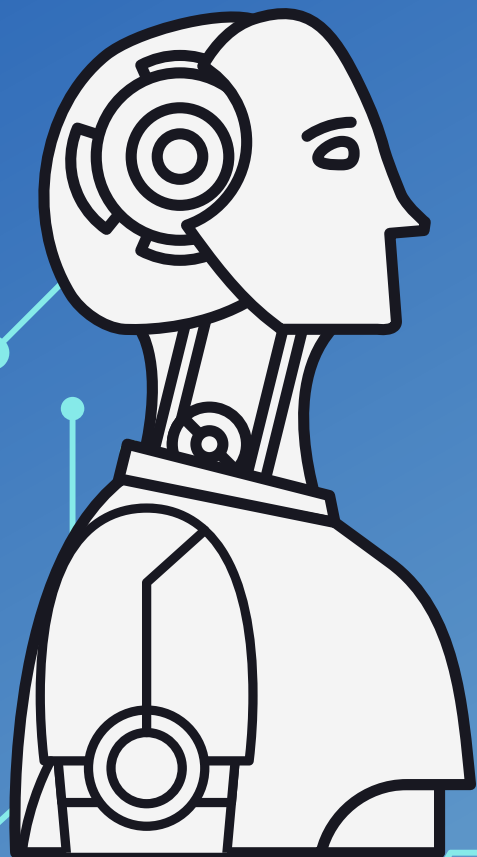
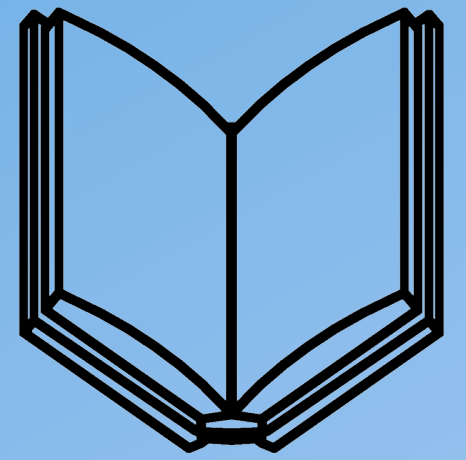
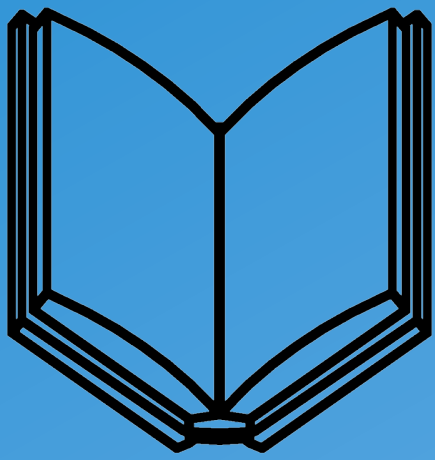


**Nhóm 16**

# **THƯ VIỆN SET, MAP**



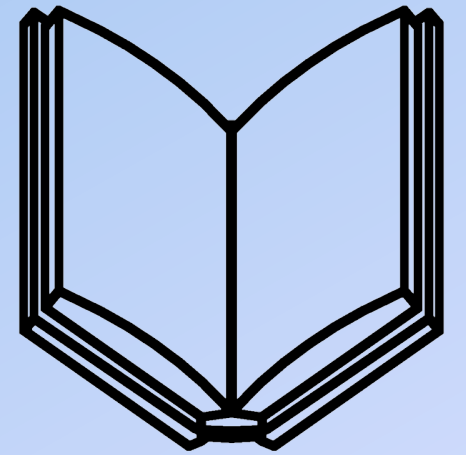
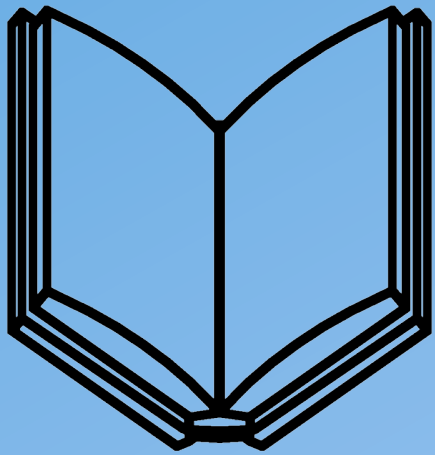


# ***Thành viên nhóm***

Vũ Gia Khang

Ngô Minh Huy

Nguyễn Đức Hưởng





```
graph TD; A((Nội Dung)) --- B[Thư viện Set]; A --- C[Thư viện Map];
```

# **Nội Dung**

Thư viện Set

Thư viện Map



**Set**

# Set

## Khái niệm

Set là một container dùng để lưu trữ các giá trị với các tính chất:

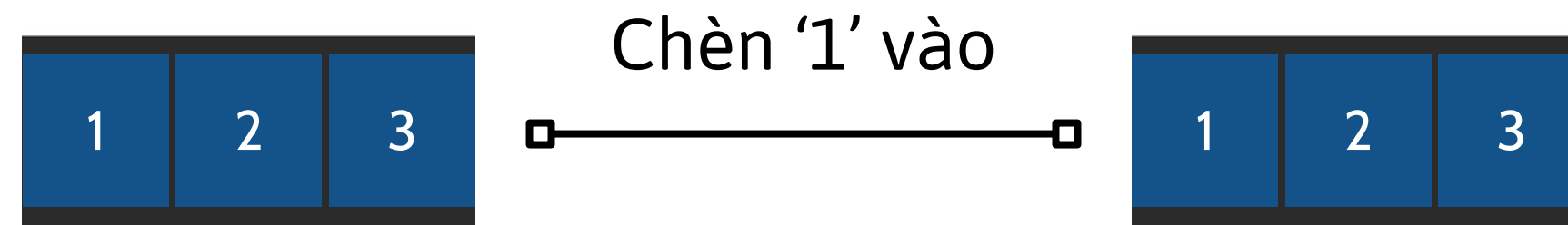
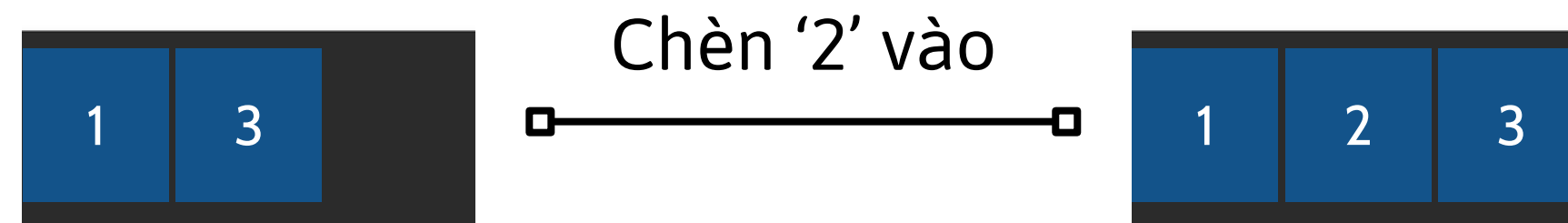
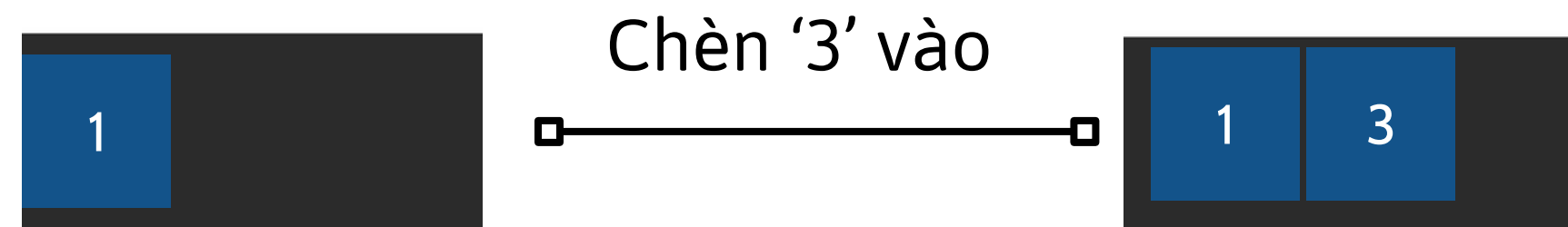
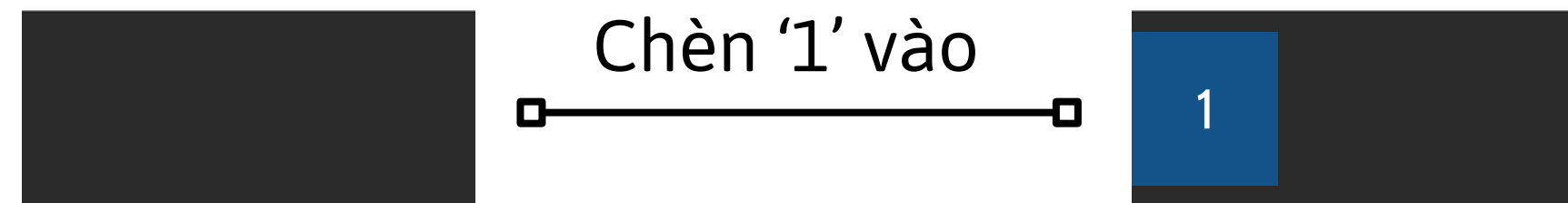
Các giá trị được lưu trữ **không trùng nhau** và có **thứ tự tăng dần** (mặc định) hoặc **giảm dần**

