

基本框架:

```
#include<iostream>    // 头文件
using namespace std;  // 命名空间
int main() {
    // 主体
    return 0;
}
```

常用的头文件:

头文件的作用: 包含了头文件就可以使用其提供的功能, 例如想使用求根号的运算, 就需要包含<math.h>

(先简要介绍下, 后面详细介绍具体的使用)

```
#include<iostream>    有了它就可以用基本的输入输出, 例如 cin,cout
#include<vector>       使用 vector 容器 常用
#include<queue>        使用队列 常用
#include<algorithm>    使用一些算法, 常用的就是排序 sort()
#include<math.h>       使用一些数学函数, 如求根号, 求 x 的 y 次方等
#include<map>          现在不常用了, 常用 unordered_map 替代
#include<set>          常用来做去重和自动排序
#include<unordered_map> 哈希表 leetcode 刷题很常用
#include<deque>        双端队列
#include<fstream>      读写文件使用, 平常刷题不用, 但是如果需要读 txt, csv, excel 文件需要
#include<string>       处理字符串很方便, 常用
#include<cstdio>       如果想用 C 语言的输入输出, scanf printf
```

万能头:

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

可以只包含这一个, 就包含上述提及或者没有提及的所有头文件了, 优点在于不用敲那么多头文件, 缺点就是编译速度慢, 有的比赛可能不支持。

命名空间:

```
using namespace std;
```

但是如果不包含这行代码:

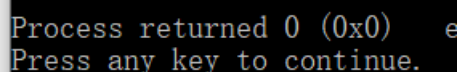
```
#include<bits/stdc++.h>
using namespace std;
int main() {
    cout<<"hello llf"<<endl;
    return 0;
}

#include<bits/stdc++.h>
int main() {
    std::cout<<"hello llf"<<std::endl;
    return 0;
}
```

cout 和 endl 就前面需要加上 std 这个作用域, 很麻烦, 因此就记住加上这段代码就可以了, 不用深入了解命名空间具体指什么。

return 0 :

如果程序正常运行, 没有语法错误, 那么可以从控制台中看到 return 0。



```
Process returned 0 (0x0)
Press any key to continue.
```

其他的数字 1 或者 -1 就表明程序有问题。

常用的数据类型:

int, double, char, string, long long, bool
float, unsigned int, unsigned long long 较少使用

整数:

常用 int, 如果数字很大, 考虑用 long long, 否则会溢出, 导致运算错误。
具体数据在什么范围内用 int?

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    cout<<INT_MIN<<" "<<INT_MAX<<endl;
    return 0;
}
```

-2147483648 2147483647

大致就是 21 亿, 通常就是 10^9 以内用 int 就可以, 超过 10^9 用 long long
如果有时候数据还是很大, 用 long long 也不够, 可以考虑用 unsigned long long ,其中 unsigned 表示无符号, 也就是不能表示负数, 只能表示正数, 正数表示范围*2, 另外为了简化书写, 常用 typedef unsigned long long ull; 意思是用 ull 表示 unsigned long long 这一长串:
具体例子:

```
#include<bits/stdc++.h>
using namespace std;
typedef unsigned long long ull;
int main() {
    ull a = 10000000000000000;
    cout<< a <<endl;
    return 0;
}
```

为了更好的理解这一问题, 再用 unsigned int 举例:

int 如今基本用 32 位表示数字, 但是其中首位是表示正负, 后面 31 位才表示数字, 因此表示范围 $[-2^{31}, 2^{31})$, 因为多表示了 0, 所以正数少一个。但如果用 unsigned int, 意思就是最高位不再表示正负, 直接表示 $[0, 2^{32})$,

注意: c++和c语言中使用某个变量之前必须要定义(或者说声明), 否则会报错, 但是c++不同于c语言变量必须在一开始定义, c++定义变量相对比较灵活, 随用随定义。

小数:

涉及到小数的运算, 直接用 double

这里面涉及到最后输出显示小数点位数的问题

```
double a = 3.1415926535;
//保留4位数字
cout << setiosflags(ios::fixed) << setprecision(4) << a <<endl;
```

字符型和字符串型:

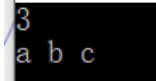
字符用 ' ', 字符串用 " ", 难点在于两者的混用

字符串 string 本质是一个字符数组, 可以像访问数组一样访问字符串。

```

string s = "abc";
// 字符串的长度
cout << s.size() << endl; // 3
// 遍历字符串 (类似遍历数组)
for(int i = 0; i < s.size(); i++){
    char c = s[i];
    cout << c << " ";
}
cout << endl;

```



C++11 特性, 简化的遍历版本:

```

for(char c : s){
    cout << c << " ";
}
cout << endl;

```

上面传统的用于理解, 平常用这个版本, 更简洁

字符串后追加字符串:

```

string s1 = "abc", s2 = "efg";
string s3 = s1 + s2; // 常用
//string s3 = s1.append(s2); //法二
cout << s3 << endl;

```

结果: abcefg

字符串后追加字符: (也可以使用+)

```

string s1 = "abc";
char c = 'x'; // 这边注意字符串用双引号, 字符用单引号
string s3 = s1 + c; // 常用
cout << s3 << endl;

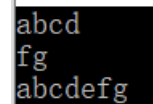
```

字符串截取: s.substr(k,len) 从 s 下标 k 开始, 截取长度 len, 如果 len>s.size(), 截取到最后。

```

string s1 = "abcdefg";
//取前4位
cout << s1.substr(0, 4) << endl;
//取后2位
cout << s1.substr(s1.size() - 2, 2) << endl;
cout << s1 << endl; //s.substr不会使原字符串发生改变

```

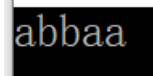


字符串插入: s1.insert(k,s2), 在 s1 的第 k 位插入字符串 s2

```

string s1 = "aaa", s2 = "bb";
cout << s1.insert(1, s2) << endl;

```



其他的操作遇到了百度即可, 通常就是用上面几种就够了。

bool 类型:

```

bool a = true, b = false;
bool a = 1,    b = 0;

```

两种方法等效, 所以通常就是用 1 表示对, 0 表示不对, 不同特意去写 True 和 False

基本的输入输出: cin, cout

例子, 输入两个数, 然后求出两数之和: (数据量较大) (直接 a+b 感觉你肯定会了)

a, b 都是 2×10^9 , 单独都在 int 范围内, 但是加起来超过了 int

```
# include<bits/stdc++.h>
using namespace std;
int main() {
    int a, b;
    cin >> a >> b; // a = 2000000000 b = 2000000000
    cout<< a + b <<endl;
    return 0;
}
```

```
2000000000 2000000000
-294967296
```

结果溢出。

解决方法：

```
# include<bits/stdc++.h>
using namespace std;
int main() {
    int a, b;
    cin >> a >> b; // a = 2000000000 b = 2000000000
    //法一：用更高的数据类型承接
    //long long c = a + b;
    //法二：强制类型转换
    cout<< (long long) a + b <<endl;
    return 0;
}
```

```
2000000000 2000000000
4000000000
```

常用的运算：

(1) 自增自减： a++, ++a, b--, --b

区分 a++和++a： 两者都是使 a 增加一，但是 a++使得 (a++) 这个整体数值不变，++a 使得 (++a) 这个整体数值加一。

```
int a = 1;
int b = a++;
cout<< a << " " << b <<endl;
```

```
2 1
```

```
int a = 1;
int b = ++a;
cout<< a << " " << b <<endl;
```

```
2 2
```

(2) 取余数和取整数运算： (leetcode 刷题常用)

取整 a/b 取余 a%b

```
int a = 11;
cout<< a / 3 <<" " << a % 3 <<endl;
```

```
3 2
```

即 $11 = 3 * 3 + 2$

(3) 逻辑运算符： 与 &&, 或 ||, 非 !

```
int a = 4;
cout << ((a > 3) && (a < 4)) << endl; // 0
cout << ((a > 2) || (a < 1)) <<endl; // 1
cout << !a << endl; // !0 等于 1, !x 等于 0 (x不等于0)
```

```
0
1
0
```

(4) 位运算：

```

int a = 4; //二进制0100
// 按位左移: <<
int b = a << 1; // 1000 = 8 , 后面补0
cout<< b << endl;

// 按位右移: >>
b = a >> 1; // 0010 = 2
cout<< b << endl;

// 按位与: &
cout<< ((a>>2) & 1) <<endl; // 0001 & 0001 = 0001 = 1

// 按位或:
cout<< (a | 1) <<endl; // 0100 | 0001 = 0101 = 5

// 按位异或: ^ 与0异或不变, 与1异或取反
cout<< (a ^ (1 << 2)) <<endl; // 0100 ^ 0100 = 0000 = 0

```

```

8
2
1
5
0

```

一些需要记得点:

ASCII 码: (字符转数字需要用)

0: 48 A: 65 a: 97

```

// 字符串第一个字符表示的数字和第二个字符表示的数字之和
string s = "23"; //输出 2 + 3 == 5
// 错误:
cout << s[0] + s[1] << endl; //结果为101, 是他们的ASCII码之和
// 正确:
cout << (s[0] - 48) + (s[1] - 48) << endl; // 数字字符减去0的ASCII码才表示真实的数字
cout << (s[0] - '0') + (s[1] - '0') << endl; //简化, 不需要记ASCII码, 注意单引号。

```

```

101
5
5

```

换行的转义字符: '\n' 记得是右斜杠

```
printf("a\nb");
```

```
a
b
```

基本语句: 条件, 循环等

条件语句:

```

if(条件){
    执行语句
}
else if(条件){
    执行语句
}
else{
    执行语句
}

if(条件){
    执行语句
}
else if(条件){
    执行语句
}
else{
    执行语句
}

如果执行语句只有一行
if(条件)  执行语句;
else      执行语句;

```

switch 语句

switch(表达式)

```

{
    case 常量表达式 1: 语句
    case 常量表达式 2: 语句
    case 常量表达式 3: 语句
}

```

当 switch 表达式的值与某个 case 子句中的常量表达式的值想匹配的时候, 就从该 case 子句往下一直执行。如下面的例子, 输出并不是仅仅输出 81-90, 而是从 case B 开始一直执行。

```
char grade = 'B';
switch(grade) {
    case 'A': cout << "91-100" << endl;
    case 'B': cout << "81- 90" << endl;
    case 'C': cout << "71- 80" << endl;
    case 'D': cout << "61- 70" << endl;
    default: cout<<"error"<<endl;
}
```

```
81- 90
71- 80
61- 70
error
```

若要想达到理想的效果，应该在每个 case 语句后加 break

```
char grade = 'B';
switch(grade) {
    case 'A': cout << "91-100" << endl; break;
    case 'B': cout << "81- 90" << endl; break;
    case 'C': cout << "71- 80" << endl; break;
    case 'D': cout << "61- 70" << endl; break;
    default: cout<<"error"<<endl;
}
```

```
81- 90
```

另外经过测试，switch(表达式)中的表达式只能是数值类型的数据(数字或者字符(字符转化为其 ASCII 码,因此也是个数字))，不能是字符串等其他类型。

循环语句： while 和 for （感觉不用说具体的语法，直接上例子）

例如：求 2 的 10 次方

```
//求 2^10 ,即10个2相乘
int res = 1, num = 10;
while(num--){
    res *= 2;
}
cout << res << endl;

//求 2^10 ,即10个2相乘
int res = 1, num = 10;
for(int i = 0; i < num; i++){
    res *= 2;
}
cout << res << endl;
```

```
1024
```

循环语句中注意 continue 和 break 的区别：

continue 指的是仅跳过本次循环，即本次循环 continue 后面的语句都不再执行，直接跳到下一次循环。

break 是跳出本层循环，即其所属的 for 或者 while 语句均不在执行，注意只能跳出一层。

例如：

```
for(int i = 0; i < 5; i++){
    if(i == 3) continue;
    cout<< i << endl;
}
```

```
0
1
2
4
```

```
for(int i = 0; i < 5; i++){
    if(i == 3) break;
    cout<< i << endl;
}
```

```
0
1
2
```

注意 break 只能跳出一层循环，这个地方容易忽略。

例如：想找到 5 以内的两个数相加等于 7，找到一个即可。如果直接里层 break，仅仅是跳出了 for(j=0;j<5;j++)，而有关 i 的循环仍会执行。

```

for(int i = 0; i < 5; i++){
    for(int j = 0; j < 5; j++){
        if(i + j == 7){
            cout << i << " " << j << endl;
            break;
        }
    }
}

```

```

3 4
4 3

```

常用解决方法：（在不改变 for 循环的写法的的情况下）

```

bool isfind = 0;
for(int i = 0; i < 5; i++){
    for(int j = 0; j < 5; j++){
        if(i + j == 7){
            cout << i << " " << j << endl;
            isfind = 1; //表示已经找到
            break;
        }
    }
    if(isfind) break; // 这里 isfind == 1可以简写为 isfind
}

```

```

3 4

```

函数：

使用函数的作用：大致就是为了将某个功能封装为一个函数，从而有助于频繁调用和精简主函数。

例如：编写一个函数 find_prime(a,b)，找到区间[a,b]内最小的质数

```

//形参：接受实参的数值
int find_prime(int a, int b){ //默认a>2, a<b
    for(int i = a; i <= b; i++){
        int x = i; //接下来判断x是否是质数
        //根据质数的定义，只有1和本身可以整除
        bool isprime = 1; //默认是质数，然后判断是否正确
        for(int j = 2; j < x; j++){
            if(x % j == 0){ //表示存在除了1和本身之外的数可以整除x，因此x不是质数
                isprime = 0;
                break;
            }
        }
        if(isprime) return x; //return返回值后整个函数就结束
    }
}

```

渴望接收int，因此函数的返回值类型为int

实参

```

int mini_prime = find_prime(100, 110);
cout << mini_prime << endl;

```

```

101

```

一个经典的问题：编写一个函数，实现交换两个数，swap(a,b),交换 a,b 的值。

```

int a = 3, b = 5;
swap(a, b);
cout << a << " " << b << endl;
return 0;

```

错误做法：

