, xóa..

- Các hàm find(), erase(), insert() của set và multiset có độ phức tạp là $O(\log N)$
- Các hàm find(), erase(), insert() của unordered_set có độ phức tạp trong *trường hợp trung bình* là $O(1)$ và *trong trường hợp tệ nhất* là $O(N)$

Mã nguồn :

```
#include <iostream>
#include <unordered_set>

using namespace std;

int main(){
    unordered_set<int> se = {1, 3, 1, 1, 2, 3, 1, 4, 1};
    cout << "So phan tu trong set : " << se.size() << endl;
    cout << "Cac phan tu trong set :\n";
    for(int x : se){
        cout << x << " ";
    }
    cout << endl;
    if(se.find(3) == se.end()){
        cout << "NOT FOUND\n";
    }
    else{
        cout << "FOUND\n";
    }
}
```

Output :

So phan tu trong set : 4

Cac phan tu trong set :

4 2 3 1

FOUND

std::unordered_set Member Methods

The following table contains all the member functions of std::unordered_set class:

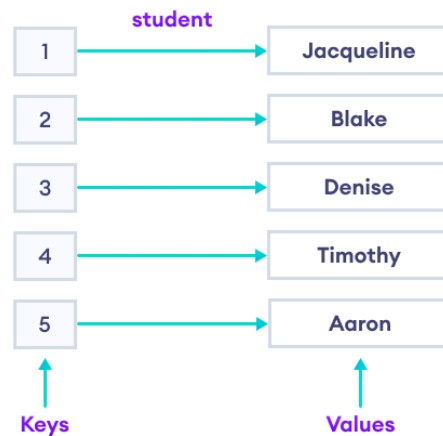
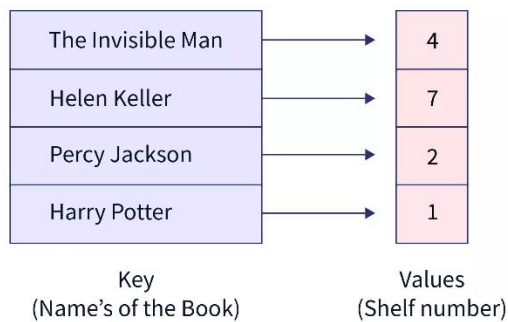
Function Name	Function Description
<u>insert()</u>	Insert a new {element} in the unordered_set container.
<u>begin()</u>	Return an iterator pointing to the first element in the unordered_set container.
<u>end()</u>	Returns an iterator pointing to the past-the-end-element.
<u>count()</u>	Count occurrences of a particular element in an unordered_set container.
<u>find()</u>	Search for an element in the container.
<u>clear()</u>	Removes all of the elements from an unordered_set and empties it.
<u>cbegin()</u>	Return a const_iterator pointing to the first element in the unordered_set container.
<u>cend()</u>	Return a const_iterator pointing to a past-the-end element in the unordered_set container or in one of its buckets.
<u>bucket_size()</u>	Returns the total number of elements present in a specific bucket in an unordered_set container.
<u>erase()</u>	Remove either a single element or a range of elements ranging from start(inclusive) to end(exclusive).
<u>size()</u>	Return the number of elements in the unordered_set container.
<u>swap()</u>	Exchange values of two unordered_set containers.
<u>emplace()</u>	Insert an element in an unordered_set container.
<u>max_size()</u>	Returns maximum number of elements that an unordered_set container can hold.
<u>empty()</u>	Check if an unordered_set container is empty or not.
<u>equal_range</u>	Returns range that includes all elements equal to a given value.

Function Name	Function Description
<u>operator=</u>	Copies (or moves) an unordered_set to another unordered_set and unordered_set::operator= is the corresponding operator function.
<u>hash_function()</u>	This hash function is a unary function that takes a single argument only and returns a unique value of type size_t based on it.
<u>reserve()</u>	Used to request a capacity change of unordered_set.
<u>bucket()</u>	Returns the bucket number of a specific element.
<u>bucket_count()</u>	Returns the total number of buckets present in an unordered_set container.
<u>load_factor()</u>	Returns the current load factor in the unordered_set container.
<u>rehash()</u>	Set the number of buckets in the container of unordered_set to a given size or more.
<u>max_load_factor()</u>	Returns(Or sets) the current maximum load factor of the unordered set container.
<u>emplace_hint()</u>	Inserts a new element in the unordered_set only if the value to be inserted is unique, with a given hint.
<u>== operator</u>	The '==' is an operator in C++ STL that performs an equality comparison operation between two unordered sets and unordered_set::operator== is the corresponding operator function for the same.
<u>key_eq()</u>	Returns a boolean value according to the comparison. It returns the key equivalence comparison predicate used by the unordered_set.
<u>operator!=</u>	The != is a relational operator in C++ STL which compares the equality and inequality between unordered_set containers.
<u>max_bucket_count()</u>	Find the maximum number of buckets that unordered_set can have.

