## Contents


kích thước và phạm vi các kiểu dữ liệu ..... 2

Chuyên đề String trong C++ ..... 2

Vấn đề 1: Cơ bản ..... 2

Vấn đề 2: Các thao tác với chuỗi qua hàm : ..... 4

1. "reverse" function :đảo ngược một chuỗi: ..... 4

2. Substring : Chuỗi con ..... 4

3.erase: Xóa ký tự ..... 5

4. insert () chèn các ký tự trong chuỗi ở vị trí xác định ..... 5

5. Replace : thay thế ..... 6

6 So sánh 2 chuỗi ..... 6

7. sắp xếp chuỗi ..... 7

STACK C++ ..... 8

QUEUE ..... 9

PRIORITY QUEUE ..... 10

Min heap : ..... 11

priority_queue value_type in C++ STL ..... 11

Vector trong C++ ..... 12

Khai báo vector ..... 13

Một số chức năng liên quan đến việc vector : ..... 13

Vector of Vectors in C++ STL ..... 16

Chuyên đề các hàm sử dụng với mảng ..... 18

1, std :: sort () trong C ++ STL ..... 18

2,Tìm min và max trong một mảng : ..... 20

3, upper_bound () và Lower_bound () c++ ..... 20

4, memset in C++.....21

**kích thước và phạm vi các kiểu dữ liệu**
Bảng dưới đây tóm tắt kích thước và phạm vi đã sửa đổi của các kiểu dữ liệu tích hợp khi kết hợp với các công cụ sửa đổi kiểu:

| DATA TYPE | BYTES | RANGE |
|---|---|---|
| short int | 2 | -32,768 to 32,767 |
| unsigned short int | 2 | 0 to 65,535 |
| unsigned int | 4 | 0 to 4,294,967,295 |
| int | 4 | -2,147,483,648 to 2,147,483,647 |
| long int | 8 | -2,147,483,648 to 2,147,483,647 |
| unsigned long int | 4 | 0 to 4,294,967,295 |
| long long int | 8 | -(2^63) to (2^63)-1 |
| unsigned long long int | 8 | 0 to 18,446,744,073,709,551,615 |
| signed char | 1 | -128 to 127 |
| unsigned char | 1 | 0 to 255 |
| float | 4 | +/- 3.4e +/- 38 (~7 digits) |
| double | 8 | +/- 1.7e +/- 308 (~15 digits) |
| long double | 12 | +/- 1.7e +/- 308 (~15 digits) |
| wchar_t | 2 or 4 | 1 wide character |

**Lưu ý** : Các giá trị trên có thể khác nhau giữa các trình biên dịch. Trong ví dụ trên, chúng ta đã xem xét GCC 64 bit.

**Chuyên đề String trong C++**
**Vấn đề 1: Cơ bản**
1.khai báo ,nhập chuỗi:

```
#include <iostream>
#include <string>
using namespace std;
int main() {
string s1,s2;
cin >> s1;          //hello world
cout <<s1 << endl; //hello
/*getline(cin, s2);//hello world
cout <<s2 << endl;//hello world  */
return 0;
}
```

2. Toán tử + :

Toán tử + có một hạn chế quan trọng. Nó không thể nối các chuỗi *literals* (tức là các chuỗi được viết trực tiếp kiểu hardcode).

```
#include  <iostream>
#include  <string>
using namespace std;
int main() {
string String;
String = "A" + "B"; // error
String = String + "C";
String = "B" + String;
cout << String << endl;
return 0;
}
```

3.chuyển chuỗi thường thành hoa

```
for(int i=0;i<s.length();i++)
      if(s[i]>='a'&&s[i]<='z') s[i]=s[i]-' ';
```

chuyển hoa thành thường:

```
for(int i=0;i<s.length();i++){
      if(s[i]>='A'&&s[i]<='Z') ) s[i]=s[i]+' '
```

4. chuyển chuỗi sang số

```
for(int i=0;i<s.size(),i++){
      sum=sum*10+s[i]-'0';
```

5. đếm số lần xuất hiện của chữ cái

```
int f[26]={0};
      for(int j=i;j<s.size();j++ ){
         f[ s[j]-'a']++;
```