**Giá trị** của các phần tử là **const**, chỉ có thể được **chèn** hoặc **xóa**

Hỗ trợ tốt các thao tác chèn, xóa, tìm kiếm ( $O(\log n)$ )

# Set

## Cách thức hoạt động



# Các thao tác với Set

## 1) Khai báo

Khởi tạo 1 set rỗng

Khai báo thư viện set:

```
#include <set>
```

```
set <{Kiểu_phần_tử}> {Tên_set};
```

```
1  #include <set>
2  using namespace std;
3
4  int main()
5  {
6      set<int> a;
7      return 0;
8  }
```

# Các thao tác với Set

## 1) Khai báo

Khởi tạo 1 set từ mảng khác

Khai báo thư viện: `#include <set>`

`set <{Kiểu_phần_tử}> {Tên_set}` (Vị trí bắt đầu, vị trí kết thúc);

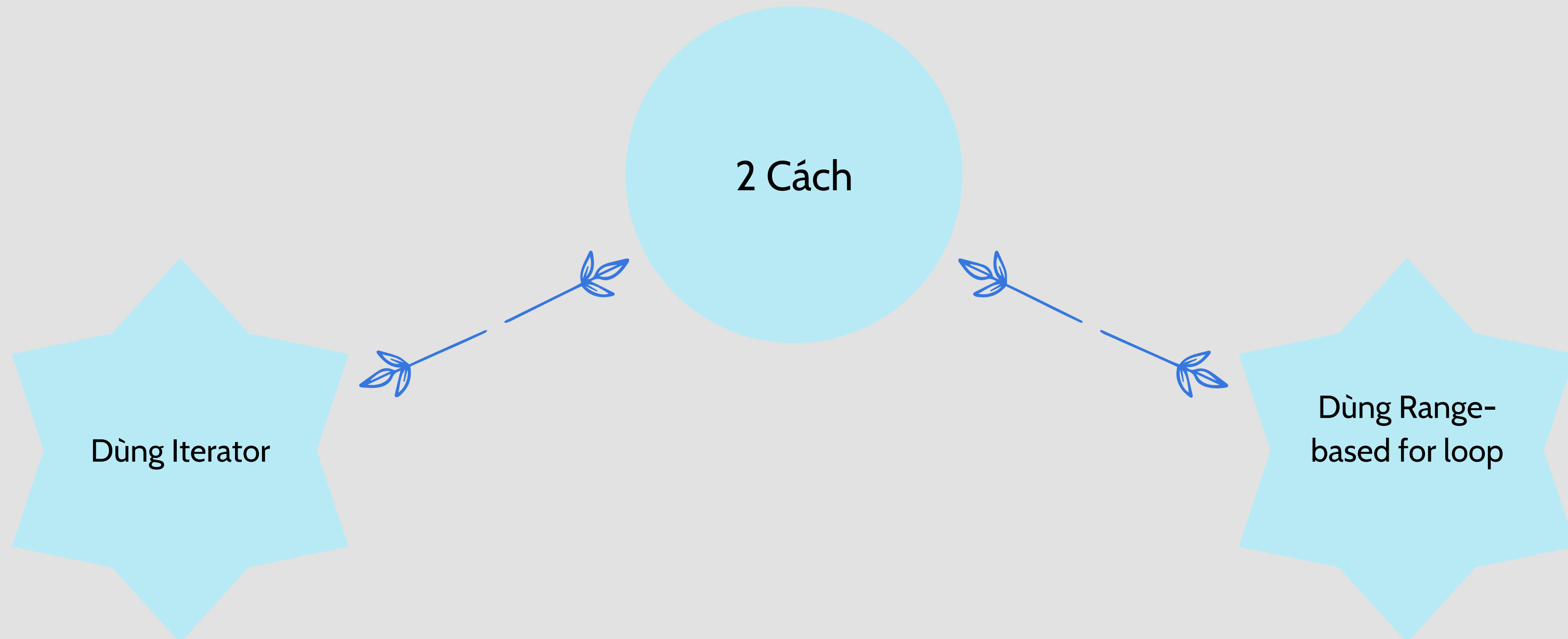
```
1  #include <iostream>
2  #include <set>
3
4  using namespace std;
5
6  int main()
7  {
8      int a[] = {1, 2, 3, 4, 5};
9      set<int> s(a, a + 5); // s = {1, 2, 3, 4, 5}
10
11     return 0;
12 }
```



# Các thao tác với Set

## 2) Duyệt set

Các phần tử trong set không thể truy cập trực tiếp qua vị trí, mà buộc phải sử dụng vòng lặp để duyệt.

