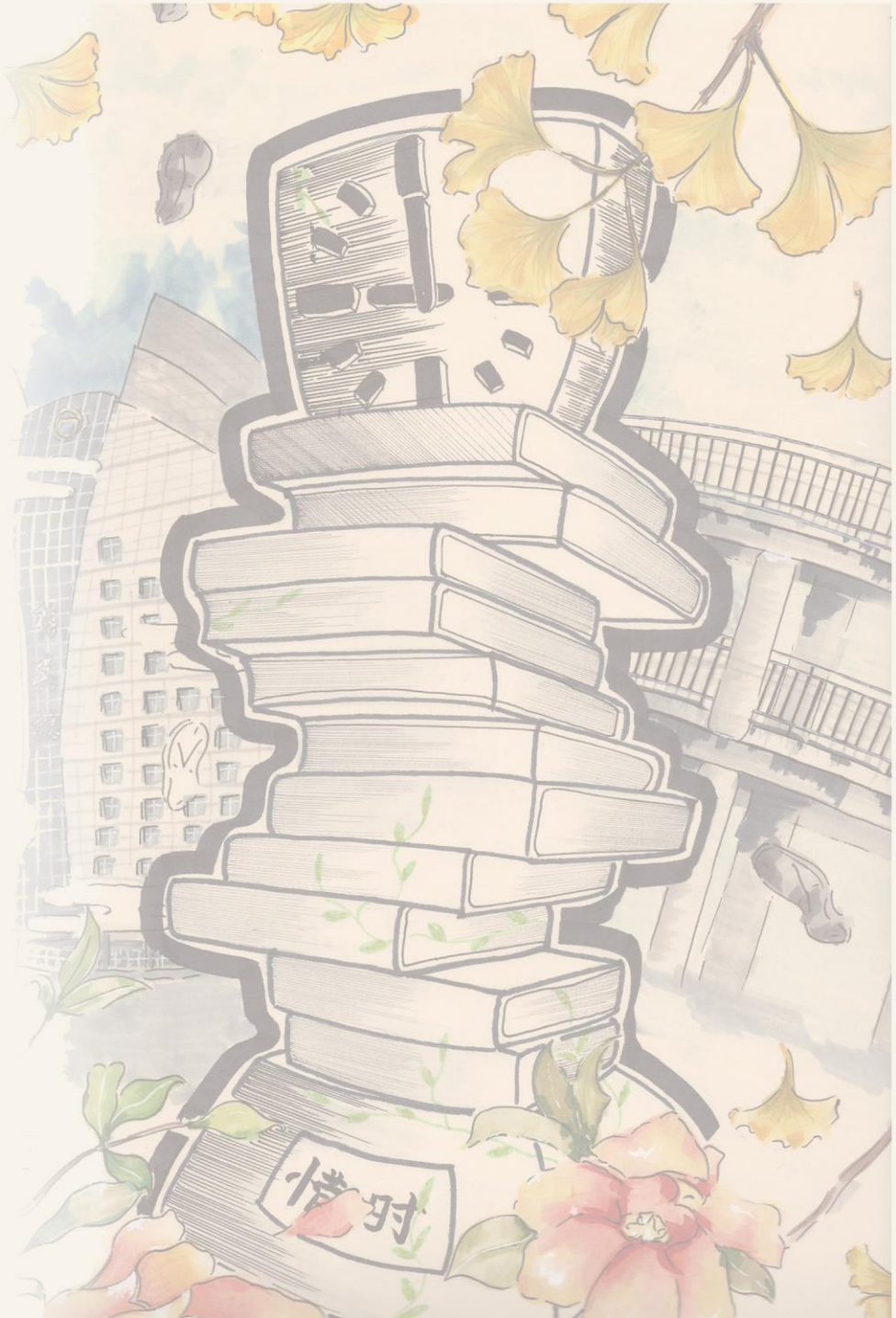
代码习惯

福州大学 任宝硕

2024 年 8 月 17 日



石家庄二中 实验学校
The Shiyan School Attached To Shijiazhuang No.2 Middle School

关于讲师

- 24届二南毕业生，21届石门（现四十八中）毕业生
- CSP 2022 提高组 一等奖
- OIerDb 主要开发者 (https://oier.baoshuo.dev)
- LibreOJ 管理员
- UniversalOJ 社区版现任维护者
- HydroOJ 管理组成员（潜水）
- 你们的讲师

- 个人网站：https://baoshuo.ren
- OI 博客：https://oi.baoshuo.ren
- 洛谷账号：宝硕

课前须知

- 本次课程内容主观性较强，由于讲师水平有限，请大佬们轻喷。
- 「磨刀不误砍柴工」，希望大家能够认真听讲。
- 本课程中所展示代码示例大多来自同学们的实际提交，请同学们务必自查自纠、认真对待。

布局整体

大局观

引用合适的头文件

你是否习惯于 <bits/stdc++.h> 一把梭?

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

using namespace std;
```

亦或是每次不管用上用不上都 include 上一堆头文件?

