算法竞赛基础 L0

一、C++简介

示例

```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

数据类型

```cpp
#include <iostream>

using namespace std;

int main() {
    int x = 1;
    double y = 3.14159;
    char z = 'a', w = '3';
    cout << "x=" << x << endl;
    cout << "y=" << y << endl;
    cout << "z=" << z << endl;
    cout << "w=" << w << endl;
    return 0;
}
```

数据类型	定义标识符	占字节数	范围
短整型	short	2	$-2^{15}\sim2^{15}-1$
整型	int	4	$-2^{31}\sim2^{31}-1$
长整型	long int / long	4	$-2^{31}\sim2^{31}-1$
双长整型	long long int / long long	8	$-2^{63}\sim2^{63}-1$
无符号整型	unsigned int	4	$0\sim2^{32}-1$
无符号双长整型	unsigned long long int / unsigned long long	8	$0\sim2^{64}-1$

数据类型	定义标识符	范围	占字节数	有效位数
单精度浮点	float	$-3.4E+38\sim3.4E+38$	4	7
双精度浮点	double	$-1.7E+308\sim1.7E+308$	8	16
长双精度浮点	long double	$-3.4E+4932\sim1.1E+4932$	8	16

全部使用long long以及double

输入输出

智能的 cin 和 cout

控制符	说明
%d	按十进制整型数据的实际长度输出。
%lld	输出长整型数据。
%u/%ulld	输出无符号整型（unsigned）。输出无符号整型时也可以用 %d，这时是将无符号转换成有符号数，然后输出。但编程的时候最好不要这么写，因为这样要进行一次转换，使 CPU 多做一次无用功。
%c	用来输出一个字符。
%f / %llf	用来输出实数，单精度或双精度，以小数形式输出。不指定字段宽度，由系统自动指定，整数部分全部输出，小数部分输出 6 位，超过 6 位的四舍五入。
%s	用来输出字符串。用 %s 输出字符串同前面直接输出字符串是一样的。但是此时要先定义字符数组或字符指针存储或指向字符串，这个稍后再讲。

第二个实际上通过 while 实现了将数组变成一个链表

C++函数

```
int f(int x){
    return x*x*x - x*x + 3;
}
int main()
{
    cout << f(3);
}
```

```
void Print_f(int x)
{
    cout << x*x*x - x*x + 3;
}
int main()
{
    Print_f(3);
}
```

C++作用域

```
if(...){
    int x;
    ...
}
cout << x;
```

```
void Print_f(int x)
{
    cout << x*x*x - x*x + 3;
}
void temp(){
    cout << x;
}
```

```
int x;
void fun(){
    cout << x;
}
int main()
{
    fun();
}
```

C++地址

	1	2	3	4	5	6	7	8
1								
2								
3								
4								
4								
5								

```
int x = 3;
int *p = &x;
*p = 4;
printf("%p\n",p);
printf("%p\n",&x);
printf("%d",x);
```

```
0058F9EC
0058F9EC
4
```

C++指针

```
void swap(int x,int y)
{
    int t = x;
    x = y;
    y = t;
    printf("SWAP: %p %p\n", &x, &y);
}

int main()
{
    int x = 3;
    int y = 4;
    swap(x, y);
    printf("MAIN: %p %p", &x, &y);
}
```

```
SWAP: 00EFF7EC 00EFF7F0
MAIN: 00EFF8D0 00EFF8C4
```

```
void swap(int *x,int *y)
{
    int t = *x;
    *x = *y;
    *y = t;
}
```

使用引用也可以实现变量互换的功能

C++引用

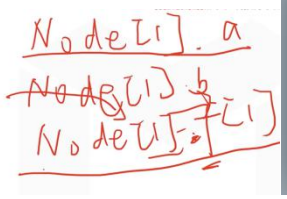
```
void swap(int &x,int &y)
{
    int t = x;
    x = y;
    y = t;
}
```

二、浅谈结构体

1.新的数据结构（struct）
2.新的运算符号（重载运算符）
3.应用

1.新的数据类型
Node 在 node 之上