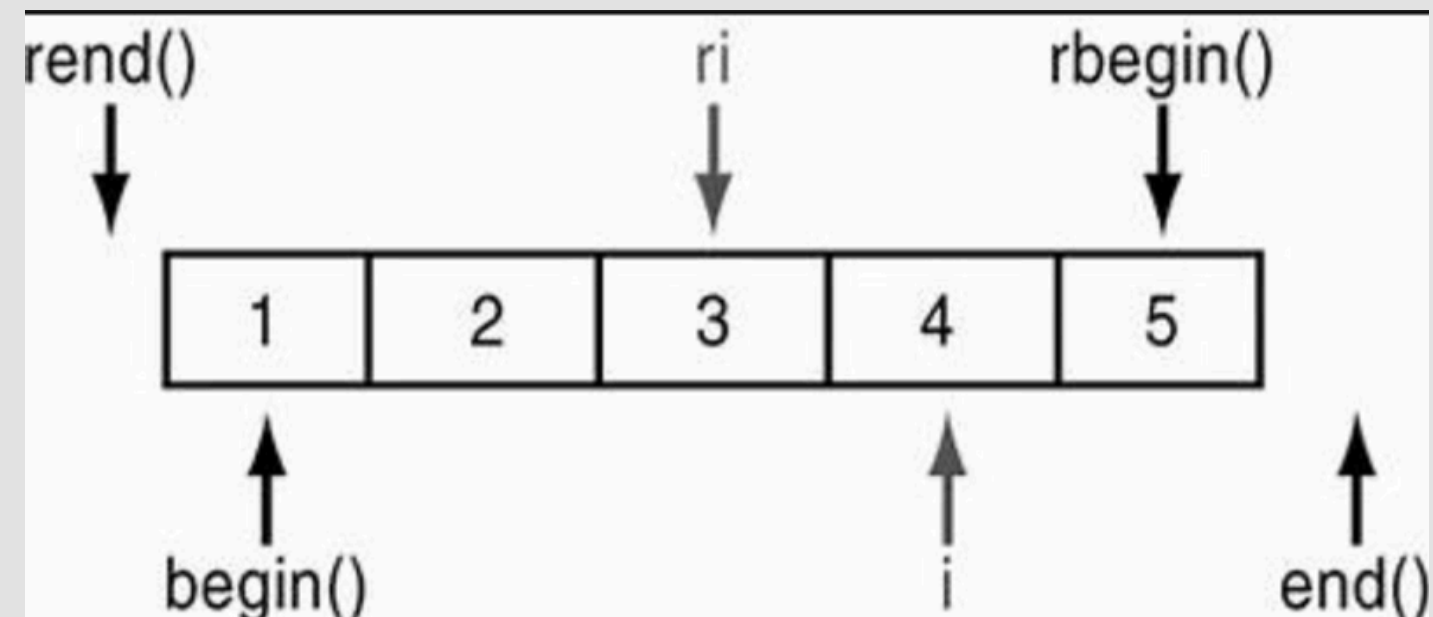


# Cách 1: Dùng Iterator

## Giới thiệu về Iterator

Giống như một con trỏ, Iterator dùng để duyệt qua các phần tử có trong container

Container chứa định nghĩa class Iterator sẽ có phương thức trả về giá trị kiểu Iterator tương ứng



## Cách khai báo

```
set<kieu_du_lieu> :: iterator <ten_cua_iterator>
```

Ví dụ:

```
set<int>::iterator it;
```

# Cách 1: Dùng Iterator

## Các toán tử trên Iterator

- =   trỏ iterator đến vị trí gán.
- ++   di chuyển iterator đến phần tử tiếp theo.
- di chuyển iterator đến phần tử trước đó.
- \*   trả về tham chiếu đến đối tượng được it trỏ tới.
- ==   true nếu 2 iterator trỏ cùng 1 vị trí, ngược lại false.
- !=   false nếu 2 iterator trỏ cùng 1 vị trí, ngược lại true.

```
set<int> s = {1, 2, 3, 4, 5};
set<int>::iterator it;
it = s.begin();    // *it = 1
it++;             // *it = 2
it--;             // *it = 1
```

```
set<int> s = {1, 2, 3, 4, 5};
set<int>::iterator it1 = s.begin(); // *it = 1;
set<int>::iterator it2 = it1++;     // *it = 2;

bool isTrue = it1 == it2;
cout << isTrue;   //0
```

# Cách 1: Dùng Iterator

## Cách dùng

```
for ({Tên_Iterator} = {Địa_chỉ_đầu}; {Biến_lặp} != {Địa_chỉ_cuối}; ++{Biến_lặp})  
{  
    <Các thao tác>  
}
```

Ví dụ: Xuất set ra màn hình

```
set<int> s = {1, 2, 3, 4, 5};  
set<int>::iterator it;  
for(it = s.begin(); it != s.end(); ++it)  
    cout << *it << endl;
```

# Cách 2: Dùng Range-based for loop

## Giới thiệu về Range-based for loop

Range-based for loop (vòng lặp for-each) là vòng lặp duyệt qua các phần tử trong container tuần tự từ đầu tới cuối.

## Cách khai báo

```
for ( range_declaration : range_expression )  
{  
    //code  
}
```

```
int a[] = {0, 1, 2, 3};  
for (int x : a)  
    cout << x << " "; //0 1 2 3
```

# Cách 2: Dùng Range-based for loop

## Áp dụng Range-based for loop vào Set

- **Sử dụng auto:** auto là từ khóa giúp suy luận ra kiểu dữ liệu tương ứng phù hợp với một biến thông qua giá trị mà biến được gán.
- Ví dụ:

```
auto integerNum = 2;  
auto realNum = 10.7;  
auto String = "Hello World";
```

- integerNum có kiểu dữ liệu là số nguyên (int)
- realNum có kiểu dữ liệu là số thực (float/double)
- String có kiểu dữ liệu là chuỗi (string)

# Cách 2: Dùng Range-based for loop

## Áp dụng Range-based for loop vào Set

```
for (<kieu du lieu> <tên biến> : <tên của set>)  
{  
    //code  
}
```

```
for (auto x : s)  
{  
    //code  
}
```

```
set<int> s;  
for (int x : s)  
{  
    //code  
}
```

Ví dụ: Xuất set ra màn hình

```
set<int> s = {0, 1, 2, 3, 4};  
  
for (auto x : s)  
    cout << x << endl;
```

# Một số hàm của Set

- Hàm insert: thêm 1 phần tử.
- Hàm emplace: thêm 1 phần tử
- Hàm erase: xóa 1 phần tử.
- Hàm find: tìm kiếm phần tử.
- Lower\_bound: tìm phần tử.
- Upper\_bound: tìm phần tử.

**insert:** Thêm một phần tử vào tập hợp bằng cách sao chép đối tượng đã cho vào tập hợp.

**Cú pháp:**

Set\_name.insert(value);

Set\_name.insert(InputIterator first, InputIterator last );

```
1  #include <iostream>
2  #include <set>
3  using namespace std;
4
5  int main()
6  {
7      int a[] = {6, 7, 8, 9, 10};
8      set<int> s = {1, 2, 3, 4};
9      s.insert(5);
10     s.insert(a, a + 5);
11     for (auto it : s)
12         cout << it << " ";
13     return 0;
14 }
15
```

→ 1 2 3 4 5 6 7 8 9 10



# Một số hàm của Set

- Hàm insert: thêm 1 phần tử.
- Hàm emplace: thêm 1 phần tử
- Hàm erase: xóa 1 phần tử.
- Hàm find: tìm kiếm phần tử.
- Lower\_bound: tìm phần tử.
- Upper\_bound: tìm phần tử.