**Vấn đề 2: Các thao tác với chuỗi qua hàm :**

**1. "reverse" function :đảo ngược một chuỗi:**

Link : https://www.geeksforgeeks.org/reverse-a-string-in-c-cpp-different-methods/

```
#include <bits/stdc++.h>
using namespace std;
int main() {
string str = "geeksforgeeks";
reverse(str.begin(), str.end());
cout << str; // skeegrofskeeg
return 0;
}
```

**2. Substring : Chuỗi con**

Link : https://www.geeksforgeeks.org/substring-in-cpp/

```
#include <string.h>
#include <iostream>
using namespace std;
int main() {
string s1 = "Geeks";
string r = s1.substr(1, 3);
cout <<  r; //eek
return 0;
}
```

Vd2:

```
#include <string.h>
#include <iostream>
using namespace std;
int main() {
string s = "dog:cat";
```

```
int pos = s.find(":"); // Find position of ':' using find()
string sub = s.substr(pos + 1);
cout << sub; //cat
return 0;
}
```

### 3.erase: Xóa ký tự

Link : https://www.geeksforgeeks.org/stdstringerase-in-cpp/

Xóa tất cả :

```
#include <iostream>
#include <string>
using namespace std;
int main() {
string str("Hello World!");
str.erase(); //còn có nhiều cách khác
cout << str;
return 0;
}
```

**Cú pháp 2:** Xóa tất cả các ký tự sau vị trí 'pos' : str.erase(1);

Before erase : Hello World!=>After erase : H

**Cú pháp 3:** Xóa tối đa các ký tự len của * this, bắt đầu từ chỉ mục idx.

str.erase(1, 4);

Before erase : Hello World!=>After erase : H World!

**Cú pháp 4:** Xóa ký tự đơn tại vị trí trình lặp.

str.erase(str.begin() + 4);

Before erase : Hello World!=>After erase : Hell World!

### 4. insert () chèn các ký tự trong chuỗi ở vị trí xác định

Link :

https://www.geeksforgeeks.org/stdstringinsert-in-c/

```
#include <iostream>
#include <string>
using namespace std;
void insertDemo(string str1, string str2) {
// Inserts str2 in str1 starting  from 6th index of str1
str1.insert(6, str2);
cout << str1; // Hello GeeksforGeeks World!
```

```
}
int main() {
string str1("Hello World! ");
string str2("GeeksforGeeks ");
insertDemo(str1, str2);
return 0;
}
```

### 5. Replace : thay thế

Link : https://www.geeksforgeeks.org/stdstringreplace-stdstringreplace_if-c/

```
#include <iostream>
using namespace std;
void replaceDemo(string s1, string s2, string s3, string s4) {
// Replaces 7 characters from 0th index by s2
s1.replace(0, 7, s2);
cout << s1 << endl; // Demonstration of replace
s4.replace(0, 3, "Hello ");
cout << s4 << endl; // Hello World !
s4.replace(6, 5, s3, 0, 5);
cout << s4 << endl; //Hello Geeks !
s4.replace(6, 5, "to all", 6);
cout << s4 << endl; // Hello to all !
// Replaces 1 character from 12th  index of s4 with  3 copies of '!'
s4.replace(12, 1, 3, '!');
cout << s4 << endl; // Hello to all!!!!
}
int main() {
string s1 = "Example of replace";
string s2 = "Demonstration";
string s3 = "GeeksforGeeks";
string s4 = "HeyWorld !";
replaceDemo(s1, s2, s3, s4);
return 0;
}
```

### 6 So sánh 2 chuỗi

```
#include <iostream>
```

```
using namespace std;
void relationalOperation(string s1, string s2) {
    if (s1 != s2)
        cout << s1 << " is not equal to "<< s2 << endl;
    if (s1 > s2)
        cout << s1 << " is greater than "<< s2 << endl;
    else
        cout << s2 << " is greater than "<< s1 << endl;
    //661 is not equal to 6601
    //661 is greater than 6601
}
int main() {
    string s1("661");
    string s2("6601");
    relationalOperation(s1, s2);
    return 0;
}
```