```cpp
#include <stdlib.h>
#include <string.h>
#include  <stdio.h>
#include   <math.h>
#include   <time.h>
#include   <vector>
#include   <bitset>
#include    <queue>
#include      <set>
#include      <map>
#include<algorithm>
#include <iostream>
```

引用合适的头文件

遵循「头文件最少化」原则，用到什么函数就引用对应的头文件。

知道自己的代码需要什么是非常重要的。

头文件	包括的方法
<iostream>	cin, cout 等（输入输出相关）
<fstream>	ifstream, ofstream 等（文件读写相关）
<iomanip>	std::fixed, std::setprecision 等（输入输出修饰相关）
<algorithm>	sort, unique, lower_bound, upper_bound 等函数
<vector>	STL vector 容器
<stack>	STL 栈
<queue>	STL 队列
…	…

要不要 using namespace std?

可以用，但不推荐。

C++ 标准库里面的函数众多，都装在 std 这个命名空间下面。

容易和自己的函数名、变量名撞车。比如 min 和 std::min、max 和 std::max，这些都是常见问题。

可以把一些常用的方法通过诸如 `using std::cin;` 的形式引入来方便编程。

```cpp
#include <iostream>

using std::cin;
using std::cout;
const char endl = '\n';

int a, b;

int main() {
    std::ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> a >> b;

```

对主要常、变量进行集中定义

大家在写代码的时候都需要开一些变量来存储一些值。

有些同学在开变量的时候非常随意：名称乱写、大小乱定、地方乱扔。

前两点暂且先放下，光是不注意最后一点就会导致在要修改、检查的时候找不到变量，无从下手。

代码短的情况还能驾驭，那么像最右边这张图上的情况呢？

```
1  #include<iostream>
2  using namespace std;
3  int maxn=1090910;
4  int a,b,c;
5  int main()
6  {
7      cin>>a>>b>>c;
8      int d=a+b,r;
9      cin>>r;
10     for(int i=1;i<=r;i++)
11     {
12         int x,y;
13         cin>>x>>y;
14         d+=x;
15         d+=y;
16     }
17     cout<<d;
18     return 0;
19 }
20
```

对主要常、变量进行集中定义

- 因此我推荐在 using 语句之后、程序主体的函数之前对主要常、变量进行集中定义。
- 另：对于循环、函数内的临时变量，在用到它们的地方之前定义即可。

```
5   using std::cin;
6   using std::cout;
7   const char endl = '\n';
8
9   const int N = 1e5 + 5;
10
11  int n, m, k, l, a[N], b[N];
12  std::vector<int> g[N];
13  int dfn[N], low[N];
14  bool bridge[N];
15  std::vector<std::pair<int, int>> ans;      You, 17个月前 •
16
17  void tarjan(int u, int f) {
18      static int cnt = 0;
19
```

常量区

变量区

正确缩进代码

- 正确的代码缩进有助于提高可读性。推荐使用 4 个空格进行缩进，大括号不换行。
- 要适当地加空格。

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
using namespace std;
long long N,M,mid,sum=0,l=0,ans,a[1000001];
int main()
{
    scanf("%d%d",&N,&M);
    for(int i=1;i<=N;i++)
    {
        scanf("%d",&a[i]);
        ans=max(ans,a[i]);
}
while(l<=ans)
{
    mid=(l+ans)/2;
    sum=0;
    for(int i=1;i<=N;i++)
    {
        if(a[i]>mid)
        {
            sum+=a[i]-mid;
        }
    }
    if(sum<M)
        ans=mid-1;
        else
        l=mid+1;
}
printf("%d",l-1);
return 0;
}
```

```cpp
#include <cstdio>
#include <iostream>
#include <algorithm>
using namespace std;
long long N, M, mid, sum = 0, l = 0, ans, a[1000001];

int main() {
    scanf("%d%d", &N, &M);
    for (int i = 1; i <= N; i++) {
        scanf("%d", &a[i]);
        ans = max(ans, a[i]);
    }
    while (l <= ans) {
        mid = (l + ans) / 2;
        sum = 0;
        for (int i = 1; i <= N; i++) {
            if (a[i] > mid) {
                sum += a[i] - mid;
            }
        }
        if (sum < M)
            ans = mid - 1;
        else
            l = mid + 1;
    }
    printf("%d", l - 1);
    return 0;
}
```

我们真的需要快读吗？

大多数情况下，答案其实是否定的。

- 关闭流同步的 cin/cout 已经足够快速，能够与 scanf/printf/普通快读 媲美。
- 快读代码编写时易出错、局限性大（只支持整数）。

关闭流同步的 cin/cout 有多快?

