

## 目录

### 目录

使用C++刷算法的好处  
名称空间using namespace std的解释  
cin和cout输入输出  
关于C++的头文件  
C++的变量声明  
C++特有的bool变量  
C++特有的用const定义常量  
C++里面超好用的string类  
C++的结构体struct和C语言的结构体的区别  
C++的引用&和传值的区别  
C++ STL之动态数组vector的使用  
C++ STL之集合set的使用  
C++ STL之映射map的使用  
C++ STL之栈stack的使用  
C++ STL之队列queue的使用  
C++ STL之unordered\_map和unordered\_set的使用  
C++中的sort函数  
C++中使用sort自定义cmp函数  
关于C++11的解释  
C++11里面很好用的auto声明  
C++11特性中的to\_string  
C++11特性中的stoi、stod  
如何在Dev-Cpp中使用C++11中的函数

### 使用C++刷算法的好处

- 在已经学习过C语言的前提下，学习C++并使用它刷算法的学习成本非常低～只需要几个小时就可以学会～
  - C++向下兼容C，C语言里面的语法完全可以在C++文件中运行，所以学习C++对刷算法时编程语言的表达能力进行扩充有益无害，例如C语言的输入输出（scanf和printf）比C++快，那么就可以在使用C++刷算法同时使用scanf和printf提高代码运行效率
  - C++拥有丰富的STL标准模板库，这也是PAT甲级、LeetCode等题目中经常需要用到的，单纯使用C语言解决问题会比C++的STL解决该问题麻烦很多～
  - C++的string超级好用～比C语言里面的char数组好用多啦～用了就再也不想回去的那种～
  - C++可以在某一变量使用前随时定义该变量，非常方便
  - 在解决一些较为简单的PAT乙级题目的时候（例如一些时间复杂度限制不严格的题目），cin、cout输入输出非常方便～用过的都说好～(๑•. •๑)
-

虽然C++是一门面向对象语言，但是对于刷算法这件事而言，我们并不需要掌握它面向对象的部分～只需要掌握刷算法的时候需要用到的部分（基本输入输出、STL标准模板库、string字符串等）就可以啦～C语言和C++有很多相似之处，且C++向下兼容C语言，所以我没有说的地方就直接用C语言的语法表示就好～以下是正文，先来段代码方便讲解：

```
1  #include <iostream>
2  using namespace std;
3  int main() {
4      int n;
5      cin >> n;
6      cout << "hello, liuchuo" << n + 1 << endl;
7      return 0;
8  }
```

### 名称空间using namespace std的解释

这句话是使用“std”这个名称空间（namespace）的意思～因为有的时候不同厂商定义的函数名称彼此之间可能会重复，为了避免冲突，就给所有的函数都封装在各自的名称空间里面，使用这个函数的时候就在main函数前面写明用了什么名称空间，几乎在C++中使用到的一些方法如cin、cout都是在std名称空间里面的，所以可以看到using namespace std;这句话几乎成了我每段C++代码的标配，就和return 0;一样必须有～其实也可以不写这句话，但是使用std里面的方法的时候就会麻烦点，要写成这样：

```
1  std::cin >> n;
2  std::cout << "hello, liuchuo" << n + 1 << endl;
```

我觉得这样比较丑，所以不管要不要用到，直接每道题的代码标配得写using namespace std;就好啦～

### cin和cout输入输出

就如同scanf和printf在stdio.h头文件中一样，cin和cout在头文件iostream里面，看名字就知道，io是输入输出input和output的首字母，stream是流，所以这个iostream头文件里包含的方法就是管理一些输入输出流的～

cin 和 cout 比较方便，不用像C语言里的scanf、printf那样写得那样繁琐，`cin >> n;` 和 `scanf("%d", &n);` 一样的意思，注意cin是向右的箭头，表示将内容输入到n中～

同样，`cout << n;` 和 `printf("%d", n);` 一样的意思，此时cout是向左的两个箭头，注意和cin区分开来～

而且不管n是double还是int或者是char类型，只用写 `cin >> n;` 和 `cout << n;` 这样简单的语句就好，不用像C语言中需要根据n的类型对应地写%lf、%d、%c这样麻烦～