5. Map

5.1 Tính chất của map

- **Map** là một cấu trúc dữ liệu được xây dựng sẵn trong C++, nó lưu trữ dữ liệu theo *cặp khóa (key)* và *giá trị (value)*. Mỗi phần tử trong map là *một ánh xạ từ key sang value*. Ví dụ đơn giản bạn có thể hình dung **map có thể lưu trữ các sinh viên và mã sinh viên tương ứng của họ** và giúp bạn truy vấn nhanh tên sinh viên thông qua mã sinh viên.



- Để sử dụng được map bạn **cần thêm thư viện "map"** vào chương trình của mình.

- Các tính chất quan trọng của map :

1. Các key trong map là riêng biệt, **không có 2 key trùng nhau**, nhưng giá trị thì có thể trùng nhau.
2. Map duy trì **thứ tự** các phần tử theo **giá trị key tăng dần**
3. Map tìm kiếm giá trị key với độ phức tạp $O(\log N)$
4. Map **không hỗ trợ truy cập thông qua chỉ số** như mảng hay vector, string.
5. **Mỗi phần tử trong map chính là 1 pair**

- Set rất tối ưu trong các bài toán liên quan tới giá trị khác nhau trong mảng, tìm kiếm nhanh, các bài toán liên quan tới tần suất.

- Cú pháp khai báo :

```
#include <map>
map<key_data_type, value_data_type> map_name;
```

- Map có ba phương pháp chính:

- + Thêm một cặp khóa-giá trị được chỉ định.
- + Truy xuất giá trị của một khóa nhất định.
- + Xóa cặp khóa-giá trị khỏi map.

5.2 Khởi tạo

- **$m[key]$** , trả về tham chiếu đến giá trị được liên kết với khóa *key* (truy xuất giá trị của value thông qua key).

- Nếu khóa không có trong map thì giá trị được liên kết với khóa sẽ được tạo bằng cách sử dụng hàm tạo mặc định của kiểu dữ liệu.

Ví dụ: nếu kiểu dữ liệu là int thì việc gọi $m[key]$ cho một khóa không nằm trong map sẽ đặt giá trị được liên kết với khóa đó thành 0.

Một ví dụ khác, nếu kiểu dữ liệu là `std::string` thì gọi $m[key]$ cho một khóa không có trong map sẽ đặt giá trị được liên kết với khóa đó thành chuỗi rỗng.

- Ngoài ra, **$m.at(key)$ hoạt động giống như $m[key]$** .

- **$m[key] = value$** sẽ gán giá trị cho khóa *key*. Nếu *key* đã xuất hiện trong map thì câu lệnh này sẽ thay đổi giá trị value của *key* đó.

Ví dụ:

```
#include <map>
#include <iostream>
using namespace std;
int main() {
    map<int, int> m;
    m[1] = 5;           // [(1, 5)]
    m[3] = 14;          // [(1, 5); (3, 14)]
    m[2] = 7;           // [(1, 5); (2, 7); (3, 14)]
    m[0] = -1;          // [(0, -1); (1, 5); (2, 7); (3, 14)]
    return 0;
}
```

5.3 Một số hàm cơ bản của map

5.3.1 insert() & size()

- insert(): Thêm phần tử vào trong map (cách khác: $m[key] = value$)

Lưu ý là khi thêm 1 phần tử vào trong map thì bạn cần thêm 1 cặp (pair) key - value vào map. Và nếu bạn thêm các cặp mà key đã tồn tại trong map thì map sẽ không thêm vào nữa để đảm bảo tính chất.

- size(): Trả về số phần tử trong map

Ví dụ:

```
#include <iostream>
#include <map>

using namespace std;

int main() {
    map<int, int> mp;
    mp[100] = 200;
    mp[200] = 300;
    mp.insert({ 300, 400 });
    mp.insert({ 500, 600 });
    cout << mp.size() << endl;

    mp[100] = 300; //(100, 200) thay thế bằng (100, 300)
    cout << mp.size() << endl; //số lượng phần tử vẫn là 4 do key = 100 đã tồn tại

    return 0;
}
```

Kết quả:

4
4

```
#include <iostream>
#include <map>

using namespace std;

int main() {
    map<int, int> mp;
    mp.insert(make_pair(1, 2)); // ok
    mp.insert(make_pair(1, 5)); // not ok
    mp.insert(make_pair(2, 4)); // ok
    mp.insert(make_pair(2, 5)); // not ok
    mp.insert(make_pair(2, 1)); // not ok
    mp.insert(make_pair(3, 1)); // ok
    cout << mp.size() << endl; // 3

    mp[5] = 10; // them moi
    mp[1] = 100; // thay doi value của key 1 thành 100
    cout << mp.size() << endl;
    cout << "Value tương ứng của key 1 : " << mp[1] << endl; // 100
    cout << "Value tương ứng của key 5 : " << mp[5] << endl; // 10
}
```

