5. Function (Hàm)

Định nghĩa của hàm:

```
type name ( argument1, argument2, ...) {
  statement
}
```

Trong đó:

- `type` là kiểu dữ liệu trả về của hàm
- `name` là tên gọi của hàm: lưu ý tên hàm không được trùng với các từ khóa đã được quy định trong C++
- `arguments` là các tham số truyền vào, với số lượng không hạn chế số lượng cũng như kiểu dữ liệu
- `statement` là thân của hàm. Hàm có thể một lệnh hoặc nhiều lệnh kết hợp

Ví dụ 1:

```
#include <iostream>
using namespace std;
int addition (int a, int b)
{
    return a+b;
}

int main ()
{
  int z;
  z = addition (5,3);
  cout << "The result is " << z;
  return 0;
}

output:  The result is 8
```

Ví dụ 2:

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

```
string add(string a, string b)
{
    return a+b;
}
int main()
{
    string a = "Toan";
    string b = "KX";
    string c = add(a,b);
    cout<<c;
    reuturn 0;
}
output: ToanKX
```

Hàm void(): hàm không kiểu

- Là hàm không có câu lệnh return: Hàm này thường được dùng để hiển thị, hoặc cập nhật các giá trị trong CP

  Lưu ý: Làm chỉ dùng để hiển thị hoặc cập nhật không nên sử dụng để tính toán trong hàm void

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

void my_function()
{
  cout << "I'm a function!";
}

int main ()
{
  my_function ();
  return 0;
}
output:  I'm a function!
```

Hàm - Tham số giá trị và tham số biến trong C++

Mục đích của hàm này là có thể sử dụng để cập nhật giá trị các biến, xem qua 2 ví dụ dưới đây

```
#include <iostream>
using namespace std;
void duplicate (int a, int b, int c)
{
  a*=2;
  b*=2;
  c*=2;
```

```
}

int main ()
{
  int x=1, y=3, z=7;
  duplicate (x, y, z);
  cout << "x=" << x << ", y=" << y << ", z=" << z;
  return 0;
}

output: x=1, y=3, z=7
```

Như chúng ta thấy hàm khi không truyền tham trị & thì các giá trị x,y,z không thay đổi

```
#include <iostream>
using namespace std;
void duplicate (int &a, int &b, int &c)
{
  a*=2;
  b*=2;
  c*=2;
}

int main ()
{
  int x=1, y=3, z=7;
  duplicate (x, y, z);
  cout << "x=" << x << ", y=" << y << ", z=" << z;
  return 0;
}
output:  x=2, y=6, z=14
```

Tóm lại khi dùng tham trị & sẽ cập nhật giá trị của biến sau khi chạy qua hàm, nếu không dùng & sẽ không cập nhật mà chỉ tính toán

Hàm - Giá trị mặc định của tham số trong C++

Khi định nghĩa một hàm chúng ta có thể chỉ định những giá trị mặc định sẽ được truyền cho các đối số trong trường hợp chúng bị bỏ qua khi hàm được gọi. Để làm việc này đơn giản chỉ cần gán một giá trị cho đối số khi khai báo hàm. Nếu giá trị của tham số đó vẫn được chỉ định khi gọi hàm thì giá trị mặc định sẽ bị bỏ qua.

```
// default values in functions
#include <iostream>
using namespace std;
int divide (int a, int b=2)
{
```

```
  int r;
  r=a/b;
  return (r);
}

int main ()
{
  cout << divide (12); // divide(12,2)
  cout << endl;
  cout << divide (20,4);// divide(20,4)
  return 0;
}
output: 6
        5
```

Các hàm trong thư viện hay sử dụng trong tính toán

- `max(a,b)` trả về giá trị lớn nhất của 2 biến:

  a = 4, b = 5 => max(a,b) = 5

- `min(a,b)` trả về giá trị nhỏ nhất của 2 biến

  a = 4, b = 5 => min(a,b) = 4

- `swap(a,b)` hàm đổi chỗ giá trị của 2 biến

  a = 4, b = 5 => swap(a,b) => a = 5, b=4

- `pow(a,b)` hàm tính lũy thừa với a là cơ số, b là số lũy thừa

  a = 4, b = 5 => pow(a,b) = 4^5

- `sqrt(x)` hàm tính toán căn bậc 2

  a = 9 => sqrt(a) = 3

Ngoài ra còn có các hàm tính toán có thể gặp trong CP các bạn có thể tham khảo tại link dưới đây

- Hàm acos() trong C / C++
- Hàm asin() trong C / C++
- Hàm atan() trong C / C++
- Hàm cbrt() trong C / C++
- Hàm ceil() trong C / C++
- Hàm abs() trong C / C++
- Hàm cos() trong C / C++

- Hàm exp() trong C / C++
- Hàm fmax() và fmin() trong C / C++
- Hàm log() trong C / C++
- Hàm pow() trong C / C++
- Hàm round() trong C / C++
- Hàm sqrt() trong C / C++
- Hàm sin() trong C / C++

Một số tài liệu tham khảo:

https://www.cplusplus.com/doc/tutorial/functions

https://www.tutorialspoint.com/cplusplus/cpp_functions.htm

Kiểu dữ liệu do người dùng tự định nghĩa trong C++

Tự định nghĩa các kiểu dữ liệu (typedef)

C++ cho phép chúng ta định nghĩa các kiểu dữ liệu của riêng mình dựa trên các kiểu dữ liệu đã có. Để có thể làm việc đó chúng ta sẽ sử dụng từ khoá `typedef` , dạng thức như sau:

```
typedef existing_type new_type_name;
```

Trong đó: `existing_type` là một kiểu dữ liệu đã được định nghĩa và `new_type_name` là tên của kiểu dữ liệu mới

ví dụ:

```
typedef string S;
typedef pair<int,int> pii;
typedef char C;
typedef unsigned int WORD;
typedef char * string_t;
typedef char field [50];
```

sau khi định nghĩa ta hoàn toàn có thể sử dụng chúng một cách hợp lệ

```
S s;
pii p;
C achar, anotherchar, *ptchar1;
```

```
WORD myword;
string_t ptchar2;
field name;
```

Union

Union cho phép một phần bộ nhớ có thể được truy xuất dưới dạng nhiều kiểu dữ liệu khác nhau mặc dù tất cả chúng đều nằm cùng một vị trí trong bộ nhớ. Phần khai báo và sử dụng nó tương tự với cấu trúc nhưng chức năng thì khác hoàn toàn:

```
union model_name {
  type1 element1;
  type2 element2;
  type3 element3;
  .
  .
} object_name;
```

Tất cả các phần tử của union đều chiếm cùng một chỗ trong bộ nhớ. Kích thước của nó là kích thước của phần tử lớn nhất. Ví dụ:

```
union mytypes_t {
  char c;
  int i;
  float f;
} mytypes;
```

Định nghĩa 3 phần tử

```
mytypes.c
mytypes.i
mytypes.f
```

mỗi phần tử có một kiểu dữ liệu khác nhau. Nhưng vì tất cả chúng đều nằm cùng một chỗ trong bộ nhớ nên bất kì sự thay đổi nào đối với một phần tử sẽ ảnh hưởng tới tất cả các thành phần còn lại.

Struct

```
Cú pháp:

struct <structure_name > {
    <data_type member_name_1>;
    <data_type member_name_i>;
```

```
    ...
    } <structure_variable1>,<structure_variable2>,...;
```

Trong đó:

```
structure_name : Tên cấu trúc

data_type member_name_i : data_type là kiểu dữ liệu của các thành phần trực thuộc structure.

structure_variable: Tên biến cấu trúc
```

Ví dụ 1:

```
#include<iostream>
using namespace std;
struct toado
{
    int x = 5;
    int y = 10;
}a;
int main()
{
    cout<<"x:"<<a.x<<endl;
    cout<<"y:"<<a.y<<endl;
}
```

Bài tập áp dụng

Cho hệ gồm N hành động. Mỗi hành động được biểu diễn như một bộ đôi <S_i, F_i> tương ứng với thời gian bắt đầu và thời gian kết thúc của mỗi hành động. Hãy tìm phương án thực hiện nhiều nhất các hành động được thực hiện bởi một máy hoặc một người sao cho hệ không xảy ra mâu thuẫn.

Input:

- Dòng đầu tiên đưa vào số lượng bộ test T.
- Những dòng kế tiếp đưa vào các bộ test. Mỗi bộ test gồm 3 dòng: dòng thứ nhất đưa vào số lượng hành động N; dòng tiếp theo đưa vào N số S_i tương ứng với thời gian bắt đầu mỗi hành động; dòng cuối cùng đưa vào N số F_i tương ứng với thời gian kết thúc mỗi hành động; các số được viết cách nhau một vài khoảng trống.
- T, N, S_i, F_i thỏa mãn ràng buộc: $1 \le T \le 100$; $1 \le N, F_i, S_i \le 1000$.

Output:

- Đưa số lượng lớn nhất các hành động có thể được thực thi bởi một máy hoặc một người.

Ví dụ:

Input	Output
1 6 1 3 0 5 8 5 2 4 6 7 9 9	4

- Phân tích:

  Bài này là bài tiêu biểu sử dụng giải thuật tham lam nhưng mình sẽ giới thiệu ở đây bằng phương pháp áp dụng struct để giải

  Tham lam ở đây chính là ta sắp xếp 2 dãy theo chiều tăng dần của công việc kết thúc và chỉ cần kiếm tra xem công việc đó thời gian bắt đầu có nhỏ hơn thời gian kết thúc hay không.

  - Nếu thời gian bắt đầu lớn hơn hoặc bằng thời gian kết thúc của người trước đó thì ta tăng tăng thêm được 1 phương án và cập nhật lại
  - Nếu ngược lại thì ta xét tiếp người bắt đầu ở vị trí tiếp theo

Do phần này là phần C++ cơ bản nên mình sẽ không đi sâu vào phân tích thuật toán mà chỉ giới thiệu cách giải quyết áp dụng struct nên nếu thấy khó hiểu thì các bạn có thể xem lại phần phân tích chi tiết ở các chương giải thuật tiếp theo.

```
#include<bits/stdc++.h>
using namespace std;
int n;
struct data
```

```
{
    int fi, se;
};
data h[1005];
bool cmp(data a, data b)
{
    return a.se < b.se;
}
int main()
{
    ios_base::sync_with_stdio(0);
    int t;
    cin>>t;
    while(t--)
    {
        cin>>n;
        for(int i=0;i<n;i++)
            cin>>h[i].fi;
        for(int i=0;i<n;i++)
            cin>>h[i].se;