**emplace:** Thêm một phần tử vào tập hợp bằng cách xây dựng đối tượng trong chính tập hợp.

**Cú pháp:**

**Set\_name.emplace(value);**

```
1  #include <iostream>
2  #include <set>
3  using namespace std;
4
5  int main()
6  {
7      set<int> s = {1, 2, 3, 4};
8      s.emplace(1);
9      s.emplace(5);
10     for (auto it : s)
11         cout << it << " ";
12     return 0;
13 }
```

→ 1 2 3 4 5

# Một số hàm của Set

**erase:** được sử dụng để xóa một hoặc nhiều phần tử khỏi tập hợp.

## Cú pháp:

Set\_name.erase ( value );

Set\_name.erase ( vị trí cần xóa );

Set\_name.erase ( vị trí bắt đầu xóa, vị trí kết thúc xóa );

// xóa giá trị ở vị trí bắt đầu, không xóa ở vị trí kết thúc

- Hàm insert: thêm 1 phần tử.
- Hàm emplace: thêm 1 phần tử
- Hàm erase: xóa 1 phần tử.
- Hàm find: tìm kiếm phần tử.
- Lower\_bound: tìm phần tử.
- Upper\_bound: tìm phần tử.

```
1  #include <iostream>
2  #include <set>
3  using namespace std;
4
5  int main()
6  {
7      set<int> s = {1, 2, 3, 4, 5, 6, 7, 8};
8      s.erase(1);           // 2, ..., 8
9      s.erase(s.begin());   // 3, ..., 8
10
11     auto it1 = s.begin();
12     auto it2 = s.find(6);
13     s.erase(it1, it2);
14
15     for (auto i : s)
16         cout << i << " ";
17     return 0;
18 }
```



6 7 8

# Một số hàm của Set

- Hàm insert: thêm 1 phần tử.
- Hàm emplace: thêm 1 phần tử
- Hàm erase: xóa 1 phần tử.
- Hàm find: tìm kiếm phần tử.
- Lower\_bound: tìm phần tử.
- Upper\_bound: tìm phần tử.

**find:** trả về vị trí của value đang tìm, nếu có tồn tại value đó trong set thì sẽ trả về vị trí của value đó, không thì sẽ trả về vị trí end() của set.

**Cú pháp:**

Set\_name.find ( key ) ;

```
1  #include <iostream>
2  #include <set>
3  using namespace std;
4
5  int main()
6  {
7      set<int> s = {1, 2, 3, 4, 5};
8      set<int>::iterator it1 = s.find(3);
9      set<int>::iterator it2 = s.find(10);
10
11     if (it1 != s.end())
12         cout << "Tim thay '3'" << endl;
13     else
14         cout << "khong tim thay" << endl;
15     if (it2 != s.end())
16         cout << "Tim thay '10'" << endl;
17     else
18         cout << "khong tim thay" << endl;
19
20     return 0;
21 }
```

Tim thay '3'  
khong tim thay

# Một số hàm của Set

- Hàm insert: thêm 1 phần tử.
- Hàm emplace: thêm 1 phần tử
- Hàm erase: xóa 1 phần tử.
- Hàm find: tìm kiếm phần tử.
- Lower\_bound: tìm phần tử.
- Upper\_bound: tìm phần tử.

1. **lower\_bound**: trả về vị trí của phần tử đầu tiên trong tập hợp có giá trị lớn hơn hoặc bằng giá trị cần tìm.
2. **upper\_bound**: trả về vị trí của phần tử đầu tiên trong tập hợp có giá trị lớn hơn giá trị cần tìm.

## Cú pháp:

Set\_name.lower\_bound ( giá trị của key );

Set\_name.upper\_bound ( giá trị của key );

```
1  #include <iostream>
2  #include <set>
3  using namespace std;
4
5  int main()
6  {
7      set<int> s = {10, 15, 20, 25, 30};
8      auto it_lower = s.lower_bound(15);
9      auto it_upper = s.upper_bound(15);
10
11     cout << *it_lower << endl;
12     cout << *it_upper << endl;
13     return 0;
14 }
```



15

20

# Một số hàm khác của Set

## 1) Hàm size, clear, swap, empty: • Cú pháp:

- Hàm size: cho biết số lượng phần tử.
  - Hàm clear: xóa tất cả phần tử.
  - Hàm swap: đổi các phần tử của 2 set;
  - Hàm empty: kiểm tra xem có rỗng không
- set\_name.size()

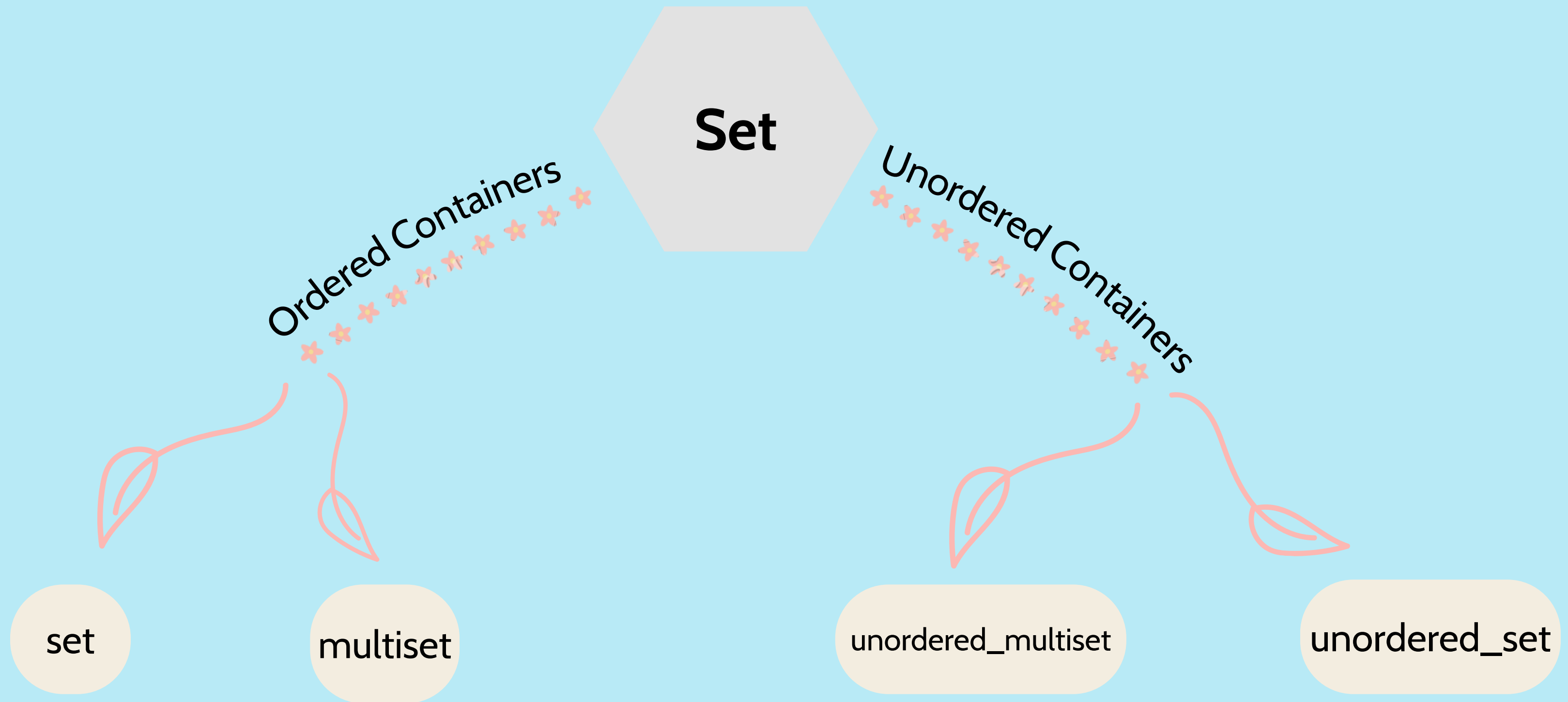
set\_name.clear()