Output :

3
4
Value tương ứng của key 1 : 100
Value tương ứng của key 5 : 10

5.3.2 clear()

clear() function is used to remove all the elements from the map container and thus leaving it's size 0.

```
// CPP program to illustrate
// Implementation of clear() function
#include <iostream>
#include <map>
using namespace std;

int main()
{
    // Take any two maps
    map<int, string> map1, map2;

    // Inserting values
    map1[1] = "India";
    map1[2] = "Nepal";
    map1[3] = "Sri Lanka";
    map1[4] = "Myanmar";

    // Print the size of map
    cout << "Map size before running function: \n";
    cout << "map1 size = " << map1.size() << endl;
    cout << "map2 size = " << map2.size() << endl;;

    // Deleting the map elements
    map1.clear();
    map2.clear();

    // Print the size of map
    cout << "Map size after running function: \n";
    cout << "map1 size = " << map1.size() << endl;
    cout << "map2 size = " << map2.size();
    return 0;
}
```

Map size before running function:

map1 size = 4

map2 size = 0

Map size after running function:

map1 size = 0

map2 size = 0

Time complexity: Linear i.e. $O(n)$

5.3.3 empty()

Kiểm tra vector có rỗng hay không.

```
// Empty map example
// CPP program to illustrate
// Implementation of empty() function
#include <iostream>
#include <map>
using namespace std;

int main()
{
    map<char, int> mymap;
    if (mymap.empty()) {
        cout << "True";
    }
    else {
        cout << "False";
    }
    return 0;
}
```

False

5.3.4 find()

Được sử dụng để *tìm giá trị key trong map*, độ phức tạp của hàm này là $O(\log N)$.

Giá trị trả về của hàm **find()** là **iterator**, nếu giá trị mà bạn tìm kiếm xuất hiện trong danh sách tập key của map thì hàm này **sẽ trả về iterator tới cặp phần tử có key tương ứng**, trường hợp key bạn tìm kiếm không xuất hiện trong map thì *hàm trả về iterator end() của map*.

Hàm này có độ phức tạp rất tốt nên bạn có thể sử dụng map trong các bài toán tìm kiếm nhanh.

Mã nguồn :

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

int main(){
    map<int, int> mp;
    mp.insert(make_pair(1, 2));
    mp.insert(make_pair(2, 3));
    mp.insert(make_pair(3, 5));
    map<int, int>::iterator it1 = mp.find(2);
    map<int, int>::iterator it2 = mp.find(5);
    if(it1 != mp.end()){
        cout << "FOUND\n";
    }
    else cout << "NOT FOUND\n";
    if(it2 != mp.end()){
        cout << "FOUND\n";
    }
    else cout << "NOT FOUND\n";
    return 0;
}
```

Output :

FOUND
NOT FOUND

5.3.5 count()

Hàm trả về số lần khóa K xuất hiện trong vùng chứa của map. Và vì mỗi key trong map chỉ xuất hiện 1 lần nên hàm này trả về 1 nếu key bạn tìm kiếm xuất hiện trong map, ngược lại trả về 0.

Độ phức tạp là $O(\log N)$ và dễ dùng hơn hàm find() nên bạn có thể sử dụng hàm này để thay thế cho hàm find() trong việc tìm kiếm.

Mã nguồn :

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

int main(){
    map<int, int> mp;
    mp.insert(make_pair(1, 2));
    mp.insert(make_pair(2, 3));
    mp.insert(make_pair(3, 5));
    if(mp.count(2) != 0){
        cout << "FOUND\n";
    }
    else cout << "NOT FOUND\n";
    if(mp.count(4) == 1){
        cout << "FOUND\n";
    }
    else cout << "NOT FOUND\n";
    return 0;
}
```

Output :

```
FOUND
NOT FOUND
```

5.3.6 erase()

Xóa 1 cặp phần tử trong map thông qua key, có 2 cách sử dụng hàm này là xóa thông qua giá trị hoặc xóa thông qua iterator.

Độ phức tạp của hàm này là $O(\log N)$ nhưng khi sử dụng bạn cần hết sức lưu ý nếu bạn *xóa 1 key không xuất hiện trong map sẽ gây lỗi*.