        sort(h,h+n,cmp);
        int ans=1, i=0;
        for(int j=1;j<n;j++)
        {
            if(h[j].fi >= h[i].se)
            {
                ans++;
                i=j;
            }
        }
        cout<<ans<<endl;
    }
}
```

Ngoài ra bài này còn có thể sử dụng kiểu cấu trúc pair mình sẽ giới thiệu ở phần các cấu trúc nâng cao.

update....

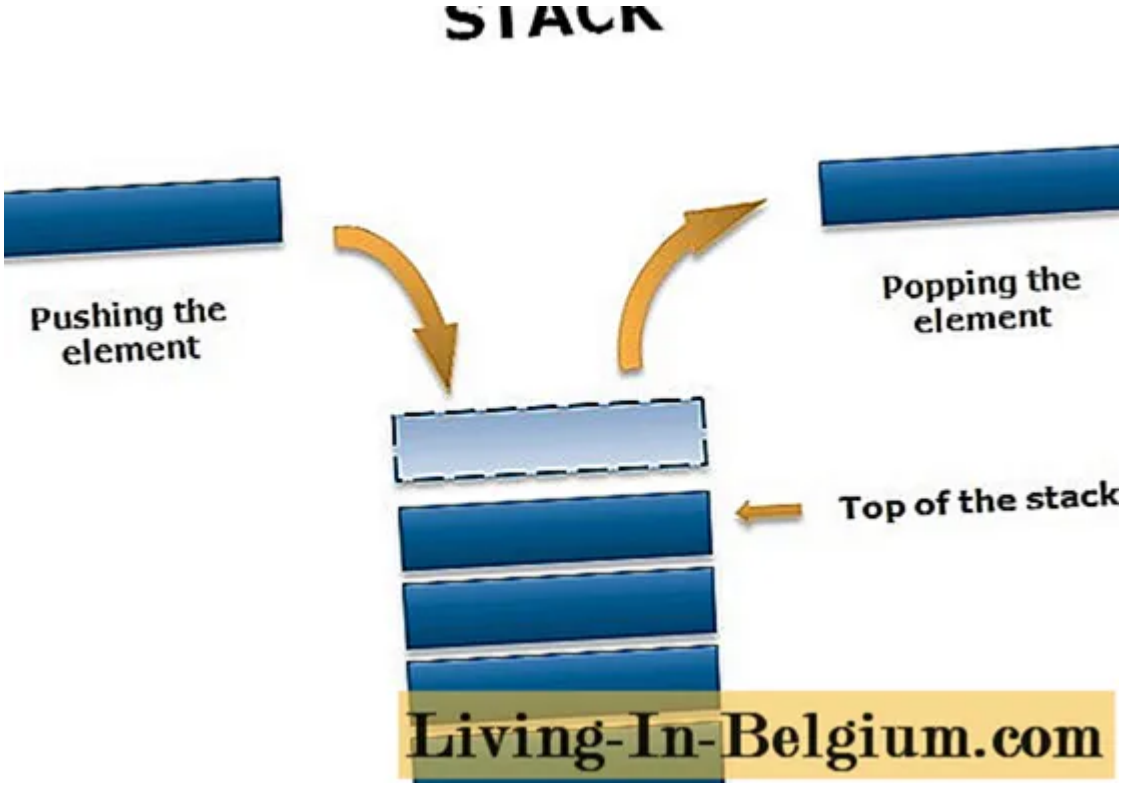
STACK, QUEUE

Giới thiệu:

Stack, Queue là các cấu trúc dữ liệu hay dùng và thường gặp trong các bài toán, chúng ta có thể xây dựng stack, queue bằng Linklist hoặc có thể cài đặt bằng mảng 1 chiều. Nhưng trong bài này mình sẽ hướng dẫn các bạn áp dụng thư viện stack, queue trong giải thuật

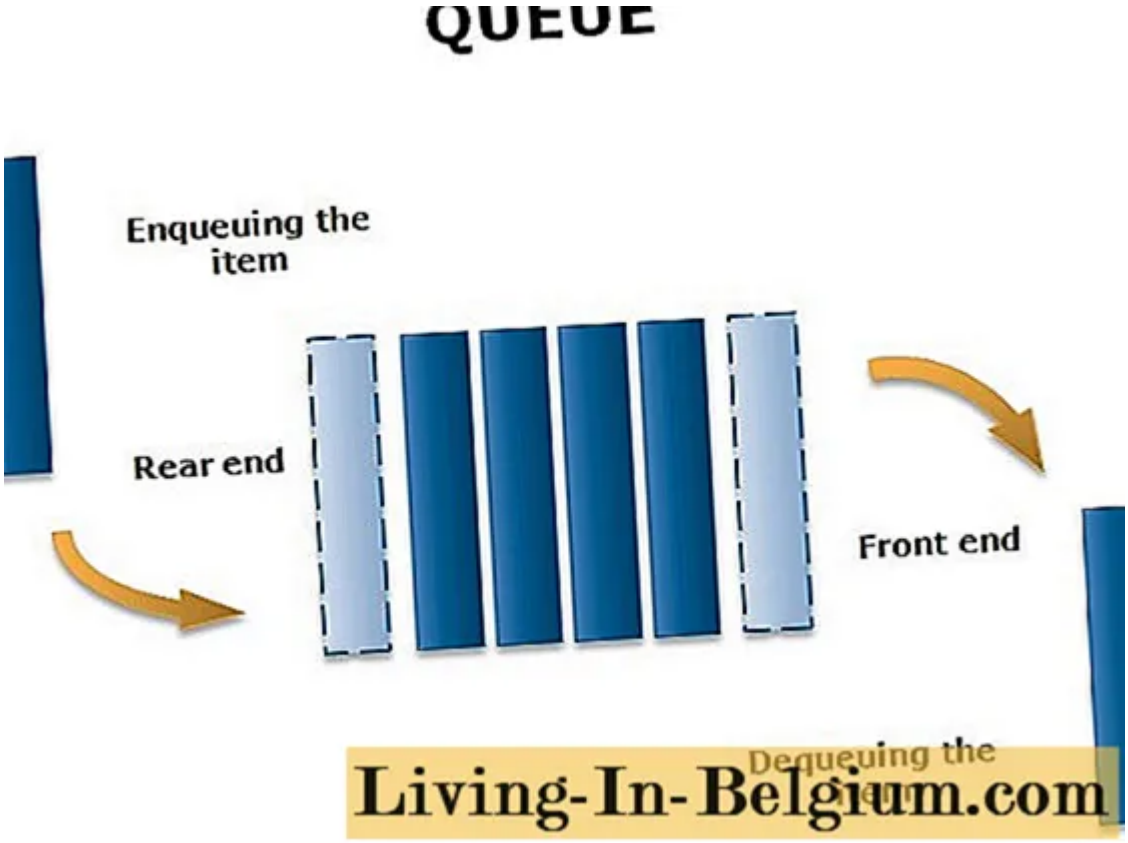
1. Stack

Stack là cấu trúc ngăn xếp, cấu trúc là FIFO (First in first out) có nghĩa là vào trước ra trước, các bạn có thể hình dung stack như một đống sách xếp chồng lên nhau. Việc thêm một phần tử vào stack cũng như đặt một cuốn sách lên đỉnh đống sách, việc lấy một phần tử trong stack ta cũng hình dung là lấy cuốn sách đầu tiên của đống sách đó ra.



2. Queue

Queue là cấu trúc hàng đợi dạng FILO (first in last out) vào trước ra sau, các bạn có thể hình dung queue như một đống sách nằm ngang. Khi thêm một phần tử vào queue thì ta đẩy một cuốn sách vào từ bên trái. Khi lấy một phần tử thì nó sẽ là cuốn sách ở đầu bên còn lại



Cú pháp trong stack