### 7. sắp xếp chuỗi

1.sắp sắp các ký tuwk trong một chuỗi
https://www.geeksforgeeks.org/sort-string-characters/

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    string s = "geeksforgeeks";
    sort(s.begin(), s.end());
    cout << s; //eeeefggkkorss
    return 0;
}
```

2.sắp xếp các chuỗi trong một chuỗi
Vd :Hãy sắp đặt lại các phần tử trong mảng sao cho khi nối các số lại với nhau ta được một số lớn nhất. vd :A[] = {54, 546, 548, 60 } =>số lớn nhất là 6054854654.
}

```
#include<bits/stdc++.h>
using namespace std;
bool cmp(string a,string b){
```

```
    string ab=a+b;
    string ba=b+a;
    return (ab>ba);
}
int main(){
    string s[100005];
    int n;cin>>n;
    for(int i=0;i<n;i++) cin>>s[i];
    sort(s,s+n,cmp);
    for(int i=0;i<n;i++) cout<<s[i];
    cout<<endl;
    return 0;
```

**STACK C++**

Link : https://www.geeksforgeeks.org/stack-in-cpp-stl/

Ngăn xếp là một loại bộ điều hợp vùng chứa với kiểu LIFO (Last In First Out) hoạt động, trong đó một phần tử mới được thêm vào ở một đầu và (trên cùng) một phần tử chỉ bị xóa khỏi đầu đó.

Các hàm được liên kết với ngăn xếp là:

empty () - Trả về ngăn xếp có trống không

size () - Trả về kích thước

top () - Trả về tham chiếu vào phần tử cao nhất

push (g) - Thêm phần tử 'g' vào đầu ngăn xếp

pop () - Xóa phần tử trên cùng nhất của ngăn xếp –

⇨ Note : Độ phức tạp về thời gian: O (1)

```
#include <bits/stdc++.h>
using namespace std;
void showstack(stack <int> s) {
    while (!s.empty()) {
        cout<< s.top()<<' ';
        s.pop();
    }  cout << '\n';
}
int main () {
    stack <int> s;
```

```
    s.push(10); s.push(30);
    s.push(20); s.push(5);
    s.push(1); // Stack becomes 10 30 20 5 1(top)
    showstack(s); //1 5 20 30 10
    cout << s.size()<<endl; //5
    cout << s.top()<<endl; //1
    s.pop(); //Stack becomes 10 30 20 5(top)
    showstack(s); //5 20 30 10
    return 0;
}..
```

**QUEUE**

https://www.geeksforgeeks.org/queue-cpp-stl/

Hàng đợi là một loại bộ điều hợp vùng chứa hoạt động theo kiểu sắp xếp trước vào trước (FIFO). Các phần tử được chèn ở phía sau (cuối) và bị xóa khỏi phía trước.

**Các chức năng được hỗ trợ bởi hàng đợi là:**

1. empty () - Trả về hàng đợi có trống hay không.
2. size () - Trả về kích thước
3. queue :: swap () : Trao đổi nội dung của hai hàng đợi nhưng các hàng đợi phải cùng loại, mặc dù kích thước có thể khác nhau.
4. queue :: emplace () : Chèn một phần tử mới vào vùng chứa hàng đợi vào cuối hàng đợi.
5. queue :: front () và queue :: back () Hàm STL - **front** () trả về một tham chiếu đến phần tử đầu tiên. Hàm **back** () trả về một tham chiếu đến phần tử cuối cùng
6. push (g) và pop () - **push** () hàm thêm phần tử 'g' vào cuối hàng đợi. Hàm **pop** () xóa phần tử đầu tiên của hàng đợi.