set\_name1.swap(set\_name2)

set\_name.empty();

```
1  #include <iostream>
2  #include <set>
3  using namespace std;
4
5  int main()
6  {
7      set<int> s = {1, 2, 3, 4, 5, 6};
8
9      cout << s.size() << endl; // 6
10     s.clear();
11     cout << s.size() << endl; // 0
12
13     bool isTrue = s.empty();
14     cout << isTrue << endl;    // 1 (true)
15
16     return 0;
17 }
```

```
1  #include <iostream>
2  #include <set>
3  using namespace std;
4
5  int main()
6  {
7      set<int> s1 = {1, 2, 3, 4, 5};
8      set<int> s2 = {6, 7, 8, 9, 10};
9
10     for (auto it : s1)
11         cout << it << " ";
12     cout << endl;
13     for (auto it : s2)
14         cout << it << " ";
15     cout << endl;
16
17     s1.swap(s2);
18
19     cout << "Sau khi doi: " << endl;
20     for (auto it : s1)
21         cout << it << " ";
22     cout << endl;
23     for (auto it : s2)
24         cout << it << " ";
25
26     return 0;
27 }
```



# multiset

- Cũng giống như set, nhưng cho phép lưu trữ nhiều phần tử có cùng giá trị với nhau
- Khai báo: `#include <set>`  
`multiset <{kiểu dữ liệu}> {tên};`

```
1  #include <set>
2  using namespace std;
3
4  int main()
5  {
6      multiset <int> s = {1, 2, 3, 4};
7
8      return 0;
9  }
```

```
multiset<int> s = {1, 2, 3, 4};
s.insert(2);
s.insert(3);
for (auto x : s)
    cout << x << " ";
```



1 2 2 3 3 4

## unordered\_set

- Các phần tử được thêm vào không sắp theo thứ tự.
- Giống với set, các phần tử trong unordered\_set không trùng nhau.
- **Khai báo:**        `#include <unordered_set>`  
                         `unordered_set <{kiểu dữ liệu}> {tên};`

```
1  #include <unordered_set>
2  using namespace std;
3
4  int main()
5  {
6      unordered_set<int> s;
7      return 0;
8  }
```





**Map**

# Một vài điều về pair

- \* Là một container lưu trữ dữ liệu theo cặp, với cách gọi first cho phần tử thứ nhất và second cho phần tử thứ hai .
- \* Các phần tử trong cặp không nhất thiết phải có kiểu dữ liệu giống nhau.
- \* Cần khai báo thư viện utility trước khi sử dụng pair.

# Các thao tác với Pair

## 1) Khai báo

Pair < kiểu dữ liệu của phần tử thứ nhất, kiểu dữ liệu của phần tử thứ hai > Pair\_name;

## 2) Các cách gán giá trị cho pair

Pair < kiểu dữ liệu, kiểu dữ liệu > pair\_name (giá trị thứ nhất, giá trị thứ hai );

Pair < kiểu dữ liệu, kiểu dữ liệu > pair\_name = { giá trị thứ nhất, giá trị thứ hai };

Pair\_name = make\_pair ( giá trị thứ nhất, giá trị thứ hai );

## 3) Truy xuất trong pair

Pair\_name.first ( truy xuất đến first )

Pair\_name.second ( truy xuất đến second )

# Một vài điều về map

- \* Map là một container chứa các phần tử có kiểu dữ liệu là các cặp giá trị ( pair ) gồm key và value .
- \* Key và Value không nhất thiết phải có kiểu dữ liệu giống nhau.
- \* Cần khai báo thư viện map trước khi sử dụng map.

# Một vài điều về map

- \* Trong cùng một map thì không bao giờ tồn tại hai key có cùng giá trị.
- \* Map mặc định sẽ sắp xếp các phần tử của nó theo thứ tự tăng dần theo các key.
- \* Không thể truy cập tiếp các phần tử trong map một cách trực tiếp như trong mảng hay là vector.
- \* Có thể truy xuất trực tiếp tới phần tử ở trong map qua `[]` hoặc qua `"at"` kết hợp với giá trị của key.

# Các thao tác với map

## 1) Khai báo

Khởi tạo 1 map rỗng

Khai báo thư viện map:

```
#include <map>
```

```
map <{datatype1,datatype2}> Tên_map;
```

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

using namespace std;

int main()
{
    map<string, int>a;
    return 0;
}
```

# Các thao tác với map

## 1) Khai báo

Khởi tạo 1 map từ  
map khác

```
map <datatype1,datatype2> Tên_map2(Tên_map1);
```

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

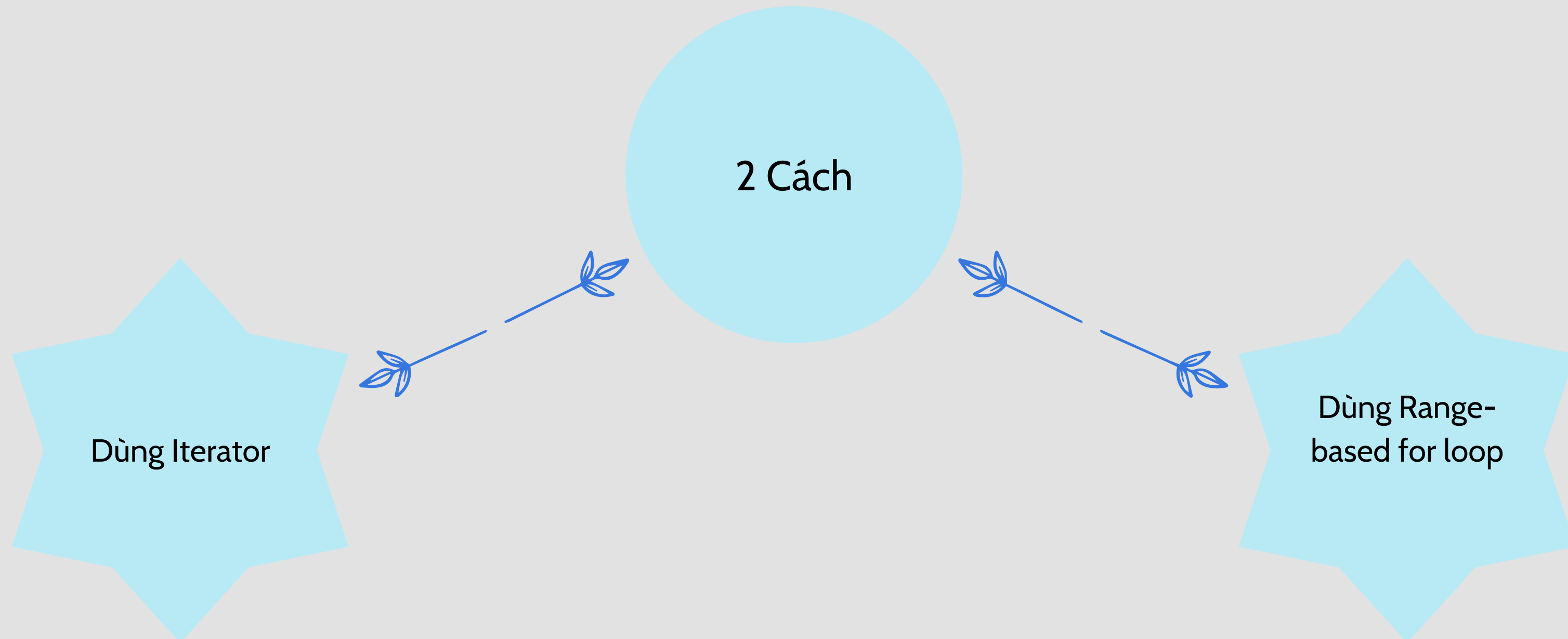
using namespace std;

int main()
{
    map<string, int> b;
    map<string, int>a(b);
    return 0;
}
```

# Các thao tác với map

## 2) Duyệt map

Tương tự set, các phần tử trong map không thể truy cập trực tiếp qua vị trí, mà buộc phải sử dụng vòng lặp để duyệt.