```
#include<iostream>
#include<stack>
using namespace std;
int main()
{
    stack<int> st; // Khai báo stack với các kiểu trả về có thể là int, long, long long, char, st
    cout<<st.size()<<endl;; //Trả về số lượng phần tử trong stack
    st.push(1); // Thêm một phần tử vào đầu stack
    cout<<st.top()<<endl;;// in ra phần tử đầu stack
    st.pop(); // xóa phần tử trong stack
}
```

Bài tập áp dụng

Bài tập.

Cho một **string**, nhiệm vụ của bạn là in chuỗi đảo ngược của **string** đó ra màn hình bằng cách dùng **stack**.

Ví dụ:

- Test mẫu 1:

Input	Output
codelearn	nraeledoc

Với `s = "codelearn"` thì `reverseString(s) = "nraeledoc"`.

- Test mẫu 2:

Input	Output
abcd	dcba

Với `s = "abcd"` thì `reverseString(s) = "dcba"`.

Hướng dẫn: áp dụng cấu trúc dữ liệu stack đẩy toàn bộ các kí tự trong string vào ngăn xếp khi đó ta chỉ cần lấy ra lần lượt các phần tử của stack ta được output như đề bài

```
#include<iostream>
#include<stack>
using namespace std;
int main()
{
    stack<char> st;
    string s;
    cin>>s;

    for(int i=0;i<s.size();i++)
        st.push(s[i]);

    while(!st.empty())  //Kiểm tra stack có rỗng hay không
    {
        cout<<st.top();
        st.pop();
    }
}
```

Bài tập.

Nhập vào một số nguyên dương `n`.

Hãy chuyển `n` thành mã nhị phân và in chuỗi đó ra màn hình.

Ví dụ:

- Test mẫu 1:

Input	OutPut
13	1101

Với `n = 13` thì `stackBin(n) ="1101"`.
Giải thích: $(1101)_2 = 13$.

- Test mẫu 2:

Input	Output
3	11

Với `n = 3` thì `stackBin(n) ="11"`.

Phân tích:

Để chuyển một số nguyên dương thành số nhị phân ta làm như sau:

Trong khi `n > 0` thì ta lấy `n` chia cho `2` và lưu số dư của phép chia đó lại. Cứ làm vậy đến khi nào `n = 0` thì dừng lại.

Kết quả cần tìm chính là danh sách các số dư được đọc ngược lại, ta sẽ nghĩ ngay đến việc sùng **stack** để lưu các số dư này, sau đó chỉ cần in ra **stack** là được.

```
#include<iostream>
#include<stack>

using namespace std;

int main(){
    stack<char> st;
    int n;
    cin >> n;
    while(n > 0){
        st.push(n%2+'0');
        n /= 2;
    }
```

```
    while(!st.empty()){
        cout << st.top();
        st.pop();
    }
    return 0;
}
```

Chúng ta sẽ đi sâu hơn ở các bài chuyên giải thuật về stack ở phần sau.

Cú pháp trong queue

```
#include<iostream>
#include<queue>
using namespace std;
int main()
{
    queue<int> q; //Khai báo queue, các kiểu trả về có thể là int, long, long long, float, char,
    cout<<q.size()<<endl;// in ra số lượng phần tử có trong hàng đợi
    q.push(1);//thêm phần tử 1 vào hàng đợi
    q.push(2);//thêm phần tử 2 vào hàng đợi
    q.push(3);//thêm phần tử 2 vào hàng đợi

    // các bạn có thể hình dung queue hiện tại  3 2 1
    cout<<q.front()<<endl;  //in ra phần tử đầu ra của queue

    q.pop(); //bỏ đi phần tử đầu ra của queue

    //in ra toàn bộ các phần tử trong queue còn lại

    while(!q.empty())
    {
        cout<<q.front()<<" ";
        q.pop();
    }
}
```

Bài tập.

Nhập vào một số nguyên dương `n`, tiếp theo nhập `n` số nguyên lần lượt là các phần tử của **queue**. Cuối cùng nhập vào một số tự nhiên `k`.

Một phép biến đổi sẽ dịch chuyển phần tử đầu tiên ra sau phần tử cuối cùng của dãy, hãy trả về dãy sau khi đã biến đổi `k` lần. In tất cả các phần tử ra màn hình, sau mỗi phần tử có đúng một khoảng trắng.

Ví dụ:

- Test mẫu 1:

Input	Output
4 1 2 3 4 1	2 3 4 1

Với `queue = [1, 2, 3, 4]` và `k = 1`, thì kết quả mong muốn là: `"2 3 4 1 ".`

- Test mẫu 2:

Input	Output
3 1 3 2 2	2 1 3

Với `queue = [1, 3, 2]` và `k = 2,`
thì kết quả mong muốn là `"2 1 3 ".`

Cách làm: Áp dụng cấu trúc dữ liệu queue

Đầu tiên ta đẩy toàn bộ input vào queue. Với k lần biến đổi ta thực hiện k bước, mỗi bước ta lấy phần tử phía bên phải của hàng đợi (hình dung quyển sách nằm ngang) loại bỏ phần tử đó và push lại, khi đó phần tử bên phải nhất của hàng sách sẽ là phần tử bên trái đầu tiên của hàng sách mới cứ tiếp tục như vậy cho đến k lần rồi in toàn bộ các phần tử trong queue