```
void swap(int a, int b) {
    int c = b;
    b = a; //a的值赋给b
    a = c; //b的值赋给a
}
```

3 5

结果表明并没有实现交换，原因在于形参的 a,b 和实参的 a,b 不是一个东西，也就是它们并不指向同一内存，形参 a,b 的作用域仅仅是定义的 swap 函数，因此当函数执行完后，它们的内存空间就释放，虽然函数内部实现了形参 a,b 的交换，但是并没有返还给实参，因此实参的数值并未发生变化。

正确做法： 用引用 & 只需要小小的一点改动

```
void swap(int &a, int &b) {
    int c = b;
    b = a; //a的值赋给b
    a = c; //b的值赋给a
}
```

```
void swap(int &x, int &y) {
    int c = y;
    y = x; //a的值赋给b
    x = c; //b的值赋给a
}
```

就是直接在形参 a,b 前加上&这个符号，这个东西类似指针。 int &x=y, 表示给 y 起一个别名 x，它们是同一个东西，仅仅是名字不同，通过不同的名字均可以访问到这同一个变量的内容。左图就表示此时实参的 a,b 和形参的 a,b 指向同一内存，因此，形参的交换就意味着实参的交换。右图刻意把 a 换成了 x，b 换成了 y，表示这仅仅是一个别名，别名可以与原名相同，即都叫 a,也可以取不同的名字，但是都表示一块内存，因为上面两个写法等效。都是对的

5 3

结果确实实现了形参和实参的交换。

通常函数只有一个返回值(int, double)或者没有返回值(void)，如果有多个返回值，怎么办？

第一种方法就是使用上述的传引用&，形参和实参表示的是同一个东西，形参修改了，实参也就跟着修改了，但是用这种方法一定要小心避免无意的修改。

第二种方法就是用数组或者 vector 或者自定义的类型返回多个值。

数组：

（后面有了 stl 中的 vector 后，数组就不经常用了。）

一维数组：

```
int a[5];
```

初始化和遍历方式：（遍历的意思就是将数组从头到尾访问一遍，每个地方都走一遍）

```
int a[5] = {1, 2, 3}; // 定义时初始化，前3个依次赋值为1, 2, 3，后面默认为0
a[3] = 4, a[4] = 5; // 直接访问赋值
for(int i=0; i<5; i++){ // 遍历数组
    cout << a[i] << " ";
}
```

1 2 3 4 5

也可以通过输入初始化，多输入的无效。例如多输入了 6，但是没影响，程序只读前 5 个。

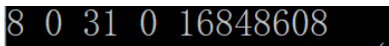
```
int a[5];
for(int i=0; i<5; i++) cin >> a[i];
for(int i=0; i<5; i++) cout << a[i] << " ";
```

1 2 3 4 5 6
1 2 3 4 5

小细节：

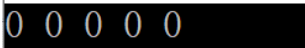
如果数组是在函数中定义的，就是局部变量，那么其数值是随机的，并不是想当然的 0

```
int main() {  
    int a[5];  
    for(int i=0;i<5;i++) cout<<a[i]<<" ";  
    return 0;  
}
```



但如果是定义在函数外的全局变量，那么默认初始化为 0

```
int a[5];  
int main() {  
    for(int i=0;i<5;i++) cout<<a[i]<<" ";  
    return 0;  
}
```



二维数组：一般就是两重循环处理

例如：生成一个二维数组[1,2,3; 4,5,6; 7,8,9] (;表示换行)，然后打印第二列。

```
int a[3][3];  
int cnt = 1;  
// 生成数组[1, 2, 3; 4, 5, 6; 7, 8, 9]  
for(int i = 0; i < 3; i++) //如果循环里面只有一条语句，可以省略大括号  
    for(int j = 0; j < 3; j++)  
        a[i][j] = cnt++;  
// 打印生成数组  
for(int i = 0; i < 3; i++) {  
    for(int j = 0; j < 3; j++)  
        cout << a[i][j] << " ";  
    cout << endl;  
}  
// 打印第二列  
cout << "第二列的结果: " << endl;  
for(int i=0;i<3;i++) cout<<a[i][1]<<" ";  
cout<<endl;
```

指针：不用看，基本不用

STL

一、vector: 向量，不定长数组（重要，基本数组都被 vector 取代了）

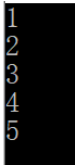
包含头文件： #include<vector>

初始化：

```
vector<int>v; //不指定长度，随着需要动态分配  
vector<int>v(10); //生成长度为10的vector, 类似于数组的用法，如果不够了仍旧可以拓展  
vector<int>v(10, 1); //生成长度为10的vector, 每个初始化为1.
```

遍历 vector:

```
vector<int>v; // 初始的v为空，没有什么内容，可以通过push_back()往里面添加内容  
// 用 1-5 初始化v  
for(int i = 0; i < 5; i++) v.push_back(i + 1);  
// push_back() 顾名思义就是在v的末尾不断追加新的内容  
  
//遍历 1: 类似数组  
for(int i = 0; i < 5; i++) cout << v[i] << endl;
```



```

//遍历 2: c++11特性, 常用, 简单方便
for(auto x : v) cout << x << endl;
//大致意思就是从v中依次取出元素给x, 直到访问完v中所有的元素
//auto 的意思就是让程序自动判别x的类型, 因为v中元素的类型为int, 因此此处auto == int

//遍历 3: 用迭代器, 了解即可
for(vector<int>::iterator i = v.begin(); i != v.end(); i++) cout<<*i<<endl;
//因为内置函数begin(), end() 返回值是一个迭代器, 迭代器理解为指针就可以, 可以通过 * (迭代器) 访问具体的值

//遍历 4: 用auto简化迭代器
for(auto i = v.begin(); i != v.end(); i++) cout<<*i<<endl;
//auto 自己判断接收的类型为迭代器

```

四种遍历方式的结果都是 1, 2, 3, 4, 5

其他常用的操作:

```

v.size()    //返回元素个数
v.empty()   //是否为空
// v.size()和v.empty()时间复杂度是O(1)的
v.clear()   //清空
v.front()/v.back() //返回第一个元素和最后一个元素
v.push_back()/pop_back() //插入或者删除一个元素
v.begin() //v的第一个数的迭代器
v.end()   //v的最后一个数后面一个位置的迭代器
a>b a<b a==b //支持比较运算 字典序比较

```

具体例子:

```

vector<int>v(5, 1); //初始化为5个1
// v.size()
int len = v.size();
cout << len<< endl; //输出当前数组的长度
// v.push_back()
v.push_back(6); //往后面添加一个数6
// 遍历v
for(auto x : v) cout<<x<<" ";
cout<<endl;

// v.back()
int fin_num = v.back(); //获取v的最后一个数
cout << fin_num << endl;
// v.front()
int sta_num = v.front(); //获取v的第一个数
cout <<sta_num <<endl;
// v.pop_back()
v.pop_back(); // 删除最后一个数
// 查看删除后的结果
for(auto x : v) cout<<x<<" ";
cout<<endl;

//v.empty()
bool isempty = v.empty(); //1表示空 0表示非空
if(isempty) cout<<"v空了"<<endl;
else cout<<"v非空"<<endl;

// v.clear()
v.clear(); //清空v

if(v.empty()) cout<<"v空了"<<endl;
else cout<<"v非空"<<endl;

```

```

5
1 1 1 1 1 6
6
1
1 1 1 1 1
v非空
v空了

```

二维 vector

```
//二维vector定义
vector<vector<int>>mat;
//赋值的话是一层一层的赋值
//例如赋值为全1矩阵(3*3)
vector<int>a;
for(int i=0;i<3;i++){ //依次处理二维矩阵的某一行
    vector<int>a;
    for(int j=0;j<3;j++) a.push_back(1); //处理某一行
    mat.push_back(a);
}
```

leetcode 常用定义方法

```
//定义时初始化
int m = 3, n = 4; // 3行4列
//定义一个3行4列的全0矩阵
vector<vector<int>>mat(m, vector<int>(n, 0));
```

二、pair

需要包含的头文件: #include<utility> (有的编译器不包含也没问题)

pair 用来存储成对出现的东西。

```
// 存储一个人的名字和年龄
pair<string, int> a;
a.first = "Alice";
a.second = 21;
cout<< a.first << " " << a.second << endl;

pair<string, int> a;
// 更加简单的赋值方式:
a = {"Alice", 21}; //用花括号|
cout<< a.first << " " << a.second << endl;
```

Alice 21

如果需要处理很多人, 结合 vector

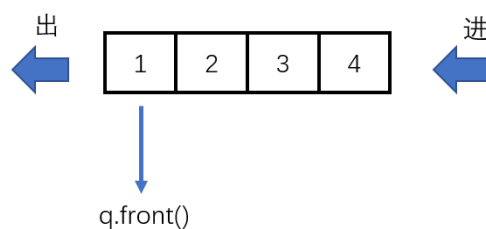
vector<pair<string,int>>v

考虑到 pair 一般需要很长, 可以用 typedef 简化:

```
typedef pair<string, int> PSI;
PSI a;
a = {"Alice", 21};
cout<< a.first << " " << a.second << endl;
```

三、队列:

```
queue, 队列
size()
empty()
push() 向队尾插入一个元素
front() 返回队头元素
back() 返回队尾元素
pop() 弹出队头元素
```



需要包含头文件: #include<queue>