```
#include <iostream>
#include <queue>
using namespace std;
void showq(queue <int> gq) {
    queue <int> g = gq; //note
    while (!g.empty()) {
        cout<< g.front()<<' ';
        g.pop();
```

```
    }  cout << '\n';
}
int main() {
    queue <int> gquiz;
    gquiz.push(10);
    gquiz.push(20);
    gquiz.push(30);
    showq(gquiz); //10 20 30
    cout <<gquiz.size()<<"\n";//3
    cout << gquiz.front()<<"\n";//10
    cout <<gquiz.back()<<"\n"; //30
    gquiz.pop();
    showq(gquiz);//20 30
    return 0;
}
```

**PRIORITY QUEUE**

Link : https://www.geeksforgeeks.org/priority-queue-in-cpp-stl/?ref=lbp

Hàng đợi ưu tiên là một loại bộ điều hợp vùng chứa, được thiết kế đặc biệt sao cho phần tử đầu tiên của hàng đợi là phần tử lớn nhất trong tất cả các phần tử trong hàng đợi và các phần tử có thứ tự không tăng dần (do đó chúng ta có thể thấy rằng mỗi phần tử của hàng đợi có mức độ ưu tiên { thứ tự cố định}).

Ví dụ :

```
#include <iostream>
#include <queue>
using namespace std;
void showpq(priority_queue <int> gq) {
    priority_queue <int> g = gq;
    while (!g.empty()) {
        cout << g.top()<<' ';
        g.pop();
    }  cout << '\n';
}
int main () {
    priority_queue <int> gquiz;
    gquiz.push(10); gquiz.push(30);
```

```
    gquiz.push(20); gquiz.push(5);
    gquiz.push(1);
    showpq(gquiz); //30 20 10 5 1
    cout  << gquiz.size()<<endl;//5
    cout  << gquiz.top() <<endl; //30
    gquiz.pop();
    showpq(gquiz); //20 10 5 1
    return 0;
}
```

**Min heap :**

```
#include <iostream>
#include <queue>
using namespace std;
void showpq(priority_queue <int, vector<int>, greater<int> > gq) {
    priority_queue <int, vector<int>, greater<int> > g = gq;
    while (!g.empty()) {
        cout << g.top()<<" ";
        g.pop();
    } cout << '\n';
}
int main () {
    priority_queue <int, vector<int>, greater<int> > gquiz;
    gquiz.push(10); gquiz.push(30);
    gquiz.push(20); gquiz.push(5);
    gquiz.push(1);
    showpq(gquiz); //1 5 10 20 30
    cout<< gquiz.size()<<endl ; //5
    cout << gquiz.top()<<endl; //1
    gquiz.pop();
    showpq(gquiz); //5 10 20 30
    return 0;
}
```

**priority_queue value_type in C++ STL**

link : https://www.geeksforgeeks.org/priority_queue-value_type-in-c-stl/

Phương thức **priority_queue :: value_type** là một hàm nội trang trong C ++ STL

đại diện cho kiểu đối tượng được lưu trữ dưới dạng một phần tử trong priority_queue. Nó hoạt động như một từ đồng nghĩa với tham số mẫu.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    priority_queue<int>::value_type AnInt;
    priority_queue<int> q1;
    AnInt = 20;
    q1.push(AnInt);
    AnInt = 30;
    q1.push(AnInt);
    cout << q1.top() << endl; //30
    return 0;
}
```

Ví dụ 2:

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    priority_queue<string>::value_type AString;
    priority_queue<string> q2;
    AString = "geeks for geeks";
    AString = "abc"; q2.push(AString);
    AString = "def"; q2.push(AString);
    AString = "ghi"; q2.push(AString);
    while (!q2.empty()) {
        cout << q2.top()<<' ';
        q2.pop();
    } //ghi def abc
    return 0;
}
```

....

**Vector trong C++**

https://www.geeksforgeeks.org/vector-in-cpp-stl/

Note quan trọng : Thêm lệnh -std=c++11.

- Vectơ giống như mảng động với khả năng tự động thay đổi kích thước khi một phần tử được chèn hoặc xóa, với việc lưu trữ của chúng sẽ được vùng chứa tự động xử lý.
- Vector là một chuỗi các phần tử có cùng kiểu dữ liệu, cũng giống như mảng bình thường trong C++. Vậy thì tại sao phải dùng nó?

+Đầu tiên, vector có thể tự tăng kích thước của nó mỗi khi ta thực hiện thêm một phần tử vào vector.
+Thứ 2, vector có thể tự giải phóng bộ nhớ khi ta thực hiện xong đoạn code và thoát ra khỏi scope chứa vector đó, việc này nhằm tránh rò rỉ bộ nhớ khi ta quên delete[] như con trỏ.
+Thứ ba là vector cung cấp các hàm cần thiết để chúng ta cho thể thao tác với mảng một cách dễ dàng.
-Theo nguyên tắc chung, bạn nên sử dụng:
std::array nếu kích thước cố định tại thời điểm biên dịch
std :: vector là kích thước không cố định tại thời điểm biên dịch

**Khai báo vector**