```
#include<iostream>
#include<queue>

using namespace std;

int main(){
    queue<int> q;
    int n, k, temp;
    cin >> n;
    for (int i = 0; i < n; i++){
        cin >> temp;
```

```
        q.push(temp);
    }
    cin >> k;
    for (int i = 0; i < k; i++){
        int x = q.front();
        q.pop();
        q.push(x);
    }
    while (!q.empty()){
        cout << q.front() << " ";
        q.pop();
    }

    return 0;
}
```

PIORITY QUEUE

Mình sẽ giới thiệu thêm 1 cấu trúc nâng cao về priority queue. Cấu trúc này được xây dựng từ cấu trúc dữ liệu vun đống 'heap' nhưng dùng để xây dựng hàng đợi ưu tiên giúp ta có thể lấy ra phần tử nhỏ nhất/ lớn nhất của một dãy số chỉ với độ phức tạp O(log(n))

```
#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> pq;
    pq.push(10);
    pq.push(30);
    pq.push(20);
    pq.push(5);
    pq.push(1);

    cout << "Hien thi priority queue: ";
    showpq(pq);
```

```
    cout << "pq.size() : " << pq.size()<<endl;
    cout << "pq.top() : " << pq.top()<<endl;

    cout << "pq.pop() : ";
    pq.pop();
    showpq(pq);

    return 0;
}


output:

Hien thi priority queue: 30 20 10 5 1
pq.size() : 5
pq.top() : 30
pq.pop() : 20 10 5 1
```

Nếu muốn tạo ra hàng đợi ưu tiên với phần tử đầu tiên là nhỏ nhất ta làm như sau:

```
#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> > pq;
    pq.push(10);
    pq.push(30);
    pq.push(20);
    pq.push(5);
    pq.push(1);

    cout << "Hien thi priority queue: ";
    showpq(pq);
```

```
    cout << "pq.size() : " << pq.size()<<endl;
    cout << "pq.top() : " << pq.top()<<endl;

    cout << "pq.pop() : ";
    pq.pop();
    showpq(pq);

    return 0;
}


output:

Hien thi priority queue: 1 5 10 20 30
pq.size() : 5
pq.top() : 1
pq.pop() : 5 10 20 30
```

update...

CÁC CẤU TRÚC DỮ LIỆU NÂNG CAO

Mình xin giới thiệu đến các bạn các cấu trúc dữ liệu nâng cao thường sử dụng trong lập trình thi đấu. Các phần mình trình bày dưới đây đều được sử dụng trong bộ thư viên STL của C++. Nếu bạn nào có nhu cầu tìm hiểu chi tiết để xây dựng từng cấu trúc thì rất tốt nhưng đó sẽ tốn rất nhiều thời gian của các bạn vì đơn giản Việc sử dụng thành thạo STL sẽ là rất quan trọng nếu các bạn có ý định tham gia các kì thi như Olympic Tin Học, hay ACM. “STL sẽ nối dài khả năng lập trình của các bạn” (trích lời thầy Lê Minh Hoàng)

VECTOR (Mảng động)

vector là một cấu trúc nâng cao của mảng nên nó sẽ có ưu điểm là linh hoạt hơn nhiều so với mảng và 1 số điểm nổi trội sau đây

- Bạn không cần phải khai báo kích thước của mảng ví dụ int A[100]..., vector có thể tự động nâng kích thước lên.
- Nếu bạn thêm 1 phần tử vào vector đã đầy rồi, thì vector sẽ tự động tăng kích thước của nó lên để dành chỗ cho giá trị mới này.- Vector còn có thể cho bạn biết số lượng các phần tử mà bạn đang lưu trong nó.
- Dùng số phần tử âm vẫn được trong vector ví dụ A-10], A[-3], rất tiện trong việc cài đặt các giải thuật khác.

Ví dụ

```
#include <vector>
...
/* Vector 1 chiều */
/* tạo vector rỗng kiểu dữ liệu int */
vector <int> first;
//tạo vector với 4 phần tử là 100
vector <int> second (4,100);
// lấy từ đầu đến cuối vector second
vector <int> third (second.begin(),second.end())
//copy từ vector third
vector <int> four (third)
/*Vector 2 chiều*/
/* Tạo vector 2 chiều rỗng */
vector < vector <int> > v;
/* khai báo vector 5×10 */
vector < vector <int> > v (5, 10) ;
/* khai báo 5 vector 1 chiều rỗng */
vector < vector <int> > v (5) ;
//khai báo vector 5*10 với các phần tử khởi tạo giá trị là 1
vector < vector <int> > v (5, vector <int> (10,1) ) ;
```

Các hàm thành viên:

- Capacity:
  - size : trả về số lượng phần tử của vector. ĐPT O(1).
  - empty : trả về true(1) nếu vector rỗng, ngược lại là false(0). ĐPT O(1)
  - operator [] : trả về giá trị phần tử thứ []. ĐPT O(1).
  - at : tương tự như trên. ĐPT O(1).
  - front: trả về giá trị phần tử đầu tiên. ĐPT O(1).
  - back: trả về giá trị phần tử cuối cùng. ĐPT O(1).
- Chỉnh sửa
  - push_back : thêm vào ở cuối vector. ĐPT O(1).
  - pop_back : loại bỏ phần tử ở cuối vector. ĐPT O(1).
  - insert (iterator,x): chèn “x” vào trước vị trí “iterator” (x có thể là phần tử hay iterator của 1 đoạn phần tử…). ĐPT O(n).
  - erase : xóa phần tử ở vị trí iterator. ĐPT O(n).
  - swap : đổi 2 vector cho nhau (ví dụ: first.swap(second);). ĐPT O(1). - clear: xóa vector. ĐPT O(n).
- Nhận xét
  - Sử dụng vector sẽ tốt khi:
    - Truy cập đến phần tử riêng lẻ thông qua vị trí của nó O(1)
    - Chèn hay xóa ở vị trí cuối cùng O(1).

- Vector làm việc giống như một “mảng động”.

ví dụ