```
struct node{
    int a,b;
    bool f[10];
    char c;
}Node[maxn];
```

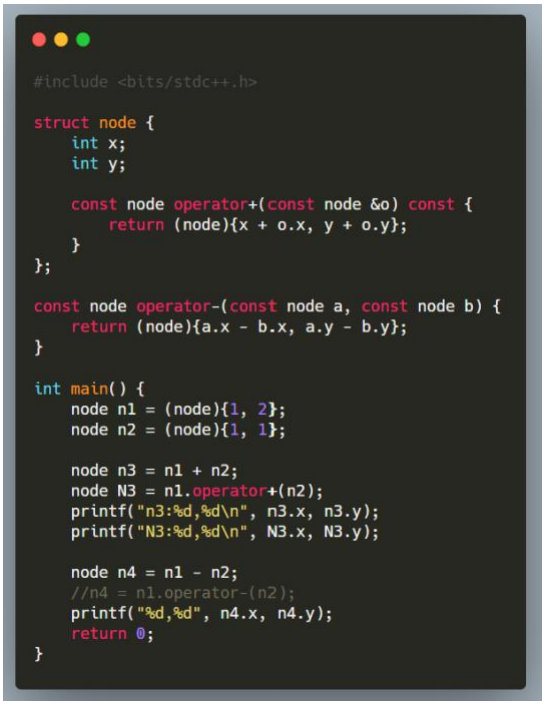



```
#include<bits/stdc++.h>
#define ll long long

const int maxn=100010;
struct node{
    int a,b;
    bool f[10];
    char c;
    inline void init(){
        memset(f,0,sizeof(f));
        a=b=0;
    }
}Node[maxn];
```

自定义的结构体函数 inline

2.新的运算符号
重写+，-运算符



```cpp
#include <bits/stdc++.h>

struct node {
    int x;
    int y;

    const node operator+(const node &o) const {
        return (node){x + o.x, y + o.y};
    }
};

const node operator-(const node a, const node b) {
    return (node){a.x - b.x, a.y - b.y};
}

int main() {
    node n1 = (node){1, 2};
    node n2 = (node){1, 1};

    node n3 = n1 + n2;
    node N3 = n1.operator+(n2);
    printf("n3:%d,%d\n", n3.x, n3.y);
    printf("N3:%d,%d\n", N3.x, N3.y);