Mã nguồn 1 : Xóa thông qua giá trị

```
#include <iostream>
#include <map>
using namespace std;
int main(){
    map<int, int> mp;
    mp.insert(make_pair(1, 2));
    mp.insert(make_pair(2, 3));
    mp.insert(make_pair(3, 5));
    if(mp.count(2))
        mp.erase(2);
    cout << "Map sau khi xoa key 2 : \n";
    for(pair<int, int> it : mp)
        cout << it.first << " " << it.second << endl;
    return 0;
}
```

Output :

Map sau khi xoa key 2 :

1 2

3 5

Mã nguồn 2 : Xóa thông qua iterator

```
#include <iostream>
#include <map>

using namespace std;

int main(){
    map<int, int> mp;
    mp.insert(make_pair(1, 2));
    mp.insert(make_pair(2, 3));
    mp.insert(make_pair(3, 5));
    map<int, int>::iterator it = mp.find(2);
    if(it != mp.end()){
        mp.erase(it);
    }
    cout << "Map sau khi xoa key 2 : \n";
    for(pair<int, int> it : mp){
        cout << it.first << " " << it.second << endl;
    }
    return 0;
}
```

Output :

Map sau khi xoa key 2 :

1 2

3 5

5.3.7 lower_bound()

Hàm **lower_bound()** ngoài sử dụng với mảng hay vector đã được sắp xếp thì còn có thể áp dụng với set & map. Hàm này *trả về iterator tới giá trị nhỏ nhất trong map có key lớn hơn hoặc bằng giá trị tìm kiếm.*

Ví dụ map của bạn có tập key là {1, 3, 6, 8, 9, 12} và bạn tìm kiếm giá trị $X = 7$ thì hàm này sẽ trả về iterator tới phần tử có key là 8 trong map, tương tự nếu bạn tìm $X = 3$ thì sẽ trả về iterator tới phần tử có key là 3.

Nếu trong trường hợp map của bạn không có key nào lớn hơn hoặc bằng giá trị key tìm kiếm thì hàm *trả về iterator end() của map.*

Độ phức tạp là **$O(\log N)$**

Mã nguồn :

```
#include <iostream>
#include <map>

using namespace std;

int main(){
    map<int, int> mp;
    mp.insert(make_pair(1, 2));
    mp.insert(make_pair(3, 3));
    mp.insert(make_pair(5, 5));
    mp.insert(make_pair(6, 8));
    mp.insert(make_pair(10, 12));
    map<int, int>::iterator it1 = mp.lower_bound(4);
    if(it1 == mp.end()){
        cout << "Khong tim thay key >= 4" << endl;
    }
    else{
        cout << "Key nho nhat >= 4 la : " << (*it1).first << endl;
    }
    map<int, int>::iterator it2 = mp.lower_bound(14);
    if(it2 == mp.end()){
        cout << "Khong tim thay key >= 14" << endl;
    }
    else{
        cout << "Key nho nhat >= 14 la : " << (*it2).first << endl;
    }
    return 0;
}
```

Output :

```
Key nho nhat >= 4 la : 5
Khong tim thay key >= 14
```

5.3.8 upper_bound()

Tương tự như hàm **lower_bound()** thì hàm **upper_bound()** được dùng để *tìm giá trị nhỏ nhất lớn hơn giá trị key mà bạn tìm kiếm.*

Ví dụ map của bạn có tập key là {1, 3, 6, 8, 9, 12} và bạn tìm kiếm giá trị X = 7 thì hàm này sẽ trả về iterator tới phần tử có key là 8 trong map, tương tự nếu bạn tìm X = 3 thì sẽ trả về iterator tới phần tử có key là 6.

Nếu trong trường hợp map của bạn không có key nào lớn hơn giá trị key tìm kiếm thì hàm trả về iterator end() của map.

Độ phức tạp là **O(logN)**

Mã nguồn :

```
#include <iostream>
#include <map>

using namespace std;

int main(){
    map<int, int> mp;
    mp.insert(make_pair(1, 2));
    mp.insert(make_pair(3, 3));
    mp.insert(make_pair(5, 5));
    mp.insert(make_pair(6, 8));
    mp.insert(make_pair(10, 12));
    map<int, int>::iterator it1 = mp.upper_bound(5);
    if(it1 == mp.end()){
        cout << "Khong tim thay key > 5" << endl;
    }
    else{
        cout << "Key nho nhat >= 5 la : " << (*it1).first << endl;
    }
    map<int, int>::iterator it2 = mp.upper_bound(14);
    if(it2 == mp.end()){
        cout << "Khong tim thay key > 14" << endl;
    }
    else{
        cout << "Key nho nhat >= 14 la : " << (*it2).first << endl;
    }
    return 0;
}
```

Output :

Key nho nhat >= 5 la : 6
Khong tim thay key > 14

List of all Functions of std::map

The following table contains all the functions defined inside std::map class.