```
#include <iostream>
#include <vector>
using namespace std;
vector <int> v; //Khai báo vector
vector <int>::iterator it; //Khai báo iterator
vector <int>::reverse_iterator rit; //Khai báo iterator ngược
int i;
main() {
 for (i=1;i<=5;i++) v.push_back(i); // v={1,2,3,4,5}
 cout << v.front() << endl; // In ra 1
 cout << v.back() << endl; // In ra 5

 cout << v.size() << endl; // In ra 5

 v.push_back(9); // v={1,2,3,4,5,9}
 cout << v.size() << endl; // In ra 6

 v.clear(); // v={}
 cout << v.empty() << endl; // In ra 1 (vector rỗng)

 for (i=1;i<=5;i++) v.push_back(i); // v={1,2,3,4,5}
 v.pop_back(); // v={1,2,3,4}
 cout << v.size() << endl; // In ra 4

 v.erase(v.begin()+1); // Xóa ptử thứ 1 v={1,3,4}
 v.erase(v.begin(),v.begin()+2); // v={4}
 v.insert(v.begin(),100); // v={100,4}
 v.insert(v.end(),5); // v={100,4,5}

 /*Duyệt theo chỉ số phần tử*/
 for (i=0;i<v.size();i++) cout << v[i] << " "; // 100 4 5
 cout << endl;

/*Chú ý: Không nên viết
 for (i=0;i<=v.size()-1;i++) ...
 Vì nếu vector v rỗng thì sẽ dẫn đến sai khi duyệt !!!
 */

 /*Duyệt theo iterator*/
 for (it=v.begin();it!=v.end();it++)
 cout << *it << " " ;
 //In ra giá trị ma iterator đang trỏ tới "100 4 5"
 cout << endl;
 /*Duyệt iterator ngược*/
 for (rit=v.rbegin();rit!=v.rend();rit++)
 cout << *rit << " "; // 5 4 100
```

```
 cout << endl;

 system("pause");
}
```

Ví dụ 2: Cho đồ thị vô hướng G có n đỉnh (các đỉnh đánh số từ 1 đến n) và m cạnh và không có khuyên (đường đi từ 1 đỉnh tới chính đỉnh đó).
Cài đặt đồ thị bằng danh sách kề và in ra các cạnh kề đối với mỗi cạnh của đồ thị.
Dữ liệu vào:

- Dòng đầu chứa n và m cách nhau bởi dấu cách
- M dòng sau, mỗi dòng chứa u và v cho biết có đường đi từ u tới v. Không có cặp đỉnh u,v nào chỉ cùng 1 đường đi.

Dữ liệu ra:

- M dòng: Dòng thứ i chứa các đỉnh kề cạnh i theo thứ tự tăng dần và cách nhau bởi dấu cách.

Giới hạn: 1 <= n,m <= 10000
Ví dụ:

INPUT	OUTPUT
6 7	2 3 5 6
1 2	1 3 6
1 3	1 2 5
1 5	
2 3	1 3
2 6	1 2
3 5	
6 1	

```
#include<bits/stdc++.h>
using namespace std;
vector < vector <int> > a (10001);
int n,m,i,j,u,v;
int main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        cin>>u>>v;
        a[u].push_back(v);
        a[v].push_back(u);
    }
    for(int i=1;i<=m;i++)
    {
        sort(a[i].begin(), a[i].end());
    }
    for(int i=1;i<=m;i++)
    {
        for(int j=0;j<a[i].size();j++)
            cout<<a[i][j]<<" ";
        cout<<endl;
```

```
    }
}
```

STL Iterator

Một Iterator là một đối tượng có thể đi qua (iterate over) một container class mà không cần biết trật tự các phần tử bên trong mảng. Iterator còn là một cách để truy cập dữ liệu bên trong các container.

Các bạn có thể hình dùng Iterator giống như một con trỏ trỏ đến một phần tử nào đó bên trong container với một số toán tử đã được định nghĩa:

- Operator* cereference và trả về giá trị bên trong container tại vị trí mà iterator được đặt.
- Operator++ di chuyển iterator đến phần tử tiếp theo trong container.
- Operator-- ngược lại so với operator++.
- Operator== và operator!= dùng để so sánh vị trí tương đối của 2 phần tử đang được trỏ đến bởi 2 iterator.
- Operator= dùng để gán vị trí mà iterator trỏ đến.

Khai báo một Iterator

Với mỗi container class chúng ta sẽ có một kiểu iterator tương ứng. Mình sẽ lấy ví dụ về iterator của class std::vector như sau:

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

int main()
{
    std::vector<int> vec;

    std::vector<int>::iterator iter;

    return 0;
}
```

```
/*khai báo iterator “it”*/
vector <int> :: iterator it;
/* trỏ đến vị trí phần tử đầu tiên của vector */
it=vector.begin();
/*trỏ đến vị trí kết thúc (không phải phần tử cuối cùng nhé) của vector) */
it=vector.end();
/* khai báo iterator ngược “rit” */
```

```
vector <int> :: reverse_iterator rit; rit = vector.rbegin();
/* trỏ đến vị trí kết thúc của vector theo chiều ngược (không phải phần tử
đầu tiên nhé*/
rit = vector.rend();
```