    node n4 = n1 - n2;
    //n4 = n1.operator-(n2);
    printf("%d,%d", n4.x, n4.y);
    return 0;
}
```

n2 在此时传递了地址进去
减法操作的运算符定义在结构体外，就不能用.进行函数调用

3.应用
多权排序
减轻出现使用变量错误的几率

三、STL

1.入门

STL，英文全称 standardtemplatelibrary，中文可译为标准模板库或者泛型库，其包含有大量的模板类和模板函数，是 C++提供的一个基础模板的集合，用于完成诸如输入/输出、数学计算等功能。

思考下列问题
(1)现在给你 n 个无序的数，要求给他们排序
(2)现在给你 n 个无序的数，要求排序并去重
(3)现在给你 n 场不同球赛的参赛名单（如：勇士 vs 骑士），每支球队只赛一场，要求输入任意球队名称得到他的对手名称
朴素解法：(1)用某种排序方法（快速排序，冒泡排序等）
(2)在排序时加入判断重复的操作
(3)将球队名称与数组对应，再在数组间建立对应关系
调用 stl 解法：(1)无需手写，调用 std::sort()，完成排序
(2)将数存在 set 容器中，自动排序+去重
(3)将输入的球队名称建立 map 的对应关系，无需借助数组

STL的组成	含义
容器	一些封装数据结构的模板类，例如 vector 向量容器、list 列表容器等。
算法	STL 提供了非常多（大约 100 个）的数据结构算法，它们都被设计成一个个的模板函数，这些算法在 std 命名空间中定义，其中大部分算法都包含在头文件 <algorithm> 中，少部分位于头文件 <numeric> 中。
迭代器	在 C++ STL 中，对容器中数据的读和写，是通过迭代器完成的，扮演着容器和算法之间的胶合剂。
函数对象	如果一个类将 () 运算符重载为成员函数，这个类就称为函数对象类，这个类的对象就是函数对象（又称仿函数）。
适配器	可以使一个类的接口（模板的参数）适配成用户指定的形式，从而让原本不能在一起工作的两个类工作在一起。值得一提的是，容器、迭代器和函数都有适配器。
内存分配器	为容器类模板提供自定义的内存申请和释放功能，由于往往只有高级用户才有改变内存分配策略的需求，因此内存分配器对于一般用户来说，并不常用。

2.容器相关

容器分为三大类：

1)顺序容器

vector：后部插入/删除，直接访问

deque：前/后部插入/删除，直接访问

list：双向链表，任意位置插入/删除

2)关联容器

set：快速查找，无重复元素

multiset：快速查找，可有重复元素

map：一对一映射，无重复元素，基于关键字查找

multimap：一对一映射，可有重复元素，基于关键字查找

前 2 者合称为第一类容器

3)容器适配器

stack：LIFO

queue：FIFO

priority_queue：优先级高的元素先出

顺序容器

1)vector 头文件<vector>

实际上就是个动态数组。随机存取任何元素都能在常数时间完成。在尾端增删元素具有较佳的性能。

2)deque 头文件<deque>

也是个动态数组，随机存取任何元素都能在常数时间完成(但性能次于 vector)。在两端增删元素具有较佳的性能。

3)list 头文件<list>

双向链表，在任何位置增删元素都能在常数时间完成。不支持随机存取。

上述三种容器称为顺序容器，是因为元素的插入位置同元素的值无关，只跟插入的时机有关。

vector 的特点

顺序序列

顺序容器中的元素按照严格的线性顺序排序。可以通过元素在序列中的位置访问对应的元素。

动态数组

支持对序列中的任意元素进行快速直接访问，甚至可以通过指针算述进行该操作。操供了在序列末尾相对快速地添加或删除元素的操作。

能够感知内存分配器的（Allocator-aware）

容器使用一个内存分配器对象来动态地处理它的存储需求。

什么是 string?

标准模板库（STL）提供了一个 std::string 类，它是一个容器类，可把字符串当作普通类型来使用，并支持比较、连接、遍历、STL 算法、复制、赋值等等操作，这个类定义在<string>头文件中。

关联容器

关联式容器内的元素是排序的，插入任何元素，都按相应的排序准则来确定其位置。关联式容器的特点是在查找时具有非常好的性能。

1)set/multiset:头文件<set>

set 即集合。set 中不允许相同元素，multiset 中允许存在相同的元素。

2)map/multimap:头文件<map>

map 与 set 的不同在于 map 中存放的是成对的 key/value。

并根据 key 对元素进行排序，可快速地根据 key 来检索元素

map 同 multimap 的不同在于是否允许多个元素有相同的 key 值。

上述 4 种容器通常以平衡二叉树方式实现，插入、查找和删除的时间都是 O(logN)

什么是 set?

Set（集合）是 C++STL 库中自带的一个容器

set 具有以下两个特点：

1、set 中的元素都是排好序的

2、set 集合中没有重复的元素

什么是 map?

map 是 STL 的一个关联容器，它提供一对一的 hash。

第一个可以称为关键字(key)，每个关键字只能在 map 中出现一次；

第二个可能称为该关键字的值(value)；

map 以模板(泛型)方式实现，可以存储任意类型的数据，包括使用者自定义的数据类型。

Map 主要用于资料一对一映射(one-to-one)的情况。

map 内部的实现自建一颗红黑树，这颗树具有对数据自动排序的功能。在 map 内部所有数据都是有序的，后边我们会见识到有序的好处。比如一个班级中，每个学生的学号跟他的姓名就存在著一对一映射的关系。

容器适配器

1)stack:头文件<stack>

栈。是项的有限序列，并满足序列中被删除、检索和修改的项只能是最近插入序列的项。即按照后进先出的原则

2)queue:头文件<queue>

队列。插入只可以在尾部进行，删除、检索和修改只允许从头部进行。按照先进先出的原则。

3）priority_queue:头文件<queue>

优先级队列。最高优先级元素总是第一个出列

所有标准库容器共有的成员函数

相当于按词典顺序比较两个容器大小的运算符:

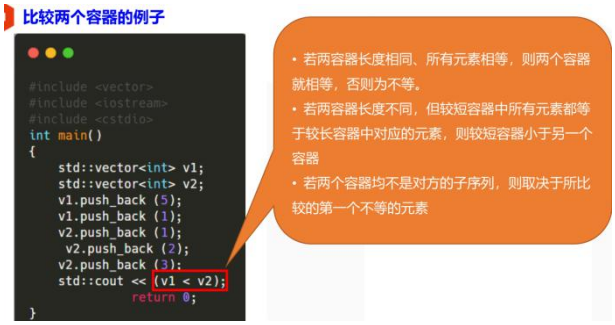
=,<,<=,>,>=,==,!=

empty:判断容器中是否有元素

max_size:容器中最多能装多少元素

size:容器中元素个数

swap:交换两个容器的内容



容器成员函数

只在顺序容器中的函数:

begin 返回指向容器中第一个元素的迭代器

end 返回指向容器中最后一个元素后面的位置的迭代器

rbegin 返回指向容器中最后一个元素的迭代器

rend 返回指向容器中第一个元素前面的位置的迭代器

erase 从容器中删除一个或几个元素

clear 从容器中删除所有元素

3.迭代器

要访问顺序容器和关联容器中的元素，需要通过“迭代器（iterator）”进行。迭代器是一个变量，相当于容器和操纵容器的算法之间的中介。迭代器可以指向容器中的某个元素，通过迭代器就可以读写它指向的元素。从这一点上看，迭代器和指针类似。

定义一个容器类的迭代器的方法可以是：容器类名::iterator 变量名；
或：容器类名::const_iterator 变量名；
访问一个迭代器指向的元素：
*迭代器变量名
迭代器上可以执行++操作,以指向容器中的下一个元素。