Function	Definition
map::insert()	Insert elements with a particular key in the map container → O(log n)
map::count()	Returns the number of matches to element with key-value 'g' in the map. → O(log n)
map::equal_range()	Returns an iterator of pairs. The pair refers to the bounds of a range that includes all the elements in the container which have a key equivalent to k.
map::erase()	Used to erase elements from the container → O(log n)

Function	Definition
<u>map rend()</u>	Returns a reverse iterator pointing to the theoretical element right before the first key-value pair in the map(which is considered its reverse end).
<u>map rbegin()</u>	Returns a reverse iterator which points to the last element of the map.
<u>map find()</u>	Returns an iterator to the element with key-value 'g' in the map if found, else returns the iterator to end.
<u>map crbegin()</u> and <u>crend()</u>	crbegin() returns a constant reverse iterator referring to the last element in the map container. crend() returns a constant reverse iterator pointing to the theoretical element before the first element in the map.
<u>map cbegin()</u> and <u>cend()</u>	cbegin() returns a constant iterator referring to the first element in the map container. cend() returns a constant iterator pointing to the theoretical element that follows the last element in the multimap.
<u>map emplace()</u>	Inserts the key and its element in the map container.
<u>map max_size()</u>	Returns the maximum number of elements a map container can hold $\rightarrow O(1)$
<u>map upper_bound()</u>	Returns an iterator to the first element that is equivalent to mapped value with key-value 'g' or definitely will go after the element with key-value 'g' in the map
<u>map operator=</u>	Assigns contents of a container to a different container, replacing its current content.
<u>map lower_bound()</u>	Returns an iterator to the first element that is equivalent to the mapped value with key-value 'g' or definitely will not go before the element with key-value 'g' in the map $\rightarrow O(\log n)$
<u>map emplace_hint()</u>	Inserts the key and its element in the map container with a given hint.
<u>map value_comp()</u>	Returns the object that determines how the elements in the map are ordered ('<' by default).
<u>map key_comp()</u>	Returns the object that determines how the elements in the map are ordered ('<' by default).
<u>map::size()</u>	Returns the number of elements in the map.

Function	Definition
<u>map::empty()</u>	Returns whether the map is empty
<u>map::begin() and end()</u>	begin() returns an iterator to the first element in the map. end() returns an iterator to the theoretical element that follows the last element in the map
<u>map::operator[]</u>	This operator is used to reference the element present at the position given inside the operator.
<u>map::clear()</u>	Removes all the elements from the map.
<u>map::at() and map::swap()</u>	at() function is used to return the reference to the element associated with the key k. swap() function is used to exchange the contents of two maps but the maps must be of the same type, although sizes may differ.

5.4 Duyệt map

Bạn có thể sử dụng range based for loop để duyệt map hoặc duyệt map bằng iterator.

Cách 1 : Duyệt map bằng range based for loop

```
#include <iostream>
#include <algorithm>
#include <map>

using namespace std;

int main(){
    map<int, int> mp;
    mp.insert(make_pair(1, 2)); // ok
    mp.insert(make_pair(2, 4)); // ok
    mp.insert(make_pair(3, 5)); // ok
    for(pair<int, int> it : mp){
        cout << "key = " << it.first << ", value = " << it.second << endl;
        // first = key, second = value
    }
}
```

Output :

```
key = 1, value = 2
key = 2, value = 4
key = 3, value = 5
```

Cách 2 : Duyệt map bằng iterator

```
#include <iostream>
#include <algorithm>
#include <map>

using namespace std;

int main(){
    map<int, int> mp;
    mp.insert(make_pair(1, 2)); // ok
    mp.insert(make_pair(2, 4)); // ok
    mp.insert(make_pair(3, 5)); // ok
    map<int,int>::iterator it;
    for(it = mp.begin(); it != mp.end(); it++){
        cout << "key = " << (*it).first << ", value = " << (*it).second << endl;
    }
}
```

Output :

```
key = 1, value = 2
key = 2, value = 4
key = 3, value = 5
```

Duyệt ngược map:

Khi duyệt ngược từ cuối của map bạn có 2 cách đó là sử dụng `reverse_iterator` hoặc bạn có thể đưa các phần tử trong map lưu vào 1 vector rồi sau đó in ngược lại từ cuối của vector.

```
#include <iostream>
#include <algorithm>
#include <map>

using namespace std;

int main(){
    map<int, int> mp;
    mp.insert(make_pair(1, 2)); // ok
    mp.insert(make_pair(2, 4)); // ok
    mp.insert(make_pair(3, 5)); // ok
    map<int,int>::reverse_iterator it;
    for(it = mp.rbegin(); it != mp.rend(); it++){
        cout << "key = " << (*it).first << ", value = " << (*it).second << endl;
    }
}
```

Output :

```
key = 3, value = 5
key = 2, value = 4
key = 1, value = 2
```

Nếu bạn muốn truy cập vào phần tử đầu tiên trong map thì thông qua iterator `begin()`, còn nếu muốn truy cập vào phần tử cuối cùng trong map thì thông qua iterator `rbegin()`

```
#include <iostream>
#include <algorithm>
#include <map>

using namespace std;

int main(){
    map<int, int> mp;
    mp.insert(make_pair(1, 2)); // ok
    mp.insert(make_pair(2, 4)); // ok
    mp.insert(make_pair(3, 5)); // ok
    map<int, int>::iterator it1 = mp.begin(); // auto ?
    cout << (*it1).first << " " << (*it1).second << endl;
    map<int, int>::reverse_iterator it2 = mp.rbegin(); // auto ?
    cout << (*it2).first << " " << (*it2).second << endl;
}
```