```

queue<int>q; //定义一个队列，初始为空
// 往队列中加入一个数2
q.push(2); //vector用的是push_back(), 容易混淆
q.push(3); //再加入一个3
//队列不能直接看到队列中所有的内容，它只能看到队列中的第一个数
// 因此访问队列所有的元素并打印的方法为：
while(q.size()) // 当q不为空，有元素时
{
    int e = q.front(); //得到队首元素
    cout<<e<<endl;
    q.pop(); // 队首元素弹出，从而使得后面的元素补位
}

```

2
3

一个小例子：假设考虑一维空间，公主在 $x = 10$ ，王子在 $x = 1$ ，王子每次有两种选择，直接跳到当前位置的两倍处（例如当前在 $x = 4$ ，可以直接跳到 8），或者向左移动一步，问王子最短多少步可以找到他的公主。（queue + pair）

```

typedef pair<int,int>PII; // first表示当前王子在的位置，second表示当前的步数
queue<PII>q;
q.push({1,0}); //初始化，王子在x=1, 当前步数为0
while(q.size()){
    // 取出队首元素
    auto e = q.front();
    q.pop();
    //判断是否已经找到公主，如果找到，输出最短的步数
    if(e.first == 10){
        cout<<e.second<<endl;
        break; //终止
    }
    // 如果没找到，王子有两种选择
    q.push({e.first * 2, e.second + 1}); //直接跳到2倍处，步数加1
    q.push({e.first - 1, e.second + 1}); //向左走一步，步数加1
}

```

6

四、优先队列

需要包含头文件：#include<queue>

你尽管往队列中添加元素，队列自动将当前“最大”的元素放在队首，因为队列只能访问队首元素，因此每次访问的都是队列中的“最大”值，此处，“最大”可以自己定义什么是大，自定义排序规则。

```

priority_queue<int>pq;
// 依次插入数字 5, 3, 7
pq.push(5);
pq.push(3);
pq.push(7);
//输出队首元素，也就是队列中的最大值
cout<<pq.top()<<endl;

```

7

如果想自定义排序规则，定义小的数在前面：

```

priority_queue<int,vector<int>,greater<int>>pq;
// 依次插入数字 5, 3, 7
pq.push(5);
pq.push(3);
pq.push(7);
//输出队首元素，也就是队列中的最大值
cout<<pq.top()<<endl;

```

3

五、哈希表 unordered_map

需要包含头文件： `#include<unordered_map>`

功能：给定一串字符，问每个字符出现的次数

```
unordered_map<char, int>hash;
string s = "sdasdadsadfrfrgrgtrgh";
//统计字符串s中每个字符c出现的次数
for(auto c : s){
    hash[c]++; //字符c的出现次数+1
}
// 输出a出现的次数
cout<< hash['a']<<endl;
```

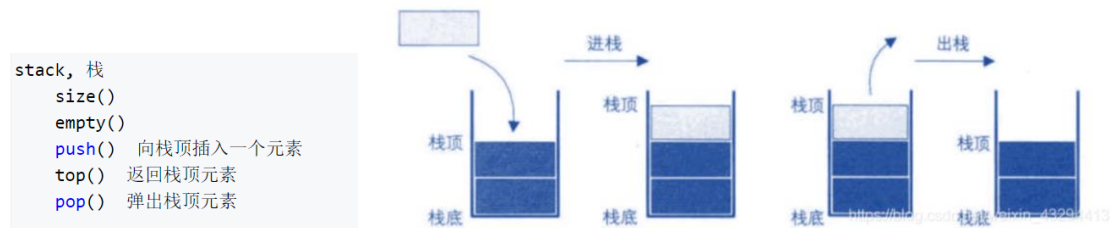
结果：3

```
//遍历所有的字符出现次数
for(auto x : hash){
    cout<<x.first<<" "<<x.second<<endl;
}
```

```
h 1
t 1
d 4
s 3
r 3
a 3
f 2
g 3
```

六、栈： stack

需要包含头文件： `#include<stack>` 特点：后进先出



```
stack<int>stk; //定义栈
vector<int>v = {1,2,3};
for(auto x : v) stk.push(x); //将1, 2, 3依次插入栈中
while(stk.size()){ //只要栈非空
    int e = stk.top(); //查看栈顶元素并输出
    cout<<e<<endl;
    stk.pop(); //弹出栈顶元素
}
```

```
3
2
1
```

七： map, set, multiset, multimap, list, deque 感觉用的不多，需要用到的时候再学。

结构体：

大致意思就是把有相互关系的一些变量（可以是不同类型的）反正一起处理。

例如定义一个（姓名，年龄，性别）的结构体

```
struct node{
    string name;
    int age;
    string sex;
}; //这个地方的：记得不要丢掉
```

赋值:

```
// 赋值方式一
node a;
a.name = "Alice";
a.age = 18;
a.sex = "女";
// 赋值方式二: {} 按顺序赋值
node b = {"Bob", 21, "男"};
node c = {"HH", 17, "男"};
```

排序: (太常用了, sort) 一定要会自定义排序规则, 利用结构体

需要包含头文件: `#include<algorithm>`

简单常规排序(默认从小到大)

```
vector<int>v = {2, 4, 1, 6, 7, 4, 11};
sort(v.begin(), v.end());
for(auto x: v) cout<<x<<" ";
cout<<endl;
```

1 2 4 4 6 7 11

从大到小

```
vector<int>v = {2, 4, 1, 6, 7, 4, 11};
sort(v.begin(), v.end(), greater<int>());
for(auto x: v) cout<<x<<" ";
cout<<endl;
```

11 7 6 4 4 2 1

自定义排序规则: 以结构体样例为例, 按照每个人的年龄顺序从大到小输出

```
struct node{
    string name;
    int age;
    string sex;
}; // 这个地方的; 记得不要丢掉
```

```
// 自定义排序规则
bool cmp(node a, node b){
    return a.age > b.age; // 年龄大的放在前面
}
```

```
vector<node>v;
// 将这三个人按照年龄从大到小的顺序输出
v.push_back({"Alice", 18, "女"});
v.push_back({"Bob", 21, "男"});
v.push_back({"HH", 17, "男"});
sort(v.begin(), v.end(), cmp); // 自定义排序规则cmp
for(auto x: v){
    cout<<x.name<<" "<<x.age<<" "<<x.sex<<endl;
}
```

Bob 21 男
Alice 18 女
HH 17 男

文件读写: (有时需要从 txt, csv 文件中读数据)

需要包含头文件: `#include<fstream>`

txt, csv 文件类似, 一样的读法

读文件:

从 txt 文档中读取信息, 存放到上述的二维数组中。

a - 记事本

文件(F) 编辑(E) 格式

1,2,3
4,5,6
7,8,9

txt 文档内容如左图所示:

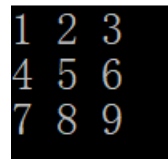
```
//设置路径
string path = "C:/Users/17292/Desktop/a.txt"; //注意是左斜杠, 如果是右斜杠, 需要两个
//string path = "C:\\Users\\17292\\Desktop\\a.txt";
ifstream infile(path, ios::in);
if(!infile) cout<<"error"<<endl; //如果打开文件失败, 输出错误信息

vector<vector<int>>mat; //二维vector存储矩阵

string line;
while (getline(infile, line)) { //逐行读取txt文档的内容存入line中
    // 注意此时的line是一个字符串, 需要进一步处理将其按逗号分割
    vector<int>v; // 矩阵每一行的数据放到res中

    istringstream ss(line); //
    string s;
    //处理当前行的每一列
    while (getline(ss, s, ',')) { //将ss以, 分割后依次给s
        int num = stoi(s); //stoi() 表示将字符串转化为整数
        v.push_back(num);
    }
    mat.push_back(v);
}

//打印读入的矩阵
for(int i = 0; i < mat.size(); i++) {
    for(int j = 0; j < mat[0].size(); j++) {
        cout<<mat[i][j]<<" ";
    }
    cout<<endl;
}
```



1	2	3
4	5	6
7	8	9

注意点:

- (1) 分隔符可以根据实际情况修改, 空格", 逗号", 或者其他符号
- (2) 用 istringstream 需要包含头文件 #include <sstream>
- (3) 不是固定的读法, 网上版本很多, 看自己喜欢哪个

写文件: (比读文件简单)

就是把内存中的数据写入文档中

例: 将上述矩阵输出到另一个 b.txt 中。

// 写文件

```
ofstream outfile("C:/Users/17292/Desktop/b.txt", ios::out);
for(int i = 0; i < mat.size(); i++) {
    for(int j = 0; j < mat[0].size(); j++) {
        outfile<<mat[i][j]<<" ";
    }
    outfile<<endl;
}
```

b - 记事本

文件(F) 编辑(E) 格式(O)

1 2 3
4 5 6
7 8 9

leetcode



下面可以选择按类型刷题，通常按类型刷题提升较大，效率高，具体的顺序参照这个网站：<https://programmercarrl.com/>

数组 1247 字符串 600 哈希表 441 动态规划 431 数学 419 深度优先搜索 295 排序 279 贪心 248
广度优先搜索 239 树 230 二分查找 201 二叉树 200 数据库 195 矩阵 186 双指针 182 位运算 164 栈 148
设计 137 堆 (优先队列) 124 回溯 113 图 112 模拟 100 链表 98 前缀和 92 计数 87 滑动窗口 80
并查集 71 递归 61 二叉搜索树 54 分治 52 字典树 50 单调栈 47 有序集合 45 队列 40 状态压缩 37
几何 35 记忆化搜索 35 枚举 34 线段树 30 拓扑排序 30 哈希函数 25 博弈 24 数据流 22 树状数组 22
字符串匹配 19 交互 18 滚动哈希 17 最短路 16 组合数学 16 随机化 14 数论 14 归并排序 12
单调队列 11 双向链表 11 快速选择 10 迭代器 10 脑筋急转弯 10 概率与统计 9 多线程 9 桶排序 8
后缀数组 6 计数排序 6 最小生成树 5 扫描线 4 Shell 4 水桶抽样 4 欧拉回路 3 强连通分量 2
拒绝采样 2 基数排序 2 双连通分量 1

收起

以第一题为例：经典两数之和：

题目描述

评论 (10.9k)

题解 (17.3k)

提交记录

1. 两数之和

难度 简单

14204 人做过

给定一个整数数组 `nums` 和一个整数目标值 `target`，请你在该数组中找出 **和为目标值** `target` 的那 **两个** 整数，并返回它们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素在答案里不能重复出现。

你可以按任意顺序返回答案。

示例 1：