# Cách 1: Dùng Iterator

## Cách dùng

Cú pháp: `map< datatype1, datatype 2 > :: iterator {tên_biến};`

```
for ({Tên_Iterator} = {Địa_chỉ_bắt_đầu}; {Biến_lặp} != {Địa_chỉ_kết_thúc}; ++{Biến_lặp})  
{  
    <Các thao tác>  
}
```

Ví dụ: Xuất map ra màn hình

```
#include<iostream>  
#include<iomanip>  
#include<map>  
  
using namespace std;  
  
int main()  
{  
    std::map<char, int> mp;  
    mp['a'] = 1; mp['b'] = 2; mp['c'] = 3; mp['d'] = 4;  
    mp['e'] = 5; mp['f'] = 6; mp['g'] = 7;  
    for (map<char, int>::iterator it = mp.begin(); it != mp.end(); it++)  
        cout << it->first << " " << it->second << endl;  
    return 0;  
}
```

# Cách 2: Dùng Range-based for loop

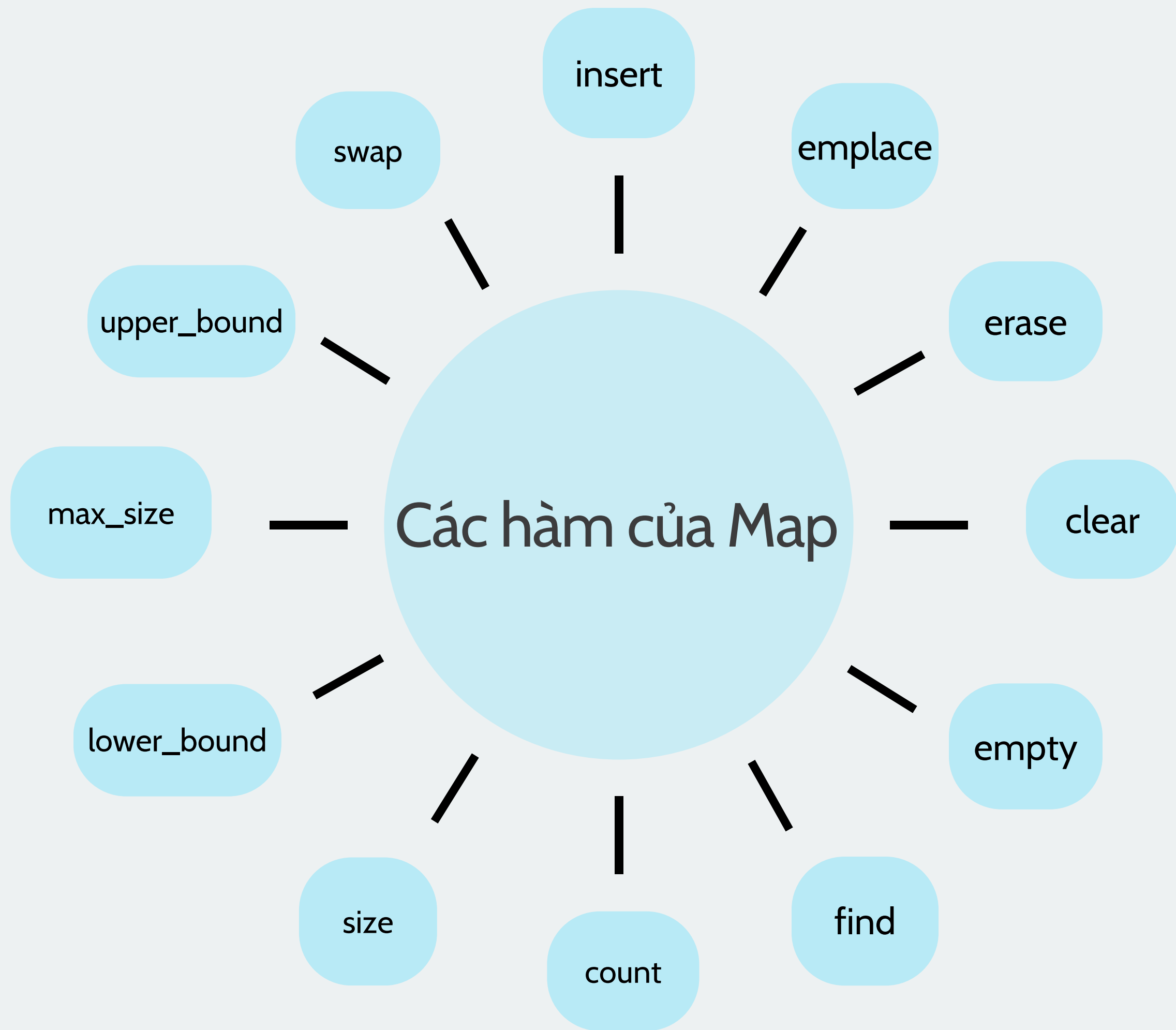
## Áp dụng Range-based for loop vào map

```
for (auto <tên biến> : <tên của map>)  
{  
    //code  
}
```

```
for (auto x : s)  
{  
    //code  
}
```

## Ví dụ: Xuất map ra màn hình

```
#include<iostream>  
#include<iomanip>  
#include<map>  
  
using namespace std;  
  
int main()  
{  
    std::map<char, int> mp;  
    mp['a'] = 1; mp['b'] = 2; mp['c'] = 3; mp['d'] = 4;  
    mp['e'] = 5; mp['f'] = 6; mp['g'] = 7;  
    for(auto it: mp)  
        cout << it.first << " " << it.second << endl;  
    return 0;  
}
```



# Các hàm của map

**1) Insert, Emplace:** chèn thêm phần tử vào trong map.

**Cú pháp: ( insert và emplace giống nhau )**

Map\_name.insert ( pair < kiểu dữ liệu của key, kiểu dữ liệu của value >  
( key, value ) );

Map\_name.insert ( { key, value } ) ;

Map\_name.insert (make\_pair (key,value) ) ;

Map\_name.insert ( vị trí cần chèn, { key, value } ) ;