```cpp
#include <vector>
#include <iostream>
using namespace std;
int main()  {
    vector<int> v; //一个存放int元素的向量，一开始里面没有元素
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    v.push_back(4);
    vector<int>::const_iterator i;   //常量迭代器
    for( i = v.begin();i != v.end();i ++ )
        cout << * i << ",";
    cout << endl;
    vector<int>::reverse_iterator r;  //反向迭代器
    for( r = v.rbegin();r != v.rend();r++ )
        cout << * r << ",";
    cout << endl;
    vector<int>::iterator j;   //非常量迭代器
    for( j = v.begin();j != v.end();j ++ )
        * j =  100;
    for( i = v.begin();i != v.end();i++ )
        cout << * i << ",";
}
```

输出结果：

```
1,2,3,4,
4,3,2,1,
100,100,100,100,
```

2、特别注意

不同容器上支持的迭代器功能强弱有所不同。
容器的迭代器的功能强弱，决定了该容器是否支持 STL 中的某种算法。
例 1：只有顺序容器能用迭代器遍历。
例 2：排序算法需要通过随机迭代器来访问容器中的元素，那么有的容器就不支持排序算法。

3、STL 中的迭代器

STL 中的迭代器按功能由弱到强分为 5 种：
1.输入：Inputiterators 提供对数据的只读访问。
1.输出：Outputiterators 提供对数据的只写访问
2.正向：Forwarditerators 提供读写操作，并能一次一个地向前推进迭代器。
3.双向：Bidirectionaliterators 提供读写操作，并能一次一个地向前和向后移动。
4.随机访问：Randomaccessiterators 提供读写操作，并能在数据中随机移动。
编号大的迭代器拥有编号小的迭代器的所有功能，能当作编号小的迭代器使用。

4、不同迭代器所能进行的操作(功能)

所有迭代器：++p,p++
输入迭代器：*p,p=p1,p==p1,p!=p1
输出迭代器：*p,p=p1
正向迭代器：上面全部
双向迭代器：上面全部，--p,p--,
随机访问迭代器：上面全部，以及：
p+=i,p-=i,
p+i:返回指向 p 后面的第 i 个元素的迭代器
p－i:返回指向 p 前面的第 i 个元素的迭代器
p[i]:p 后面的第 i 个元素的引用
p<p1,p<=p1,p>p1,p>=p1

5、容器所支持的迭代器类别

容器迭代器类别

vector 随机

deque 随机

list 双向

set/multiset 双向

map/multimap 双向

stack 不支持迭代器

queue 不支持迭代器

priority_queue 不支持迭代器

例如，vector 的迭代器是随机迭代器，所以遍历 vector 可以有以下几种做法：