`endl` 和 `"\n"` 是一个意思，一般如果前面是个字符串引号的话直接 `"\n"` 比较方便，如果是变量之类的我觉得写endl会比较好看～

```
1  cout << "hello, 小可爱～\n";
2  cout << n << endl;
```

cin和cout虽然使用起来更方便，但是输入输出的效率不如scanf和printf快，所以如果是做PAT乙级里面那种简单、对时间复杂度要求不高的题目，直接用cin和cout会觉得写起来比较省事；如果题目对时间复杂度要求比较高，全都改成scanf和printf可以提高代码的输入输出效率，比如有的时候发现用cin、cout做题目超时了，改成scanf和printf就AC了～

## 关于C++的头文件

C++的头文件一般是沒有像C语言的.h这样的扩展后缀的，一般情况下C语言里面的头文件去掉.h然后在前面加个c就可以继续在C++文件中使用c语言头文件中的函数啦～比如：

```
1 #include <cmath> // 相当于C语言里面的#include <math.h>
2 #include <cstdio> // 相当于C语言里面的#include <stdio.h>
3 #include <cctype> // 相当于C语言里面的#include <ctype.h>
4 #include <cstring> // 相当于C语言里面的#include <string.h>
```

## C++的变量声明

C语言的变量声明一般都在函数的开头，但是C++在首次使用变量之前声明即可～（当然也可以都放在函数的开头），而且一般C语言里面会在for循环的外面定义i变量，但是C++里面可以在for循环内部定义～（关于这点，VC++6.0里面可能会发现代码复制进去编译不通过，这是因为这个编译器太老啦，建议不要用这么上古的编译器啦～）而且在for循环里面定义的局部变量，在循环外面就失效啦（就是脱离这个局部作用域就会查无此变量的意思），所以一个main函数里面可以定义好多次局部变量i，再也不用担心写的循环太多变量名i、j、k不够用啦～

```
1 #include <iostream>
2 using namespace std;
3 int main() {
4     int n;
5     cin >> n;
6     cout << "hello, liuchuo" << n + 1 << endl;
7     int m;
8     cin >> m;
9     for (int i = 0; i < n; i++) { // 这个i只在for循环里面有用，出了这个for循环就相当于不见了
10         cout << i;
11     }
12     for (int i = 0; i < m; i++) { // 又可以定义一个i啦，和上面那个i不会冲突～
13         cout << i + 2;
14     }
15     return 0;
16 }
```

## C++特有的bool变量

bool变量有两个值，false和true，以前用C语言的时候都是用int的0和1表示false和true的，现在C++里面引入了这个叫做bool（布尔）的变量，而且C++把所有非零值解释为true，零值解释为false～所以直接赋值一个数字给bool变量也是可以的～它会自动根据int值是不是零来决定给bool变量赋值true还是false～

```
1 | bool flag = true;
2 | bool flag2 = -2; // flag2为true
3 | bool flag3 = 0; // flag3为false
```

## C++特有的用const定义常量

之前C语言里面会用#define定义常量，但是C++里面用const这个限定符定义常量，这样做有个好处就是可以定义常量的类型，比如int类型的常量a这样定义：

```
1 | const int a = 99999999;
```

## C++里面超好用的string类

以前用char[]的方式处理字符串很繁琐，现在有了string类，定义、拼接、输出、处理都更加简单啦～不过string只能用cin和cout处理，无法用scanf和printf处理：

```
1 | string s = "hello world"; // 赋值字符串
2 | string s2 = s;
3 | string s3 = s + s2; // 字符串拼接直接用+号就可以
4 | string s4;
5 | cin >> s4; // 读入字符串
6 | cout << s; // 输出字符串
```

用cin读入字符串的时候，是以空格为分隔符的，如果想要读入一整行的字符串，就需要用getline～

s的长度可以用s.length()获取～（有几个字符就是长度多少，不存在char[]里面的什么末尾的结束符之类的～）

```
1 | string s; // 定义一个空字符串s
2 | getline(cin, s); // 读取一行的字符串，包括空格
3 | cout << s.length(); // 输出字符串s的长度
```

string中还有个很常用的函数叫做substr，作用是截取某个字符串中的子串，用法有两种形式：

```
1 | string s2 = s.substr(4); // 表示从下标4开始一直到结束
2 | string s3 = s.substr(5, 3); // 表示从下标5开始，3个字符
```