Output :

```
1 2
3 5
```

```
#include <iostream>
#include <map>
#include <utility>
using namespace std;

int main() {
    map<int, int> mp;
    mp[100] = 200;
    mp[200] = 300;
    mp.insert({ 300,400 });
    mp.insert({ 500, 600 });
    for (pair<int, int>x : mp) {
        cout << x.first << " " << x.second << endl; //x = key, y = value
    }
    cout << "===== \n";
    for (auto it : mp) {
        cout << it.first << " " << it.second << endl;
    }
    cout << "===== \n";

    for (map<int, int>::iterator it = mp.begin(); it != mp.end(); it++) {
        cout << (*it).first << " " << (*it).second << endl;
        //cần * để truy xuất được giá trị của pair
    }

    return 0;
}
```

Kết quả:

```
100 200
200 300
300 400
500 600
```

```
=====
100 200
200 300
```

300 400

500 600

=====

100 200

200 300

300 400

500 600

5.5 Ứng dụng

- Tính tần suất xuất hiện của các giá trị được nhập vào.

```
#include <iostream>
#include <map>
#include <utility>
using namespace std;

int main() {
    map<int, int> mp;
    int n; //số lượng số được nhập vào
    cin >> n;
    for (int i = 0; i < n; i++) {
        int x; //giá trị nhập vào
        cin >> x;
        mp[x]++;
    }
    for (auto x : mp) {
        cout << x.first << " " << x.second << endl;
    }
    return 0;
}
```

Kết quả:

8

1 1 2 1 3 5 1 -4

-4 1

1 4

2 1

3 1

5 1

*** In ra theo thứ tự mà người dùng nhập vào**

Ví dụ:

1 1 2 1 3 5 1 -4

Kết quả:

1 4

2 1

3 1

5 1

-4 1

```

#include <iostream>
#include <map>
using namespace std;

int main() {
    map<int, int> mp;
    int n, a[100]; //số lượng số được nhập vào, mảng lưu các giá trị nhập vào
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
        mp[a[i]]++;
    }
    for (int i = 0; i < n; i++) {
        if (mp[a[i]] != 0) {
            cout << a[i] << " " << mp[a[i]] << endl;
            mp[a[i]] = 0; //để phần tử a[i] không in ra tần suất lần nữa.
        }
    }
    return 0;
}

```

- Tính tần suất xuất hiện của các từ được nhập vào.

```

#include <iostream>
#include <map>
#include <utility>
using namespace std;

int main() {
    map<string, int> mp;
    int n; //số lượng từ nhập vào
    cin >> n;
    for (int i = 0; i < n; i++) {
        string s;
        cin >> s;
        mp[s]++;
    }
    cout << "=====\n";
    for (auto i : mp) {
        cout << i.first << " " << i.second << endl;
    }
    return 0;
}

```

Kết quả:

```

9
python
java
string
java
python
C++
sql
sql
java
=====
C++ 1
java 3
python 2

```

sql 2
string 1

- * Sử dụng set khi muốn tìm kiếm/ đếm số lượng phần tử khác nhau trong mảng/ đếm số từ khác nhau trong chuỗi.
- * Sử dụng map khi muốn tìm tần suất xuất hiện các phần tử trong mảng/ các từ trong chuỗi.

5.6 Multimap trong C++

- Tương tự như map thì **multimap** cũng lưu các phần tử là cặp key - value, nó có các hàm tương tự như **map** nhưng có 1 vài điểm khác biệt khi sử dụng.
- Multimap vẫn lưu các phần tử theo thứ tự key tăng dần nhưng **các key có thể trùng nhau**. Và vì các key có thể trùng nhau nên multimap **không hỗ trợ việc bạn truy cập value thông qua key bằng cú pháp map[key]** như map.
- Các tính chất của multimap :
 - Các phần tử trong map có thể có **key trùng nhau**
 - Multimap lưu các phần tử theo thứ tự **tăng dần về key**
 - Multimap **không** hỗ trợ truy cập value thông qua key bằng cú pháp **map[key]**
 - Multimap **không hỗ trợ chỉ số** như mảng hay vector mà chỉ số của multimap chính là key

Mã nguồn :

```
#include <iostream>
#include <map>

using namespace std;

int main(){
    multimap<int, int> mp;
    mp.insert(make_pair(1, 2));
    mp.insert(make_pair(1, 3));
    mp.insert(make_pair(1, 4));
    mp.insert(make_pair(2, 2));
    mp.insert(make_pair(2, 3));
    mp.insert(make_pair(2, 4));
    mp.insert(make_pair(3, 1));
    mp.insert(make_pair(3, 1));
    cout << "Map size : " << mp.size() << endl;
    for(pair<int, int> it : mp){
        cout << it.first << " " << it.second << endl;
    }
    return 0;
}
```

Output :

```
Map size : 8
1 2
1 3
1 4
2 2
2 3
2 4
3 1
3 1
```

- Về độ phức tạp của 3 hàm **find()**, **erase()** và **count()** trong multimap thì tương tự như map là $O(\log N)$. Tuy nhiên có một vài điểm khác biệt các bạn cần lưu ý.