```
#include <vector>
...
/* Vector 1 chiều */
vector <int> first; /* tạo vector rỗng kiểu dữ liệu int */
vector <int> second (4,100); //tạo vector với 4 phần tử là 100
vector <int> third (second.begin(),second.end()) // lấy từ đầu đến cuối vector second
vector <int> four (third) //copy từ vector third

/*Vector 2 chiều*/
vector < vector <int> > v; /* Tạo vector 2 chiều rỗng */
vector < vector <int> > v (5, 10) ; /* khai báo vector 5×10 */
vector < vector <int> > v (5) ; /* khai báo 5 vector 1 chiều rỗng*/
vector < vector <int> > v (5, vector <int> (10,1) ) ; //khai báo vector 5*10 với các phần tử khởi tạo giá trị là 1
```

**Một số chức năng liên quan đến việc vector :**

Vòng lặp

1. begin () - Trả về một trình lặp trỏ đến phần tử đầu tiên trong vectơ
2. end () - Trả về một trình lặp trỏ đến phần tử lý thuyết theo sau phần tử cuối

cùng trong vectơ
3. rbegin () ,rend ()
4. cbegin () ,cend ()
5. crbegin () ,crend ()

Note : front :truy cập phần tử đầu tiên
back :truy cập phần tử cuối cùng
end/cend : trả về một trình lặp về cuối
begin/cbegin : trả về trình lặp về đầu

Modifiers: Bổ ngữ:

1. assign() :gán () Gán giá trị mới cho các phần tử vectơ bằng cách thay thế các giá trị cũ
2. push_back () -đẩy các phần tử vào một vector từ phía sau
3. pop_back () - xóa các phần tử khỏi một vector từ phía sau.
4. insert () - chèn các phần tử mới trước phần tử ở vị trí được chỉ định
5. erase () - xóa các phần tử khỏi vùng chứa khỏi vị trí hoặc phạm vi được chỉ định.
6. swap () - đổi nội dung của một vectơ này với một vectơ khác cùng loại. Kích thước có thể khác nhau.
7. clear () loại bỏ tất cả các phần tử của vùng chứa vector
8. emplace () - mở rộng vùng chứa bằng cách chèn phần tử mới vào vị trí
9. emplace_back () - chèn một phần tử mới vào vùng chứa vectơ, vào cuối vectơ

```
#include <bits/stdc++.h>
#include <vector>
using namespace std;
int main() {
    vector<int> v;
    v.assign(5, 10); // fill the array with 10 five times
    for (int i = 0; i < v.size(); i++)
        cout << v[i] << " "; //10 10 10 10 10
    v.push_back(15); // inserts 15 to the last position
    int n = v.size();
    cout << "\n " << v[n - 1]; //15
    v.pop_back(); // removes last element
    for (int i = 0; i < v.size(); i++)
```

```
        cout << v[i] << " "; //10 10 10 10 10
    v.insert(v.begin(), 5); // inserts 5 at the beginning
    cout << "\n " << v[0]; //5
    v.erase(v.begin()); // removes the first element
    cout << "\n " << v[0]; //10
    v.emplace(v.begin(), 5); // inserts at the beginning
    cout << "\n " << v[0]; //5
    v.emplace_back(20); // Inserts 20 at the end
    n = v.size();
    cout << "\n " << v[n - 1]; //20
    v.clear();  // erases the vector
    cout << "\n " << v.size(); //0