## C++的结构体struct和C语言的结构体的区别

定义好结构体stu之后，使用这个结构体类型的时候，C语言需要写关键字struct，而C++里面可以省略不写：

```

1 struct stu {
2     int grade;
3     float score;
4 };
5 struct stu arr1[10]; // C语言里面需要写struct
6 stu arr2[10]; // C++里面不用写

```

## C++的引用&和传值的区别

这个引用符号&要和C语言里面的取地址运算符&区分开来，他们没有什么关系，C++里面的引用是指在变量名之前加一个&符号，比如在函数传入的参数中int &a，那么对这个引用变量a做的所有操作都是直接对传入的原变量进行的操作，并没有像原来int a一样只是拷贝一个副本（传值），举两个例子：

```

1 void func(int &a) { // 传入的是n的引用，相当于直接对n进行了操作，只不过在func函数
    中换了个名字叫a
2     a = 99;
3 }
4 int main() {
5     int n = 0;
6     func(n); // n由0变成了99
7 }

```

```

1 void func(int a) { // 传入的是0这个值，并不会改变main函数中n的值
2     a = 99;
3 }
4 int main() {
5     int n = 0;
6     func(n); // 并不会改变n的值，n还是0
7 }

```

## C++ STL之动态数组vector的使用

之前C语言里面用int arr[]定义数组，它的缺点是数组的长度不能随心所欲的改变，而C++里面有一个能完全替代数组的动态数组vector，它能够在运行阶段设置数组的长度、在末尾增加新的数据、在中间插入新的值、长度任意被改变，很好用～它在头文件vector里面，也在命名空间std里面，所以使用的时候要引入头文件vector和using namespace std;

vector、stack、queue、map、set这些在C++中都叫做容器，这些容器的大小都可以用 .size() 获取到，就像string s的长度用 s.length() 获取一样～（string其实也可以用 s.size()，不过对于vector、stack、queue、map、set这样的容器我们一般讨论它的大小size，字符串一般讨论它的长度length～其实string里面的size和length两者是没有区别、可以互换使用的，比如我之前写过一篇博客《C++：string类中size()和length()的区别》，最终的结论就是两者没有区别，里面对官方文档进行了详细阐述，有兴趣的可以去看一下：<https://www.liuchuo.net/archives/2013>）

```

1  #include <iostream>
2  #include <vector>
3  int main() {
4      vector<int> v1; // 定义一个vector v1, 定义的时候没有分配大小
5      cout << v1.size(); // 输出vector v1的大小, 此处应该为0
6      return 0;
7  }

```

vector可以一开始不定义大小，之后用resize分配大小，也可以一开始就定义大小，之后还可以对它插入删除动态改变它的大小～而且不管在main函数里还是在全局中定义，它都能够直接将所有的值初始化为0（不用显式地写出来，默认就是所有的元素为0），再也不用担心C语言里面出现的那种int arr[10];结果忘记初始化为0导致的各种bug啦～

```

1  vector<int> v(10); // 直接定义长度为10的int数组, 默认这10个元素值都为0
2
3  // 或者
4  vector<int> v1;
5  v1.resize(8); //先定义一个vector变量v1, 然后将长度resize为8, 默认这8个元素都是0
6
7  // 在定义的时候就可以对vector变量进行初始化
8  vector<int> v3(100, 9); // 把100长度的数组中所有的值都初始化为9
9
10 // 访问的时候像数组一样直接用[]下标访问即可～(也可以用迭代器访问, 下面会讲～)
11 v[1] = 2;
12 cout << v[0];

```

不管是vector、stack、queue、map还是set都有很多好用的方法，这些方法都可以在[www.cplusplus.com](http://www.cplusplus.com)官方网站中直接查询官方文档，上面有方法的讲解和代码示例～官方文档是刷题时候必不可少的好伙伴～