* Hàm find() :

- Do **multimap** có thể lưu trữ nhiều cặp phần tử có cùng key nên khi bạn tìm kiếm key sử dụng hàm **find()** thì sẽ *trả về iterator tới cặp phần tử đầu tiên có key tương ứng* trong map.
- Ví dụ multimap đang lưu các cặp phần tử {(1, 2), (1, 3), (2, 3), (2, 1), (2, 4), (5, 6)} và bạn tìm kiếm key là 2 thì hàm find() sẽ trả về iterator tới cặp phần tử (2, 3) là cặp phần tử có key 2 xuất hiện đầu tiên trong map.

Mã nguồn :

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

int main(){
    multimap<int, int> mp;
    mp.insert(make_pair(1, 2));
    mp.insert(make_pair(1, 3));
    mp.insert(make_pair(2, 3));
    mp.insert(make_pair(2, 1));
    mp.insert(make_pair(2, 4));
    mp.insert(make_pair(5, 6));
    multimap<int, int>::iterator it = mp.find(2);
    cout << (*it).first << " " << (*it).second << endl;
    return 0;
}
```

Output :

2 3

* Hàm erase() :

- Hàm **erase()** nếu bạn xóa thông qua giá trị của key sẽ *xóa toàn bộ cặp phần tử có key tương ứng* trong multimap. Để *xóa đi 1 cặp phần tử duy nhất thì bạn nên xóa thông qua iterator*.

Mã nguồn :

```
#include <iostream>
#include <map>
using namespace std;
int main(){
    multimap<int, int> mp;
    mp.insert(make_pair(1, 2));
    mp.insert(make_pair(1, 3));
    mp.insert(make_pair(2, 3));
    mp.insert(make_pair(2, 1));
    mp.insert(make_pair(2, 4));
    mp.insert(make_pair(5, 6));
    mp.erase(2);
    cout << "Map sau khi xoa : \n";
    for(pair<int, int> it : mp)
        cout << it.first << " " << it.second << endl;
    return 0;
}
```


Output :

Map sau khi xoa :

```
1 2
1 3
5 6
```

* Hàm count() :

- Hàm **count()** trong multimap khi bạn sử dụng để *đếm số lần xuất hiện của key*, trả về giá trị tương ứng với số cặp phần tử có key tương ứng. *Nếu key bạn đếm không xuất hiện thì hàm này trả về 0.*

Mã nguồn :

```
#include <iostream>
#include <map>

using namespace std;

int main(){
    multimap<int, int> mp;
    mp.insert(make_pair(1, 2));
    mp.insert(make_pair(1, 3));
    mp.insert(make_pair(2, 3));
    mp.insert(make_pair(2, 1));
    mp.insert(make_pair(2, 4));
    mp.insert(make_pair(5, 6));
    cout << "Key 2 xuất hiện : " << mp.count(2) << " lan\n";
    cout << "key 28 xuất hiện : " << mp.count(28) << " lan\n";
    return 0;
}
```

Output :