# Các hàm của map

## 1) Insert, Emplace: chèn thêm phần tử vào trong map.

```
map3
1 #include<iostream>
2 #include<map>
3 using namespace std;
4
5 int main()
6 {
7     std::map<char, int> mp, mp1;
8     mp.insert(std::pair<char, int>('a', 1)); //cach 1
9     mp.insert(std::pair<char, int>('b', 2));
10    mp.insert(make_pair('f', 5)); //cach 2
11    mp.insert({ 'c', 3 });
12    mp.insert({ 'd', 4 }); //cach 3
13    mp['g'] = 7;
14    mp.insert(mp.find('d'), { 'e', 5 }); //cach 4
15    mp1.insert(mp.begin(), mp.find('d'));
16    cout << "mp" << endl;
17    for(auto& it : mp)
18    {
19        cout << it.first << " " << it.second << endl;
20    }
21    cout << "\nmp1" << endl;
22    for (auto& it : mp1)
23    {
24        cout << it.first << " " << it.second << endl;
25    }
26    return 0;
27 }
28
```

```
1 #include<iostream>
2 #include<map>
3 using namespace std;
4
5 int main()
6 {
7     std::map<char, int> mp, mp1;
8
9     // Insert elements using emplace
10    mp.emplace('a', 1); // Insert 'a' with value 1
11    mp.emplace('b', 2); // Insert 'b' with value 2
12    mp.emplace('f', 5); // Insert 'f' with value 5
13    mp.emplace('c', 3); // Insert 'c' with value 3
14    mp.emplace('d', 4); // Insert 'd' with value 4
15
16    mp['g'] = 7;
17    mp.emplace_hint(mp.find('d'), 'e', 5);
18    mp1.emplace(mp.begin(), mp.find('d'));
19    cout << "mp" << endl;
20    for (auto& it : mp)
21    {
22        cout << it.first << " " << it.second << endl;
23    }
24    cout << "\nmp1" << endl;
25    for (auto& it : mp1)
26    {
27        cout << it.first << " " << it.second << endl;
28    }
29    return 0;
30 }
31
```

# Các hàm của map

**2) Erase:** xóa phần tử ở trong map.

**3) Find:** trả vị trí của key đang tìm.

- **Cú pháp:**

Map\_name.erase ( key );

Map\_name.erase ( vị trí cần xóa ( sử dụng iterator ) );

Map\_name.erase ( vị trí bắt đầu xóa, vị trí kết thúc xóa );

Map\_name.find ( key );

```
{
    std::map<char, int> mp;
    mp['a'] = 1; mp['b'] = 2; mp['c'] = 3; mp['d'] = 4;
    mp['e'] = 5; mp['f'] = 6; mp['g'] = 7;
    cout << "Truoc khi xoa " << endl;
    for (auto& it : mp)
        cout << it.first << " " << it.second << endl;
    mp.erase('7'); //xóa phần tử tại key có giá trị 7
    mp.erase('a'); //xóa phần tử tại key có giá trị a
    mp.erase(mp.find('c')); //xóa phần tử tại key có giá trị c ( xóa bằng iterator )
    mp.erase(mp.begin(), mp.find('e')); //xóa các phần tử nằm trong vùng từ key thứ nhất của map tới key trước key e
    cout << "Sau khi xoa " << endl;
    for (auto& it : mp)
        cout << it.first << " " << it.second << endl;
    return 0;
}
```

# Các hàm của map

## 4) lower\_bound, upper\_bound

- lower\_bound: trả vị trí của phần tử đầu tiên trong tập hợp có giá trị **lớn hơn hoặc bằng** giá trị cần tìm.
- upper\_bound: trả vị trí của phần tử đầu tiên trong tập hợp có giá trị **lớn hơn** giá trị cần tìm.  
→ Nếu không tìm thấy, trả đến vị trí end().

- Cú pháp:

Map\_name.lower\_bound ( giá trị của key );

Map\_name.upper\_bound ( giá trị của key );

```
map<char, int> mymap;
std::map<char, int>::iterator itlow, itup;

mymap['a'] = 20;
mymap['b'] = 40;
mymap['c'] = 60;
mymap['d'] = 80;
mymap['e'] = 100;

itlow = mymap.lower_bound('b'); // itlow trả về b
itup = mymap.upper_bound('d');  // itup trả về e (not d!)

mymap.erase(itlow, itup);      // xóa [itlow,itup)

// print nội dung
for (auto& it : mymap)
    std::cout << it.first << " => " << it.second << '\n';

return 0;
```

# Các hàm của map

## 4) **clear, empty, size, max\_size, count, swap**

- **Clear:** xóa tất cả các phần tử có ở trong map.
- **Empty:** kiểm tra xem map có rỗng hay không.
- **Size:** xuất ra số lượng phần tử mà map đang có.
- **Max\_size:** xuất ra số lượng tối đa phần tử mà map có thể có.
- **Count:** kiểm tra xem có key cần tìm ở trong map hay không.
- **Swap:** đổi các phần tử mà hai map đang giữ với nhau.

- **Cú pháp:**

`Map_name.clear();`

`Map_name.empty();`

`Map_name.size();`

`Map_name.max_size();`

`Map_name.count();`

`Map_name1.swap(Map_name2);`



## Đặc biệt

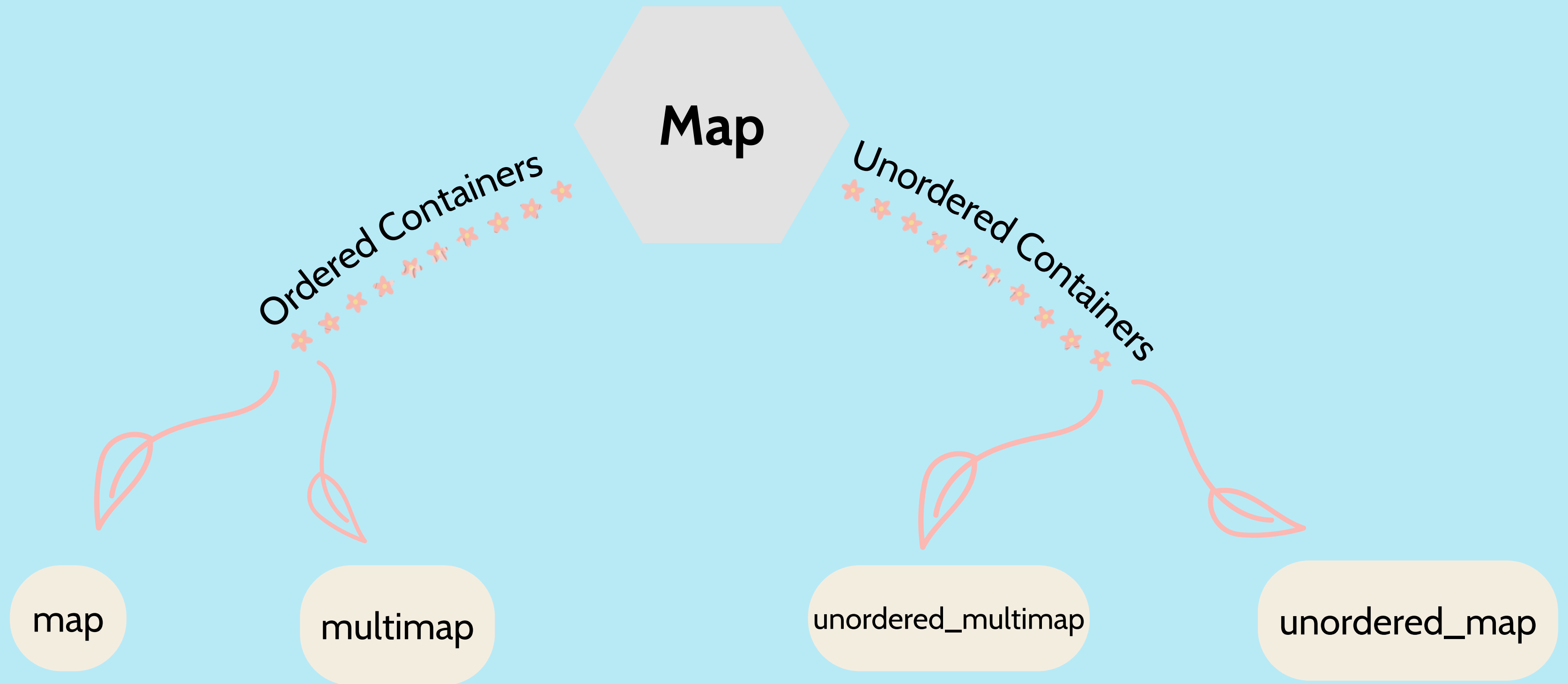
# Có thể sử dụng toán tử gán cho map

```
int main ()
{
    std::map<char,int> first;
    std::map<char,int> second;

    first['x']=8;
    first['y']=16;
    first['z']=32;

    second=first;                // second now chứa ba phần tử của first
    first=map<char,int>();       // and first now không có phần tử nào

    std::cout << "Size of first: " << first.size() << '\n';
    std::cout << "Size of second: " << second.size() << '\n';
    return 0;
}
```