比如进入官网搜索 `vector`，就会出现vector拥有的所有方法，点进去一个方法就能看到这个方法的详细解释和代码示例～当然我们平时写算法用不到那么多方法啦，只有几个是常用的～以下是一些常用的vector方法：

```

1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  int main() {
5      vector<int> a; // 定义的时候不指定vector的大小
6      cout << a.size() << endl; // 这个时候size是0
7      for (int i = 0; i < 10; i++) {
8          a.push_back(i); // 在vector a的末尾添加一个元素i
9      }
10     cout << a.size() << endl; // 此时会发现a的size变成了10
11     vector<int> b(15); // 定义的时候指定vector的大小, 默认b里面元素都是0
12     cout << b.size() << endl;
13     for (int i = 0; i < b.size(); i++) {
14         b[i] = 15;

```

```

15     }
16     for (int i = 0; i < b.size(); i++) {
17         cout << b[i] << " ";
18     }
19     cout << endl;
20     vector<int> c(20, 2); // 定义的时候指定vector的大小并把所有的元素赋一个指定
    的值
21     for (int i = 0; i < c.size(); i++) {
22         cout << c[i] << " ";
23     }
24     cout << endl;
25     for (auto it = c.begin(); it != c.end(); it++) { // 使用迭代器的方式访
    问vector
26         cout << *it << " ";
27     }
28     return 0;
29 }

```

容器vector、set、map这些遍历的时候都是使用迭代器访问的，`c.begin()`是一个指针，指向容器的第一个元素，`c.end()`指向容器的最后一个元素的后一个位置，所以迭代器指针`it`的for循环判断条件是`it != c.end()`

访问元素的值要对`it`指针取值，要在前面加星号～所以是`cout << *it;`

这里的`auto`相当于 `vector<int>::iterator` 的简写，关于`auto`下文有讲解～

## C++ STL之集合set的使用

set是集合，一个set里面的各元素是各不相同的，而且set会按照元素进行从小到大排序～以下是set的常用用法：

```

1  #include <iostream>
2  #include <set>
3  using namespace std;
4  int main() {
5      set<int> s; // 定义一个空集合s
6      s.insert(1); // 向集合s里面插入一个1
7      cout << *(s.begin()) << endl; // 输出集合s的第一个元素（前面的星号表示要对
    指针取值）
8      for (int i = 0; i < 6; i++) {
9          s.insert(i); // 向集合s里面插入i
10     }
11     for (auto it = s.begin(); it != s.end(); it++) { // 用迭代器遍历集合s
    里面的每一个元素
12         cout << *it << " ";
13     }
14     cout << endl << (s.find(2) != s.end()) << endl; // 查找集合s中的值，如
    果结果等于s.end()表示未找到（因为s.end()表示s的最后一个元素的下一个元素所在的位置）
15     cout << (s.find(10) != s.end()) << endl; // s.find(10) != s.end()表
    示能找到10这个元素

```

```

16     s.erase(1); // 删除集合s中的1这个元素
17     cout << (s.find(1) != s.end()) << endl; // 这时候元素1就应该找不到啦~
18     return 0;
19 }

```

## C++ STL之映射map的使用

map是键值对，比如一个人名对应一个学号，就可以定义一个字符串string类型的人名为“键”，学号int类型为“值”，如 `map<string, int> m;`；当然键、值也可以是其它变量类型~map会自动将所有的键值对按照键从小到大排序，以下是map中常用的方法：

```

1  #include <iostream>
2  #include <map>
3  #include <string>
4  using namespace std;
5  int main() {
6      map<string, int> m; // 定义一个空的map m，键是string类型的，值是int类型的
7      m["hello"] = 2; // 将key为"hello"，value为2的键值对(key-value)存入map中
8      cout << m["hello"] << endl; // 访问map中key为"hello"的value，如果key不存在，则返回0
9      cout << m["world"] << endl;
10     m["world"] = 3; // 将"world"键对应的值修改为3
11     m[","] = 1; // 设立一组键值对，键为"," 值为1
12     // 用迭代器遍历，输出map中所有的元素，键用it->first获取，值用it->second获取
13     for (auto it = m.begin(); it != m.end(); it++) {
14         cout << it->first << " " << it->second << endl;
15     }
16     // 访问map的第一个元素，输出它的键和值
17     cout << m.begin()->first << " " << m.begin()->second << endl;
18     // 访问map的最后一个元素，输出它的键和值
19     cout << m.rbegin()->first << " " << m.rbegin()->second << endl;
20     // 输出map的元素个数
21     cout << m.size() << endl;
22     return 0;
23 }