```
vector<int> v(100);
int i;
for(i = 0;i < v.size() ; i ++)
    cout << v[i];
vector<int>::const_iterator ii;
for( ii = v.begin(); ii != v.end ();ii ++
)   cout << * ii;
//间隔一个输出：
ii = v.begin();
while( ii < v.end()) {
    cout << * ii;
    ii = ii + 2;
}
```

而 list 的迭代器是双向迭代器，所以以下代码**可以**：

```
list<int> v;
list<int>::const_iterator ii;
for( ii = v.begin(); ii ! = v.end ();ii ++ )
        cout << * ii;
```

以下代码则不行：

```
for( ii = v.begin(); ii < v.end ();ii ++ )
    cout << * ii;
//双向迭代器不支持 <
for(int i = 0;i < v.size() ; i ++)
    cout << v[i]; //双向迭代器不支持 []
```

如何实现数组边遍历边删除

3.使用 for 循环正/倒序遍历

```
1   List<String> platformList = new ArrayList<>();
2   pplatformList.add("1");
3   platformList.add("2");
4   platformList.add("3");
5
6   for (int i = 0; i < platformList.size(); i++) {
7       String item = platformList.get(i);
8
9       if (item.equals("2")) {
10          platformList.remove(i);   注意：删除元素后，
11          i = i - 1;                要修正下下标的值，
12      }                             即 i = i - 1;
13  }
```

```
1   List<String> platformList = new ArrayList<>();
2   pplatformList.add("1");
3   platformList.add("2");
4   platformList.add("3");
5
6   for (int i = platformList.size() - 1; i >= 0; i--) {
7       String item = platformList.get(i);
8
9       if (item.equals("2")) {
10          platformList.remove(i);
11      }
12  }
```

4.其他 STL——算法

算法简介

STL 中提供能在各种容器中通用的算法，比如插入，删除，查找，排序等。大约有 70 种标准算法。

算法就是一个个函数模板。

算法通过迭代器来操纵容器中的元素。许多算法需要两个参数，一个是起始元素的迭代器，一个是终止元素的后面一个元素的迭代器。比如，排序和查找

有的算法返回一个迭代器。比如 find()算法，在容器中查找一个元素，并返回一个指向该元素的迭代器。

算法可以处理容器，也可以处理 C 语言的数组

算法分类

变化序列算法：

copy,remove,fill,replace,random_shuffle,swap,…..

会改变容器

非变化序列算法：

adjacent-find,equal,mismatch,find,count,search,count_if,for_each,search_n

以上函数模板都在<algorithm>中定义

此外还有其他算法，比如<numeric>中的算法

顺序容器共同操作

front():返回容器中第一个元素的引用

back():返回容器中最后一个元素的引用

push_back():在容器末尾增加新元素

pop_back():删除容器末尾的元素

算法示例：find()

template<classInIt,classT>

InIt**find(InItfirst,InItlast,constT&val);**

first 和 last 这两个参数都是容器的迭代器，它们给出了容器中的查找区间起点和终点。

这个区间是个左闭右开的区间，即区间的起点是位于查找范围之中的，而终点不是

val 参数是要查找的元素的值

函数返回值是一个迭代器。如果找到，则该迭代器指向被找到的元素。如果找不到，则该迭代器指向查找区间终点。