    // two vector to perform swap
    //v1.swap(v2); // Swaps v1 and v2
}
```

Element access: Quyền truy cập phần tử:

1. reference operator [g] :toán tử tham chiếu [g] - Trả về một tham chiếu đến phần tử ở vị trí 'g' trong vectơ
2. at (g) - Trả về một tham chiếu đến phần tử ở vị trí 'g' trong vectơ
3. front () - Trả về phần tử đầu tiên trong vectơ
4. back () - Trả về phần tử cuối cùng trong vectơ
5. data () - Trả về một con trỏ trực tiếp đến mảng bộ nhớ được vector sử dụng bên trong để lưu trữ các phần tử thuộc sở hữu của nó.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    vector<int> g1;
    for (int i = 1; i <= 10; i++) g1.push_back(i * 10);
    cout << "\n " << g1[2]; //30
    cout << "\n " << g1.at(4); //50
    cout << "\n" << g1.front(); //10//*g1.begin()
    cout << "\n " << g1.back(); //100//*(g1.end()-1)
    int* pos = g1.data();
    cout << "\n" << *pos; //10
```

```
    return 0;
}
```

Capacity : **Sức chứa**

1. size () - Trả về số phần tử trong vector.
2. resize (n) - Thay đổi kích thước vùng chứa để nó chứa các phần tử 'n'.
3. empty() - Trả về liệu vùng chứa có trống hay không.
4. shrink_to_fit() ) - Giảm dung lượng của vùng chứa để phù hợp với kích thước của nó và hủy tất cả các phần tử vượt quá dung lượng.
5. Reserve () - Yêu cầu vectơ công suất ít nhất đủ để chứa n phần tử.

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
        vector<int> g1;
        for (int i = 1; i <= 5; i++) g1.push_back(i);
        cout  << g1.size()<<endl; //5
        g1.resize(4); // resizes the vector size to 4
        cout << g1.size()<<endl; //4
        if (g1.empty() == false)
        cout << "no empty"<<endl; //no empty
        else cout << "empty"<<endl;
        g1.shrink_to_fit();
        for (auto it = g1.begin(); it != g1.end(); it++)
        cout << *it << " "; //1 2 3 4
        return 0;
}
```

**Vector of Vectors in C++ STL**

Vectơ của Vectơ là một vectơ hai chiều với số hàng thay đổi trong đó mỗi hàng là vectơ. Mỗi chỉ mục của vectơ lưu trữ một vectơ có thể được duyệt và truy cập bằng các trình vòng lặp . Nó tương tự như Mảng Vectơ nhưng có các thuộc tính động.

Cú pháp:

vector <vector <kiểu_dữ liệu>> vec;

Chèn trong Vectơ của Vectơ

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<vector<int> > vec; // Initializing
    int num = 10; // Elements to insert in column
    for (int i = 0; i < 4; i++) {
        vector<int> v1;
        for (int j = 0; j < 5; j++) {
            v1.push_back(num);
            num += 5;
        }
        vec.push_back(v1);
    }
    for (int i = 0; i < vec.size(); i++) {
        for (int j = 0; j < vec[i].size(); j++)
            cout << vec[i][j] << " ";
        cout << endl;
    }
    return 0;
    /*10 15 20 25 30
    35 40 45 50 55
    60 65 70 75 80
    85 90 95 100 105*/
}
```

Loại bỏ hoặc Xóa trong một Vectơ Vectơ

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
    vector<vector<int> > vec{ { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
    // Removing elements from the  last row of the vector
    vec[2].pop_back();
```

```
    vec[1].pop_back();
    for (int i = 0; i < 3; i++) {
        for ( auto it = vec[i].begin(); it != vec[i].end(); it++)
            cout << *it << " "; //1 2 3
        cout << endl;           //4 5
    }                           //7 8
    return 0;

}
```

Thư viện mẫu chuẩn (STL)
https://www.geeksforgeeks.org/the-c-standard-template-library-stl/
https://www.geeksforgeeks.org/cpp-stl-tutorial/

**Chuyên đề các hàm sử dụng với mảng**
**1, std :: sort () trong C ++ STL**
link : https://www.geeksforgeeks.org/sort-c-stl/?ref=lbp

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int arr[] = {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
    sort(arr, arr+n); // string :    sort(s.begin(), s.end() );
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
        //0 1 2 3 4 5 6 7 8 9
    return 0;
}
```

Ví dụ 2: Làm thế nào để sắp xếp theo thứ tự giảm dần?