```

## C++ STL之栈stack的使用

栈stack在头文件 `#include <stack>` 中，是数据结构里面的栈~以下是常用用法：



```

1  #include <iostream>
2  #include <stack>
3  using namespace std;
4  int main() {
5      stack<int> s; // 定义一个空栈s
6      for (int i = 0; i < 6; i++) {
7          s.push(i); // 将一个元素压入栈s中
8      }
9      cout << s.top() << endl; // 访问s的栈顶元素
10     cout << s.size() << endl; // 输出s的元素个数
11     s.pop(); // 移除栈顶元素
12     return 0;
13 }

```

## C++ STL之队列queue的使用

队列queue在头文件 `#include <queue>` 中，是数据结构里面的队列～以下是常用用法：

```

1  #include <iostream>
2  #include <queue>
3  using namespace std;
4  int main() {
5      queue<int> q; // 定义一个空队列q
6      for (int i = 0; i < 6; i++) {
7          q.push(i); // 将i的值依次压入队列q中
8      }
9      cout << q.front() << " " << q.back() << endl; // 访问队列的队首元素和队
尾元素
10     cout << q.size() << endl; // 输出队列的元素个数
11     q.pop(); // 移除队列的队首元素
12     return 0;
13 }

```

## C++ STL之unordered\_map和unordered\_set的使用

unordered\_map在头文件 `#include <unordered_map>` 中，unordered\_set在头文件 `#include <unordered_set>` 中～

unordered\_map和map（或者unordered\_set和set）的区别是，map会按照键值对的键key进行排序（set里面会按照集合中的元素大小进行排序，从小到大顺序），而unordered\_map（或者unordered\_set）省去了这个排序的过程，如果偶尔刷题时候用map或者set超时了，可以考虑用unordered\_map（或者unordered\_set）缩短代码运行时间、提高代码效率～至于用法和map、set是一样的～

## C++中的sort函数

sort函数在头文件 `#include <algorithm>` 里面，主要是对一个数组进行排序（int arr[]数组或者vector数组都行），vector是容器，要用v.begin()和v.end()表示头尾；而int arr[]用arr表示数组的首地址，arr+n表示尾部～

```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  using namespace std;
5  bool cmp(int a, int b) { // cmp函数返回的值是bool类型
6      return a > b; // 从大到小排列
7  }
8  int main() {
9      vector<int> v(10);
10     for (int i = 0; i < 10; i++) {
11         cin >> v[i];
12     }
13     sort(v.begin(), v.end()); // 因为这里没有传入参数cmp, 所以按照默认, v从小到大排列
14
15     int arr[10];
16     for (int i = 0; i < 10; i++) {
17         cin >> arr[i];
18     }
19     sort(arr, arr + 10, cmp); // arr从大到小排列, 因为cmp函数排序规则设置了从大到小
20     return 0;
21 }

```

## C++中使用sort自定义cmp函数

sort默认是从小到大排列的, 也可以指定第三个参数cmp函数, 然后自己定义一个cmp函数指定排序规则~cmp最好用的还是在结构体中, 尤其是很多排序的题目~比如一个学生结构体stu有学号和成绩两个变量, 要求如果成绩不同就按照成绩从大到小排列, 如果成绩相同就按照学号从小到大排列, 那么就可以写一个cmp函数实现这个看上去有点复杂的排序过程:

```

1  #include <iostream>
2  using namespace std;
3  struct stu { // 定义一个结构体stu, number表示学号, score表示分数
4      int number;
5      int score;
6  }
7  bool cmp(stu a, stu b) { // cmp函数, 返回值是bool, 传入的参数类型应该是结构体stu类型
8      if (a.score != b.score) // 如果学生分数不同, 就按照分数从大到小排列
9          return a.score > b.score;
10     else // 如果学生分数相同, 就按照学号从小到大排列
11         return a.number < b.number;
12 }
13
14 // 有时候这种简单的if-else语句我喜欢直接用一个C语言里面的三目运算符表示~
15 bool cmp(stu a, stu b) {
16     return a.score != b.score ? a.score > b.score : a.number <
17     b.number;
18 }