```cpp
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;
int main()  {
    int array[10] = {10,20,30,40};
    vector<int> v;
    v.push_back(1); v.push_back(2);
    v.push_back(3); v.push_back(4);
    vector<int>::iterator p;
    p = find(v.begin(),v.end(),3);
    if( p != v.end())
        cout << * p << endl;
    p = find(v.begin(),v.end(),9);
    if( p == v.end())
        cout << "not found " << endl;
    p = find(v.begin()+1,v.end()-2,1);
    if( p != v.end())
        cout << * p << endl;
    int * pp = find( array,array+4,20);
    cout << * pp << endl;
}
//输出：
//3
//not found
//3
//20
```

在 STL 中调用 STL

在使用 stl 中的容器时，也可以调用 stl 中的算法

例如：stl 中的算法之一：

lower_bound(begin,end,num)：从数组的 begin 位置到 end-1 位置二分查找第一个大于或等于 num 的数字，找到返回该数字的地址，不存在则返回 end。通过返回的地址减去起始地址 begin,得到找到数字在数组中的下标。

在 set 容器下，set.lower_bound()也是可以实现的操作

附：常用容器的常见成员函数

1、vector

2、set

3、map

4、string

std::vector 不定长数组——成员函数

c.assign(beg,end) c.assign(n,elem)	将[beg; end)区间中的数据赋值给c。 将n个elem的拷贝赋值给c。
c.at(idx)	传回索引idx所指的数据，如果idx越界，抛出out_of_range。
c.back()	传回最后一个数据，不检查这个数据是否存在。
c.begin()	传回迭代器重的可一个数据。
c.capacity()	返回容器中数据个数。
c.clear()	移除容器中所有数据。
c.empty()	判断容器是否为空。
c.end()	指向迭代器中的最后一个数据地址。

c.erase(pos) c.erase(beg,end)	删除pos位置的数据，传回下一个数据的位置。 删除[beg,end)区间的数据，传回下一个数据的位置。
c.front()	传回第一个数据。
get_allocator	使用构造函数返回一个拷贝。
c.insert(pos,elem) c.insert(pos,n,elem) c.insert(pos,beg,end)	在pos位置插入一个elem拷贝，传回新数据位置。 在pos位置插入n个elem数据。无返回值。 在pos位置插入在[beg,end)区间的数据。无返回值。
c.max_size()	返回容器中最大数据的数量。
c.pop_back()	删除最后一个数据。
c.push_back(elem)	在尾部加入一个数据。

c.rbegin()	传回一个逆向队列的第一个数据。
c.rend()	传回一个逆向队列的最后一个数据的下一个位置。
c.resize(num)	重新指定队列的长度。
c.reserve()	保留适当的容量。
c.size()	返回容器中实际数据的个数。
c1.swap(c2) swap(c1,c2)	将c1和c2元素互换。 同上操作。
vector<Elem> c vector <Elem> c1(c2) vector <Elem> c(n) vector <Elem> c(n, elem) vector <Elem> c(beg,end) c.~ vector <Elem>()	创建一个空的vector。 复制一个vector。 创建一个vector，含有n个数据，数据均已缺省构造产生。 创建一个含有n个elem拷贝的vector。 创建一个以[beg;end)区间的vector。 销毁所有数据，释放内存。

std::string 现代化的字符串——成员函数

成员函数	作用
assign()	赋以新值
swap()	交换两个字符串的内容
append()或push_back()	在尾部添加字符
insert()	插入字符
erase(int nStart,int nEnd)	删除nStart—nEnd位置字符
clear()	删除全部字符

成员函数	作用
replace()	替换字符
compare()	比较字符串
size()或length()	返回字符数量
max_size()	返回字符的可能最大个数
empty()	判断字符串是否为空
capacity()	返回重新分配之前的字符容量

成员函数	作用
getline()	从stream读取某值
copy()	将某值赋值为一个C_string
substr()	返回某个子字符串

std::set 快速查找——成员函数

成员函数	作用
begin()	返回set容器的第一个元素的地址
end()	返回set容器的最后一个元素地址
clear()	删除set容器中的所有的元素
empty()	判断set容器是否为空
max_size()	返回set容器可能包含的元素最大个数
size()	返回当前set容器中的元素个数
erase(it)	删除迭代器指针it处元素
insert(a)	插入某个元素

std::map 数组的下标——成员函数

成员函数	作用
begin()	返回指向map头部的迭代器
clear()	删除所有元素
count()	返回指定元素出现的次数
empty()	如果map为空则返回true
end()	返回指向map末尾的迭代器
equal_range()	返回特殊条目的迭代器对
erase()	删除一个元素

成员函数	作用
find()	查找一个元素
insert()	插入元素
key_comp()	返回比较元素key的函数
lower_bound()	返回键值>=给定元素的第一个位置
max_size()	返回可以容纳的最大元素个数

成员函数	作用
rbegin()	返回一个指向map尾部的逆向迭代器
rend()	返回一个指向map头部的逆向迭代器
size()	返回map中元素的个数
swap()	交换两个map
upper_bound()	返回键值>给定元素的第一个位置
value_comp()	返回比较元素value的函数

四、数据结构

1、数据结构
2、栈
3、队列
4、链表
5、括号序列

1、数据结构
数据结构是计算机存储和组织数据的一种方式。
数据结构的三要素：逻辑结构、物理结构、数据操作
我们在编写程序时，需要根据算法需求选取合适的数据结构，从而提高程序运行效率，并尽量消耗较少的空间资源。

2、栈
栈是支持如下操作的线性表：
入栈：将元素压入栈
出栈：将栈顶元素弹出
查询：返回栈顶元素
最后入栈的元素会最先被弹出，即“后进先出，LIFO”。

题目描述

给出项数为 n 的整数数列 $a_{1\ldots n}$。

定义函数 $f(i)$ 代表数列中第 i 个元素之后第一个大于 a_i 的元素的**下标**，即 $f(i)=\min_{i<j\le n,a_j>a_i}\{j\}$。若不存在，则 $f(i)=0$。

试求出 $f(1\ldots n)$。

输入格式

第一行一个正整数 n。

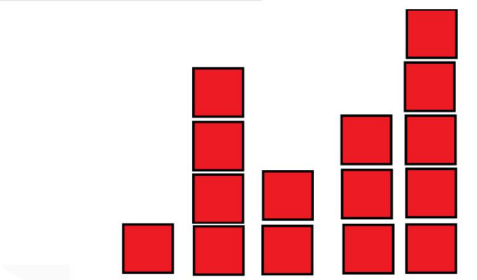
第二行 n 个正整数 $a_{1\ldots n}$。

输出格式

一行 n 个整数 $f(1\ldots n)$ 的值。