```
输入：nums = [2,7,11,15], target = 9
输出：[0,1]
解释：因为 nums[0] + nums[1] == 9，返回 [0, 1]。
```

1

class Solution {

2

public:

3

vector<int> twoSum(vector<int>& nums, int target) {

4

5

}

6

};

虽然框架结构涉及到了c++类的内容，但是可以忽略主要就是实现一个函数的功能，输入从括号中的形参获取，结果按照要求的数据类型return即可

问题的基本意思就是给定 `nums`，然后在 `nums` 中找到两个数，它们的和为 `target`，返回这两个数的下标，因为涉及到返回多个值，因此题目返回值用了 `vector`。

方法一：两重循环


```
vector<int> twoSum(vector<int>& nums, int target) {
    int n = nums.size();
    for(int i = 0; i < n; i++)
        for(int j = i + 1; j < n; j++)
            if(nums[i] + nums[j] == target)
                return {i,j};
    // vector可以用{}初始化, 等同于
    // vector<int>v;
    // v.push_back(i);
    // v.push_back(j);
    // return v;
    return {}; // 题目说了一定可以找到答案, 所以上面循环中肯定可以return答案
    // 但是系统会预先检查语法问题, 当前这个函数需要有返回值, 所以
    // 这个地方虽然return个vector就行
}
```

方法二：哈希表

```
vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int, int>hash; //记录某个数字是否出现过了
    // 如果出现过了, 就记录当前的数字的下标
    // eg hash[num] = c 表示num这个值出现过了, 其下标为c
    for(int i = 0; i < nums.size(); i++){
        int res = target - nums[i];
        // 如果要选择当前的数字nums[i]作为答案中的一个, 就要判断下之前是否已经出现过res
        // 如果出现过, 那么res + nums[i] = target
        if(hash.count(res)){ //判断res是否出现过, 如果出现过, hash.count(res) = 1, 否则为0
            return {hash[res], i};
        }
        hash[nums[i]] = i; //记录当前的值的索引
    }
    return {};
}
```

提交结果	执行用时	内存消耗	语言	提交时间	备注
通过	8 ms	10.6 MB	C++	2022/04/21 10:29	🚩 哈希表
通过	268 ms	9.9 MB	C++	2022/04/21 10:15	▶ 两重循环

结果可以看到运行时间, 也可以自己备注是什么方法, 方便今后复习。

我刚开始刷题前也是十分恐惧, 觉得应该把语言掌握的很好, 各个点都掌握好才能开始刷题, 但后来才知道, 其实没太大必要, 基本的语法会了, 其实就可以大胆的开始写了。编程的核心在于思维而不是语言的语法, 而思维这个东西是需要不断做题训练的, 而不是通过看别人的代码获取的, 所以自己动手去敲一敲效果好于看别人的, (哪怕是照着别人的代码自己动手敲一遍, 也会很有收获)。

leetcode 刷题时, 一道题自己想了 5 分钟没思路就直接看题解, 然后看懂题解, 自己照着敲一篇, 积累多了, 渐渐就知道什么类型的题目用什么方法了, 而且过程中看别人的代码, 也可以学习到很多东西, 语言语法等内容在实战的过程中就顺便复习了, 如果有的点忘了, 比如我一开始一直记不住二维 vector 初始化, 就直接百度就找答案, 不需要一定要记住所有的东西, 不会的点, 忘了的点直接查找就可以了。

上面的 c++ 语法我仅仅总结了常用的一些知识点 (严谨点: 应该是我目前常用的一些知识), 肯定会不全面, 好多知识也可能较为零散, 不成体系, 或者跳跃性较大。但我都尽可能写了较为详细的注释帮助理解, 希望能够对你有所帮助, 加油!