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int arr[] = {1, 5, 8, 9, 6, 7, 3, 4, 2, 0};
    int n = sizeof(arr)/sizeof(arr[0]);
```

```
    sort(arr, arr+n, greater<int>() );
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    //9 8 7 6 5 4 3 2 1 0
    return 0;
}
```

Ví dụ 3 : Làm thế nào để sắp xếp theo thứ tự cụ thể?

```
#include<bits/stdc++.h>
using namespace std;
struct Interval {
    int start, end; };
// Compares two intervals according to staring times.
bool compareInterval(Interval i1, Interval i2) {
    return (i1.start < i2.start); }
int main() {
    Interval arr[] = { {6,8}, {1,9}, {2,4}, {4,7} };
    int n = sizeof(arr)/sizeof(arr[0]);
    sort(arr, arr+n, compareInterval);
    for (int i=0; i<n; i++)
    cout << "[" << arr[i].start << "," << arr[i].end << "] ";
    //[1,9] [2,4] [4,7] [6,8
    return 0;
}
```

Ví dụ 3 : sắp sếp mảng 2 chiều theo hàng :

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int n=3,m=3;
    int a[3][3]={9,8,7,6,5,4,3,2,1};
    for(int i=0;i<n;i++)
        sort(a[i],a[i]+m /*,greater<int>()*/);
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++)
            cout<<a[i][j]<<' ';
        cout<<endl;
    }//7 8 9**4 5 6**1 2 3
```

```
    return 0;
}
```

**2,Tìm min và max trong một mảng :**

Tìm Min (*min_element) còn max (*max_element)

```
#include <bits/stdc++.h>
using namespace std;
int main(){
    int n; cin>>n;
    int a[n];
    for(int i=0;i<n;i++) cin>>a[i];
    //1 2 3 4 5
    int x= *min_element(a,a+n);
    cout<<x<<endl;
    return 0;
}
```

**3, upper_bound () và Lower_bound () c++**

https://www.geeksforgeeks.org/upper_bound-in-cpp/

upper_bound () là một hàm thư viện chuẩn trong C ++ được định nghĩa trong tiêu đề . Nó trả về một trình lặp trỏ đến phần tử đầu tiên trong phạm vi [đầu tiên, cuối cùng) lớn hơn giá trị hoặc cuối cùng nếu không tìm thấy phần tử như vậy. Các phần tử trong phạm vi phải đã được sắp xếp hoặc ít nhất được phân vùng theo val. Phương thức **Lower_bound** () trong C ++ được sử dụng để trả về một trình lặp trỏ đến phần tử đầu tiên trong phạm vi [đầu tiên, cuối cùng) có giá trị không nhỏ hơn val. Điều này có nghĩa là hàm trả về chỉ số của số nhỏ nhất tiếp theo chỉ lớn hơn hoặc bằng số đó. Nếu có nhiều giá trị bằng val, low_bound () trả về chỉ mục của giá trị đó đầu tiên.

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    int arr[] = { 10, 20, 30, 40, 50 };
    vector<int> v(arr,arr+5); //10 20 30 40 50
    // using upper_bound
    int upper = upper_bound(arr, arr+5, 30) - arr;
    int lower = lower_bound(arr, arr+5, 30) - arr;
```

```
    cout << (upper)<<endl; //3 <=> 40>30
    cout << (lower); //2 <=> 30>=30
    return 0;
}
```

**4, memset in C++**

https://www.geeksforgeeks.org/memset-in-cpp/

```
#include <cstring>
#include <iostream>
using namespace std;
int main() {
    char str[] = "geeksforgeeks";
    memset(str, 't', sizeof(str));
    cout << str<<endl;; //ttttttttttttt
    int a[5];
    memset(a, -1, sizeof(a)); //note 0 or -1
    for (int i = 0; i < 5; i++)
        cout << a[i] << " "; //-1 -1 -1 -1 -1
    cout << endl;
    return 0;
}
```