```
5
1 4 2 3 5
```

```
2 5 4 5 0
```



题目要我们干什么？查找每个数后面第一个比它严格大的数。
从后往前扫
对于每个点：
弹出栈顶比她小的元素
此时栈顶就是答案
加入这个元素
由于是从前往后输出，还要把答案放到一个数组里。

```
int main()
{
    int n,i,top=0;
    n=read();
    For(i,1,n)a[i]=read();
    Down(i,n,1)
    {
        while(top&&a[sta[top]]<=a[i])top--;
        //弹栈
        f[i]=sta[top];
        //答案
        sta[++top]=i;
        //进栈
    }
    For(i,1,n)write(f[i]),putchar(' ');
    return 0;
}
```

3、队列
队列是支持如下操作的线性表
入队：将元素排入队尾
出队：将队首元素排出
查询：返回队首/队尾元素
先入队的元素也会先出队，即“先进先出，FIFO”。

在用数组模拟队列时，队首/队尾只会不断向前移动，容易造成空间上的浪费。可以采用**循环队列**进行优化。

```
int Q[1024];
int head = 0, tail = -1;

tail = (tail + 1) % 1024; Q[tail] = x;
head = (head + 1) % 1024;
cout << Q[head] << endl;
```

2729:Blah数集

总时间限制：3000ms　内存限制：65536kB

描述

大数学家高斯小时候偶然间发现一种有趣的自然数集合Blah，对于以a为基的集合Ba定义如下：
(1) a是集合Ba的基，且a是Ba的第一个元素；
(2)如果x在集合Ba中，则2x+1和3x+1也都在集合Ba中；
(3)没有其他元素在集合Ba中了。
现在小高斯想知道如果将集合Ba中元素按照升序排列，第N个元素会是多少?

输入

输入包括很多行，每行输入包括两个数字，集合的基a(1<=a<=50))以及所求元素序号n(1<=n<=1000000)

输出

对于每个输入，输出集合Ba的第n个元素值