SET (Tập hợp)

Tập hợp mình nhớ không nhầm là các bạn đã được học môn toán lớp 6. Set là tập hợp trong đó các phần tử chỉ suất hiện tối đa 1 lần

Khai báo

```
#include <set>
set <int> s;
set <int, greater<int> > s;
```

- Capacity:
  - size : trả về kích thước hiện tại của set. ĐPT O(1)
  - true nếu set rỗng, và ngược lại. ĐPT O(1).
- Modifiers:
  - insert : Chèn phần tử vào set. ĐPT O(logN).
  - erase : có 2 kiểu xóa: xóa theo iterator, hoặc là xóa theo khóa. ĐPT O(logN).
  - clear : xóa tất cả set. ĐPT O(n).
  - swap : đổi 2 set cho nhau. ĐPT O(n).
- Operations
  - find : trả về itarator trỏ đến phần tử cần tìm kiếm. Nếu không tìm thấy itarator trỏ về “end” của set. ĐPT O(logN).
  - lower_bound : trả về iterator đến vị trí phần tử bé nhất mà không bé hơn (lớn hơn hoặc bằng) khóa (dĩ nhiên là theo phép so sánh), nếu không tìm thấy trả về vị trí “end” của set. ĐPT O(logN).
  - upper_bound: trả về iterator đến vị trí phần tử bé nhất mà lớn hơn khóa, nếu không tìm thấy trả về vị trí “end” của set.. ĐPT O(logN).
  - count : trả về số lần xuất hiện của khóa trong container. Nhưng trong set, các phần tử chỉ xuất hiện một lần, nên hàm này có ý nghĩa là sẽ return 1 nếu khóa có trong container, và 0 nếu không có. ĐPT O(logN).

Ví dụ:

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

```
int main()
{
    std::set<int> s;
    s.insert(1);
    s.insert(2);
    s.insert(3);
    s.insert(1);

    // Chỉ 3 phần tử được thêm vào "s"
    std::cout<<"Số lượng phần tử: "<<s.size()<<std::endl;
    return 0;
}

output:
Số lượng phần tử: 3
```

ví dụ

```
#include <iostream>
#include <set>
using namespace std;
main() {
    set <int> s;
    set <int> ::iterator it;
    s.insert(9); // s={9}
    s.insert(5); // s={5,9}
    cout << *s.begin() << endl; //In ra 5
    s.insert(1); // s={1,5,9}
    cout << *s.begin() << endl;

    it=s.find(5);
    if (it==s.end()) cout << "Khong co trong container" << endl;
    else cout << "Co trong container" << endl;


    s.erase(it); // s={1,9}
    s.erase(1); // s={9}
    s.insert(3); // s={3,9}
    s.insert(4); // s={3,4,9}




    it=s.lower_bound(4);
    if (it==s.end()) cout << "Khong co phan tu nao trong set khong be hon 4" << endl;
    else cout << "Phan tu be nhat khong be hon 4 la " << *it << endl; // In ra 4

    it=s.lower_bound(10);
```

```
    if (it==s.end()) cout << "Khong co phan tu nao trong set khong be hon 10" << endl;
    else cout << "Phan tu be nhat khong be hon 10 la " << *it << endl; // Khong co ptu nao

    it=s.upper_bound(4);
    if (it==s.end()) cout << "Khong co phan tu nao trong set lon hon 4" << endl;
    else cout << "Phan tu be nhat lon hon 4 la " << *it << endl; // In ra 9



    /* Duyet set */
    for (it=s.begin();it!=s.end();it++) {
        cout << *it << " ";}
    // In ra 3 4 9

    cout << endl;
    return 0;
}
```

Lưu ý:

Nếu bạn muốn sử dụng hàm lower_bound hay upper_bound để tìm phần tử lớn nhất "bé hơn hoặc bằng" hoặc "bé hơn" bạn có thể thay đổi cách so sánh của set để tìm kiếm. Mời bạn xem chương trình sau để rõ hơn:

```
#include <iostream>
#include <set>
#include <vector>
using namespace std;
int main() {
 set <int, greater <int> > s;
 set <int, greater <int> > :: iterator it; // Phép toán so sánh là greater

 s.insert(1); // s={1}
 s.insert(2); // s={2,1}
 s.insert(4); // s={4,2,1}
 s.insert(9); // s={9,4,2,1}

 /* Tim phần tử lớn nhất bé hơn hoặc bằng 5 */
 it=s.lower_bound(5);
 cout << *it << endl; // In ra 4

 /* Tim phần tử lớn nhất bé hơn 4 */
 it=s.upper_bound(4);
 cout << *it << endl; // In ra 2
 return 0
}
```

Map (Ánh xạ):

- Map là một loại associative container. Mỗi phần tử của map là sự kết hợp của khóa (key value) và ánh xạ của nó (mapped value). Cũng giống như set, trong map không chứa các khóa mang giá trị giống nhau.
- Trong map, các khóa được sử dụng để xác định giá trị các phần tử. Kiểu của khóa và ánh xạ có thể khác nhau. - Và cũng giống như set, các phần tử trong map được sắp xếp theo một trình tự nào đó theo cách so sánh.
- Map được cài đặt bằng red-black tree (cây đỏ đen) – một loại cây tìm kiếm nhị phân tự cân bằng. Mỗi phần tử của map lại được cài đặt theo kiểu pair (xem thêm ở thư viện utility).