```

## 关于C++11的解释

C++11是2011年官方为C++语言带来的新语法新标准，C++11为C++语言带来了很多好用的新特性，比如auto、to\_string()函数、stoi、stof、unordered\_map、unordered\_set之类的～现在大多数OJ都是支持C++11语法的，有些编译器在使用的时候需要进行一些设置才能使用C++11中的语法，否则可能会导致编译器上编译不通过无法运行，比如我曾经写过一篇博客《如何在Dev-Cpp中使用C++11中的函数》（在本教程末尾）这个是针对DEV-cpp编译器的，其他的编译器如果发现不支持也可以百度搜索一下让编译器支持C++11的方法～总之C++11的语法在OJ里面是可以使用的～而且很多语法很好用～以下讲解一些C++11里面常用的新特性～

## C++11里面很好用的auto声明

auto是C++11里面的新特性，可以让编译器根据初始值类型直接推断变量的类型。比如这样：

```
1 auto x = 100; // x是int变量
2 auto y = 1.5; // y是double变量
```

当然这个在算法里面最主要的用处不是这个，而是在STL中使用迭代器的时候，auto可以代替一大长串的迭代器类型声明：

```
1 // 本来set的迭代器遍历要这样写：
2 for(set<int>::iterator it = s.begin(); it != s.end(); it++) {
3     cout << *it << " ";
4 }
5 // 现在可以直接替换成这样的写法：
6 for(auto it = s.begin(); it != s.end(); it++) {
7     cout << *it << " ";
8 }
```

## C++11特性中的to\_string

to\_string的头文件是 `#include <string>`，to\_string最常用的就是把一个int型变量或者一个数字转化为string类型的变量，当然也可以转double、float等类型的变量，这在很多PAT字符串处理的题目中很有用处，以下是示例代码：

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main() {
5      string s1 = to_string(123); // 将123这个数字转成字符串
6      cout << s1 << endl;
7      string s2 = to_string(4.5); // 将4.5这个数字转成字符串
8      cout << s2 << endl;
9      cout << s1 + s2 << endl; // 将s1和s2两个字符串拼接起来并输出
10     printf("%s\n", (s1 + s2).c_str()); // 如果想用printf输出string, 得加一
    ↑.c_str()
11     return 0;
12 }

```

## C++11特性中的stoi、stod

使用stoi、stod可以将字符串string转化为对应的int型、double型变量，这在字符串处理的很多问题中很有帮助～以下是示例代码和非法输入的处理方法：

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4  int main() {
5      string str = "123";
6      int a = stoi(str);
7      cout << a;
8      str = "123.44";
9      double b = stod(str);
10     cout << b;
11     return 0;
12 }

```

stoi如果遇到的是非法输入（比如stoi("123.4")，123.4不是一个int型变量）：

1.会 **自动截取最前面的数字，直到遇到不是数字为止**

(所以说如果是浮点型，会截取前面的整数部分)

2.如果最前面不是数字，会运行时发生错误

stod如果是非法输入：

1.会自动截取最前面的浮点数，直到遇到不满足浮点数为止

(所以说如果是浮点型，会截取前面的整数部分)

2.如果最前面不是数字或者小数点，会运行时发生错误

3. **如果最前面是小数点，会自动转化后在前面补0**

不仅有stoi、stod两种，相应的还有：

stof (string to float)

stold (string to long double)

stol (string to long)

stoll (string to long long)

stoul (string to unsigned long)

stoull (string to unsigned long long)

### 如何在Dev-Cpp中使用C++11中的函数

如果想要在Dev-Cpp里面使用C++11特性的函数，比如刷算法中常用的stoi、to\_string、unordered\_map、unordered\_set、auto这些，需要在设置里面让dev支持c++11～需要这样做～

在工具-编译选项-编译器-编译时加入这个命令“-std=c++11”即可～

感谢阅读 (๑•. •๑) 么么哒～