# multimap

- Cũng giống như map, nhưng cho phép lưu trữ nhiều key có cùng giá trị với nhau

- Khai báo: 

```
#include <map>
multimap <{ datatype1, datatype2 }> {tên};
```

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

using namespace std;

int main() {
    multimap<char, int> map16;
    map16.emplace('a', 100);
    map16.emplace('a', 200);
    map16.insert(make_pair('b', 150));
    map16.insert(make_pair('c', 100));
    map16.emplace('d', 1000);
    for (auto x : map16)
        cout << x.first << " " << x.second << endl;
    return 0;
}
```



```
a 100
a 200
b 150
c 100
d 1000
```


# unordered\_map

- Các phần tử được thêm vào không sắp theo thứ tự.
- Giống với map, các phần tử trong unordered\_map không trùng nhau.
- **Khai báo:** `#include <unordered_map>`  
`unordered_map <{ datatype1, datatype2 }> {tên};`

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

using namespace std;

int main() {
    unordered_map<char, int> map16;
    map16.emplace('a', 1000);
    map16.emplace('c', 1500);
    map16.emplace('b', 2000);
    for (auto x : map16)
        cout << x.first << " " << x.second << endl;
    return 0;
}
```



```
a 1000
c 1500
b 2000
```

# Bài tập

1) Cho mảng các số nguyên không âm gồm  $n$  phần tử, thực hiện đếm tần suất xuất hiện của các phần tử và in theo mẫu

Yêu cầu: đầu tiên xuất ra các phần tử theo thứ tự từ nhỏ tới lớn và tần suất của chúng, sau đó bỏ trống một dòng và in ra các phần tử và tần suất của chúng theo thứ tự xuất hiện trong mảng.

Chú ý: Mỗi phần tử chỉ được xuất một lần

2) Cho mảng  $a$  gồm  $n$  phần tử và số nguyên dương  $k$ .

Đếm số lượng cặp số  $a[i], a[j]$  ( $i \neq j$ ) có tổng bằng  $k$ .

# Bài tập

1) Cho mảng các số nguyên không âm gồm n phần tử, thực hiện đếm tần suất xuất hiện của các phần tử và in theo mẫu.

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

int main()
{
    map<int, int> mp;
    int* a;
    int n;
    cin >> n;

    a = new int [n];

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

```
for (auto x : mp)
    cout << x.first << " " << x.second<<endl;

cout << endl;

for (int i = 0; i < n; i++)
    if (mp[a[i]] != 0)
    {
        cout << a[i] << " " << mp.at(a[i]) << endl;
        mp[a[i]] = 0;
    }

return 0;
```

# Bài tập

2) Cho mảng a gồm n phần tử và số nguyên dương k.  
Đếm số lượng cặp số  $a[i], a[j]$  ( $i \neq j$ ) có tổng bằng k.

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

int main()
{
    int n, k;
    cin >> n >> k;
    vector<int> a(n);
    map<int, int> mp;

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

```
int ans = 0;
for (int i = 0; i < n; i++)
{
    ans += mp[k - a[i]];

    if (a[i] * 2 == k)
    {
        ans--;
    }
}

ans /= 2;

cout << ans;

return 0;
```

## Bài tập

3) Cho dãy số  $A[]$  gồm có  $N$  phần tử, có 3 thao tác như sau :

Thao tác 1 : Thêm 1 phần tử  $X$  vào mảng.

Thao tác 2 : Xóa 1 phần tử  $X$  khỏi mảng. Trong trường hợp phần tử  $X$  không xuất hiện trong mảng, sẽ không thực hiện xóa, nếu trong mảng có nhiều phần tử  $X$  thì chỉ xóa đi 1 phần tử  $X$  trong mảng.

Thao tác 3 : Truy vấn xem phần tử  $X$  có xuất hiện trong mảng hay không?

4) Cho mảng  $A[]$  gồm  $n$  phần tử, yêu cầu:

1 : Thêm phần tử  $X$

2 : Xóa mọi giá trị  $X$  khỏi mảng nếu  $X$  tồn tại trong mảng

3 : Tìm phần tử nhỏ nhất trong mảng

4 : Tìm phần tử lớn nhất trong mảng



# Bài tập

3) Cho dãy số  $A[]$  gồm có  $N$  phần tử, có 3 thao tác như sau :

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

using namespace std;

int main()
{
    int n,q,t1,t2;
    cin >> n;// nhập số phần tử trong mảng
    vector<int>a(n);
    multiset<int>b;
    //nhập phần tử trong mảng và đẩy vào set
    for (int x:a)
    {
        cin >> x;
        b.insert(x);
    }
```

```
cin >> q;//nhập số lượng truy vấn

for (int i = 0; i < q; i++)
{
    cin >> t1;//nhập loại truy vấn
    cin >> t2;//nhập X
    if (t1 == 1)
        b.insert(t2);
    if (t1 == 2)
        b.erase(b.find(t2));
    if (t1 == 3)
        if (b.find(t2) != b.end())
            cout << "Yes";
        else
            cout << "No";
}

return 0;
```

# Bài tập

4) Cho mảng  $A[]$  gồm  $n$  phần tử, yêu cầu:

1: Thêm phần tử  $X$

2: Xóa mọi giá trị  $X$  khỏi mảng nếu  $X$  tồn tại trong mảng

3: Tìm phần tử nhỏ nhất trong mảng

4: Tìm phần tử lớn nhất trong mảng

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

using namespace std;

int main()
{
    int n, q, t1, t2;
    cin >> n; // nhập số phần tử trong mảng
    vector<int>a(n);
    multiset<int>b;
    //nhập phần tử trong mảng và đẩy vào set

    for (int x : a)
    {
        cin >> x;
        b.insert(x);
    }
```

```
for (int i = 0; i < q; i++)
{
    cin >> t1; //nhập loại truy vấn
    if (t1 == 1)
    {
        cin >> t2;
        b.insert(t2);
    }
    if (t1 == 2)
    {
        cin >> t2;
        if (b.find(t2) != b.end())
            b.erase(t2);
    }
    if (t1 == 3)
        cout << *b.begin();
    if (t1 == 4)
        cout << *b.rbegin();
}

return 0;
```