Khai báo:

```
#include <map>
...
map <kiểu_dữ_liệu_1,kiểu_dữ_liệu_2>
// kiểu dữ liệu 1 là khóa, kiểu dữ liệu 2 là giá trị của khóa.
```

Sử dụng class so sánh:

```
struct cmp{
 bool operator() (char a,char b) {return a<b;}
};
.....
map <char,int,cmp> m;
```

Truy cập đến giá trị của các phần tử trong map khi sử dụng iterator: Ví dụ ta đang có một iterator là it khai báo cho map thì:

```
(*it).first; // Lấy giá trị của khóa, kiểu_dữ_liệu_1
 (*it).second; // Lấy giá trị của giá trị của khóa, kiểu_dữ_liệu_2
 (*it) // Lấy giá trị của phần tử mà iterator đang trỏ đến, kiểu pair

 it->first; // giống như (*it).first
 it->second; // giống như (*it).second
```

Capacity:

- size : trả về kích thước hiện tại của map. ĐPT O(1)
- empty : true nếu map rỗng, và ngược lại. ĐPT O(1).

Truy cập tới phần tử:

- operator [khóa]: Nếu khóa đã có trong map, thì hàm này sẽ trả về giá trị mà khóa ánh xạ đến. Ngược lại, nếu khóa chưa có trong map, thì khi gọi [] nó sẽ thêm vào map khóa đó. ĐPT O(logN)

Chỉnh sửa:

- insert : Chèn phần tử vào map. Chú ý: phần tử chèn vào phải ở kiểu "pair". ĐPT O(logN).
- erase : o xóa theo iterator ĐPT O(logN) o xóa theo khóa: xóa khóa trong map. ĐPT: O(logN).
- clear : xóa tất cả set. ĐPT O(n).
- swap : đổi 2 set cho nhau. ĐPT O(n).

Operations:

- find : trả về itarator trỏ đến phần tử cần tìm kiếm. Nếu không tìm thấy iterator trỏ về "end" của map. ĐPT O(logN).
- lower_bound : trả về iterator đến vị trí phần tử bé nhất mà không bé hơn (lớn hơn hoặc bằng) khóa (dĩ nhiên là theo phép so sánh), nếu không tìm thấy trả về vị trí "end" của map. ĐPT O(logN).
- upper_bound: trả về iterator đến vị trí phần tử bé nhất mà lớn hơn khóa, nếu không tìm thấy trả về vị trí "end" của map. ĐPT O(logN).
- count : trả về số lần xuất hiện của khóa trong multiset. ĐPT O(logN)

Demo

```
#include <iostream>
#include <map>
#include <vector>
using namespace std;
int main() {
 map <char,int> m;
 map <char,int> :: iterator it;

 m['a']=1; // m={{'a',1}}
 m.insert(make_pair('b',2)); // m={{'a',1};{'b',2}}
 m.insert(pair<char,int>('c',3) ); // m={{'a',1};{'b',2};{'c',3}}

 cout << m['b'] << endl; // In ra 2
 m['b']++; // m={{'a',1};{'b',3};{'c',3}}

 it=m.find('c'); // it point to key 'c'

 cout << it->first << endl; // In ra 'c'
 cout << it->second << endl; // In ra 3

 m['e']=100; //m={{'a',1};{'b',3};{'c',3};{'e',100}}

 it=m.lower_bound('d'); // it point to 'e'
 cout << it->first << endl; // In ra 'e'
```

```
    cout << it->second << endl; // In ra 100

    return 0;
}
```

Duyệt map

```
map<int, char*> mymap, copymymap;
mymap[0] = "a";
mymap[1] = "b";
mymap[0] = "c";

cout << mymap.size();
```

Kiểm tra map có rỗng hay không

```
map<int, char*> mymap, copymymap;
mymap[0] = "a";
mymap[1] = "b";
mymap[0] = "c";

if (copymymap.empty()) cout << "copymymap is empty";
```

Truy xuất theo chỉ số

```
map<int, char*> mymap, copymymap;
mymap[0] = "a";
mymap[1] = "b";

cout << mymap[1] << " " << mymap.at(0);
```

Xóa bỏ một phần tử trong map

```
map<int, char*> mymap, copymymap;
mymap[0] = "a";
mymap[1] = "b";
mymap[5] = "c";
mymap[7] = "d";
mymap[9] = "e";

// xóa cặp đối tượng với "key" là 5
mymap.erase(5);
// => mymap =  {(0,"a"),(1,"b"),(7,"d"),(9,"e")}

map<int, char*>::iterator var = mymap.begin();
```

```
// xóa cặp đối tượng mà var đang truy cập
mymap.erase(var); // => mymap =  {(1,"b"),(7,"d"),(9,"e")}

var = mymap.find(7); // => var truy cập đến (7,"d")

// xóa từ vị trí var đang truy cập cho đến (9,"e")
mymap.erase(var, mymap.end()); // => mymap = {(1,"b")}
```

Xóa tất cả phần tử trong map

```
map<int, char*> mymap;
mymap[0] = "a";
mymap[1] = "b";
mymap[5] = "c";
mymap[7] = "d";
mymap[9] = "e";

mymap.clear();
// => mymap =  {}
```

Hoán đổi nội dung 2 map

```
map<char,int> map1, map2;

map1['x'] = 100;
map1['y'] = 200;

map2['a'] = 11;
map2['b'] = 22;
map2['c'] = 33;

map1.swap(map2);
// => map1 = {("x",11),("b",22),("c",33)}
// => map2 = {("x",100),("y",200)}
```

Các bài tập hay gặp mình sẽ tiếp tục update ở dưới

.....

....

Một số tài liệu tham khảo

https://vncoder.vn

https://www.cplusplus.com/reference/map/map/

https://www.geeksforgeeks.org/map-associative-containers-the-c-standard-template-library-stl/