```
void Blah(int a,int n)
{
    q[1]=a;
    int two=1,three=1,rear=2;
    while (rear<=n) {
        long long t1=q[two]*2+1,t2=q[three]*3+1;
        int t=min(t1,t2);
        t1<t2?two++:three++;
        if(t==q[rear-1]) continue;
        q[rear++]=t;
    }
    cout<<q[n]<<endl;
}
int main()
{
    while(cin>>a>>n) {
        Blah(a,n);
    }
    return 0;
}
```

大小为n的栈：
空间复杂度：$O(n)$
时间复杂度：入栈/出栈/查询栈顶的时间均为$O(1)$

大小为n的队列：
空间复杂度：$O(n)$
时间复杂度：入队/出队/查询队首队尾的时间均为$O(1)$

4、链表

链表是一种基础数据结构。它将数据存储在不连续的内存空间中，通过指针把数据链接起来，形成一条链。它支持下列操作

在特定元素后插入一个元素

删除特定元素后的元素

遍历所有元素

注意链表并不支持随机访问，即它无法快速地回答第几个元素是什么

以双向链表为例，链表中的每一项元素不仅需要存储值，还需要存储向前和向后的两个指针，可以用结构体来表示。

为方便起见，我们通常会建一个空结点表示**链表头**。

```
struct node {
    int x;
    node *prev, *next;
    node(int _x, node *_prev, node *_next)
    : x(_x), prev(_prev), next(_next) {}
};
node *head = new node(0, NULL, NULL);
```

在结点 p 后插入值为 x 的结点：

```
void insert(node *p, int x) {
    node *q = new node(x, p, p->next);
    if (p->next)
        p->next->prev = q;
    p->next = q;
}
```

删除结点 p：

```
void erase(node *p) {
    p->prev->next = p->next;
    if (p->next)
        p->next->prev = p->prev;
    delete p;
}
```

查询第 k 个结点，返回其指针：

```
node *findk(int k) {
    node *p = head->next;
    for (int i = 0; i < k; ++i)
        p = p->next;
    return p;
}
```

遍历并输出链表：

```
void print() {
    for (node *i = head->next; i; i = i->next)
        cout << i->x << ' ';
    cout << endl;
}
```

大小为n的链表：

空间复杂度：$O(n)$

由于需要额外存储指针，空间消耗比数组略大

时间复杂度：

插入/删除单个元素：$O(1)$

随机访问：$O(n)$

遍历所有元素：$O(n)$

5、括号序列

合法的括号序列是这样定义的

1.空串是合法的。

2.如果字符串 S 是合法的，则(S)、[S]、{S}都是合法的。

3.如果字符串 A 和 B 是合法的，则 AB 也是合法的

例如，[{[]()}()]是合法的括号序列，但[(])不是合法的括号序列

现在给出一个字符串，问它是否是合法的括号序列。

题目描述

展开

定义如下规则序列(字符串)：

1．空序列是规则序列；

2．如果S是规则序列，那么(S)和[S]也是规则序列；

3．如果A和B都是规则序列，那么AB也是规则序列。

例如，下面的字符串都是规则序列：

()，[]，(())，([])，()[]，()[()]

而以下几个则不是：

(，[，]，)(，())，([(

现在，给你一些由‘(’，‘)’，‘[’，‘]’构成的序列，你要做的，是补全该括号序列，即扫描一遍原序列，对每一个右括号，找到在它左边最靠近它的左括号匹配，如果没有就放弃。在以这种方式把原序列匹配完成后，把剩下的未匹配的括号补全。

输入格式

输入文件仅一行，全部由‘(’，‘)’，‘[’，‘]’组成，没有其他字符，长度不超过100。

输出格式

输出文件也仅有一行，全部由‘(’，‘)’，‘[’，‘]’组成，没有其他字符，把你补全后的规则序列输出即可。

用栈实现