```
Key 2 xuất hiện : 3 lan
key 28 xuất hiện : 0 lan
```

List of Functions of Multimap

Function	Definition
<u>multimap::operator=</u>	It is used to assign new contents to the container by replacing the existing contents.
<u>multimap::crbegin()</u> and <u>multimap::crend()</u>	crbegin() returns a constant reverse iterator referring to the last element in the multimap container. crend() returns a constant reverse iterator pointing to the theoretical element before the first element in the multimap.
<u>multimap::emplace_hint()</u>	Insert the key and its element in the multimap container with a given hint.
<u>multimap clear()</u>	Removes all the elements from the multimap.

Function	Definition
<u>multimap empty()</u>	Returns whether the multimap is empty.
<u>multimap maxsize()</u>	Returns the maximum number of elements a multimap container can hold.
<u>multimap value_comp()</u>	Returns the object that determines how the elements in the multimap are ordered ('<' by default).
<u>multimap rend</u>	Returns a reverse iterator pointing to the theoretical element preceding to the first element of the multimap container.
<u>multimap::cbegin() and multimap::cend()</u>	cbegin() returns a constant iterator referring to the first element in the multimap container. cend() returns a constant iterator pointing to the theoretical element that follows the last element in the multimap.
<u>multimap::swap()</u>	Swap the contents of one multimap with another multimap of same type and size.
<u>multimap rbegin</u>	Returns an iterator pointing to the last element of the container.
<u>multimap size()</u>	Returns the number of elements in the multimap container.
<u>multimap::emplace()</u>	Inserts the key and its element in the multimap container.
<u>multimap::begin() and multimap::end()</u>	begin() returns an iterator referring to the first element in the multimap container. end() returns an iterator to the theoretical element that follows the last element in the multimap.
<u>multimap upper_bound()</u>	Returns an iterator to the first element that is equivalent to multimapped value with key-value 'g' or definitely will go after the element with key-value 'g' in the multimap.
<u>multimap::count()</u>	Returns the number of matches to element with key-value 'g' in the multimap.
<u>multimap::erase()</u>	Removes the key value from the multimap.
<u>multimap::find()</u>	Returns an iterator to the element with key-value 'g' in the multimap if found, else returns the iterator to end.

Function	Definition
<u>multimap equal_range()</u>	Returns an iterator of pairs. The pair refers to the bounds of a range that includes all the elements in the container which have a key equivalent to k.
<u>multimap insert()</u>	Used to insert elements in the multimap container.
<u>multimap lower_bound()</u>	Returns an iterator to the first element that is equivalent to multimapped value with key-value 'g' or definitely will not go before the element with key-value 'g' in the multimap.
<u>multimap key_comp()</u>	Returns the object that determines how the elements in the multimap are ordered ('<' by default).

5.7 Unordered_map trong C++

- Tương tự như map thì **unordered_map** được sử dụng để lưu trữ dữ liệu theo cặp key - value. Các hàm của map thì **unordered_map** đều có nhưng khác nhau về độ phức tạp.

- **unordered_map** được cài đặt bởi cấu trúc dữ liệu **bảng băm - hash table**. Bạn cần thêm thư viện "unordered_map" vào để có thể sử dụng container này.

- Các tính chất của unordered_map :

- unordered_map lưu trữ phần tử với các **key là riêng biệt**
- Các phần tử trong unordered_map **không có thứ tự**

Mã nguồn :

```
#include <iostream>
#include <map>
#include <unordered_map>

using namespace std;

int main(){
    unordered_map<string, string> mp;
    mp["28tech"] = "28tech.com.vn";
    mp["USA"] = "Trump";
    mp["Russia"] = "Putin";
    for(pair<string, string> it : mp){
        cout << it.first << " " << it.second << endl;
    }
    return 0;
}
```

Output :

```
Russia Putin
USA Trump
28tech 28tech.com.vn
```

- Các hàm find(), count(), erase()

+ Cách dùng 3 hàm phổ biến này của `unordered_map` *tương tự như map*, điều khác biệt duy nhất đó là vì `unordered_map` được cài đặt bởi bảng băm thay vì cây đỏ đen Red-black tree như `map` nên sẽ có sự khác nhau về độ phức tạp.

+ Độ phức tạp trung bình của 3 hàm này là **$O(1)$** , tuy nhiên trong trường hợp tệ nhất (Worst case) thì nó có thể là **$O(N)$** .

+ Hàm `insert` trong `unordered_map` cũng tương tự như vậy về độ phức tạp.

Mã nguồn :

```
#include <iostream>
#include <map>
#include <unordered_map>

using namespace std;

int main(){
    unordered_map<string, string> mp;
    mp["28tech"] = "28tech.com.vn";
    mp["USA"] = "Trump";
    mp["Russia"] = "Putin";
    mp.erase("USA");
    for(pair<string, string> it : mp){
        cout << it.first << " " << it.second << endl;
    }
    return 0;
}
```

Output :

```
Russia Putin
28tech 28tech.com.vn
```

Methods on unordered_map

A lot of functions are available that work on `unordered_map`. The most useful of them are:

- **operator =**
- **operator []**
- **empty**
- **size for capacity**
- **begin and end for the iterator.**
- **find and count for lookup.**
- **insert and erase for modification.**

The below table shows the complete list of the methods of an `unordered_map`:

Methods/Functions	Description
<code>at()</code>	This function in C++ <code>unordered_map</code> returns the reference to the value with the element as key k
<code>begin()</code>	Returns an iterator pointing to the first element in the container in the <code>unordered_map</code> container
<code>end()</code>	Returns an iterator pointing to the position past the last element in the container in the <code>unordered_map</code> container

Methods/Functions	Description
<u>bucket()</u>	Returns the bucket number where the element with the key k is located in the map
<u>bucket_count</u>	Bucket_count is used to count the total no. of buckets in the unordered_map. No parameter is required to pass into this function
<u>bucket_size</u>	Returns the number of elements in each bucket of the unordered_map
<u>count()</u>	Count the number of elements present in an unordered_map with a given key
<u>equal_range</u>	Return the bounds of a range that includes all the elements in the container with a key that compares equal to k
<u>find()</u>	Returns iterator to the element
<u>empty()</u>	Checks whether the container is empty in the unordered_map container
<u>erase()</u>	Erase elements in the container in the unordered_map container

The C++11 library also provides functions to see internally used bucket count, bucket size, and also used hash function and various hash policies but they are less useful in real applications. We can iterate over all elements of unordered_map using Iterator.