我之前在《C++ 输入输出的速度优化与实际测试》一文中已经给出了测试结果。

https://oi.baoshuo.ren/cpp-io-speed-optimization/

小结

- 这里给出一个框架：

AcWing > 2488 > 2488.cpp > ...

```
1   #include <iostream>      You, 24个月前 • 2488. 树套树-简单版 …
2   #include <limits>
3   #include <set>
4
5   using std::cin;
6   using std::cout;
7   const char endl = '\n';
8
9   const int N = 5e4 + 5;
10
11  int n, m, a[N];
12
13  struct node : public std::multiset<int> {…
24  } tr[N << 2];
25
26  void build(int u, int l, int r) {…
40
41  void modify(int u, int p, int x) {…
52
53  int query(int u, int l, int r, int x) {…
66
67  int main() {…
102
```

头文件区

常量区

变量区

方法定义区

功能函数区

主函数

规范命名

要能让人看得懂

命名很重要！

`nl`，`n1` 都是一个合法的变量名。

但是如果粗略的扫一眼，你能分清这两个变量吗？

命名很重要！

二分时候的 check 函数，有人会这样写：

```
7   bool panduan(int shu) {
8       // 一些代码
9   }
10
```

那么请问，tudoupeijizhi 是什么？

命名很重要！

变量命名时应该使用清晰的、有指向性的词汇来提高可读性。

如果遇到常用词汇的话可以缩写。

如果是题目里面给定的名称可以直接用，比如 a, b, c 等等。

好的：max_height, min_width

坏的：mh, mw

常见命名缩写对照（一）

缩写	原始单词	含义 / 注释
ans	answer	答案
arr	array	数组
ch	child / children	（有根树中的）儿子
cmp / comp	compare	比较（常用于 sort）
cnt	count	计数
cur	current	当前的
del	delete	删除
dis / dist	distance	距离
eps	ε (epsilon)	（浮点数中）精度设置
fa	father	（有根树中的）父亲
g	graph	图
i, j, k		各种下标

常见命名缩写对照（二）

缩写	原始单词	含义 / 注释
INF	infinity	无穷（常取 0x3f3f3f3f）
init	initialize	初始化
ins	insert	插入
inv	inverse	逆元
l, r, mid	left, right, middle	左、右、中
len	length	长度
mod	modulo	取模
mul	multiply	乘
num	number	数字
nxt	next	下一个
pi	π	
Pos	position	位置

常见命名缩写对照（三）

缩写	原始单词	含义 / 注释
pre	prefix	前缀
pre / prev	previous	上一个
ptr	pointer	指针
rnd / rand	random	随机
ref	reference	引用
res	result	结果
rev	reverse	反转
s, t	source, target	源点、汇点（图论中常见）
seg	segment	线段
seq	sequence	序列
sqrt	square root	根号
tmp	temporary	临时的

常见命名缩写对照（四）

缩写	原始单词	含义 / 注释
tot	total	总计
u, v, w		（图论）起点、终点、边权
val	value	值

注意细节

差之毫厘，谬以千里

运算符优先级

优先级	类别	运算符	结合律
1	逗号运算符	,	从左到右
2	赋值运算符	= += -= *= /= %= >>= <<= &= ^= |=	从右到左
3	逻辑或	||	从左到右
4	逻辑与	&&	从左到右
5	按位或	|	从左到右
6	按位异或	^	从左到右
7	按位与	&	从左到右
8	相等/不等	== !=	从左到右
9	关系运算符	< <= > >=	从左到右
10	位移运算符	<< >>	从左到右
11	加法/减法	+ -	从左到右
12	乘法/除法/取余	* （乘号） / %	从左到右
13	单目运算符	! * （指针） & ++ - + （正号） - （负号）	从右到左
14	后缀运算符	( ) [ ] ->	从左到右

注意细节

多个 else if 的时候不要丢掉 else。

```
if (cond1) {
    // ...
} else if (cond2) {
    // ...
} if (cond3) {
    // ...
} else {
    // ...
}
```

注意细节

- 不要丢落。

~福大24计科1班

请问你有没有遇到过做do-while循环的时候，小黑窗输入了一个数字然后再也没有反应，回车也没有，甚至小黑窗还要手动杀进程

```c
#include <stdio.h>

int main()
{
    //求最数平均数
    int number;
    int count = 0;
    int sum = 0;

    do{
        scanf("%d",&number);
        if(number != -1){
            sum += number;
            count++;
        }

    }while(number != -1);

    printf("%f",1.0*sum/count);

    return 0;
}
```

这是我的代码

注意细节

- 不要敲错。

~福大24计科1班

```
13
14  do{
15      printf("请你猜猜这个数~");
16      scanf("&d",&a);
17      count ++;
18
19
20      if(number > a){
21      printf("对不起,你才猜数字小了!");
22      }else if(number < a){
23      printf("对不起,你才的数字猜大了!");
24      }
25  }while(a != number);
26
27  printf("恭喜你,你用了%d就猜到了答案~\n",count);
28
29  return 0;
30
31  }
```

可以再帮我看看这个吗

最后死循环了

scanf 里面是 %，不是 &

写好注释

注释不宜多，但求必要。

```
52
53          f[0] = 0;
54          for (int i = 0; i < 1 << n; i++) {
55              // 两只及以上      You, 2年前 • 524. 愤怒的小鸟 …
56              for (int j = 1; j <= cnt; j++) {
57                  f[i | s[j]] = std::min(f[i | s[j]], f[i] + 1);
58              }
59              // 一只
60              for (int j = 0; j < n; j++) {
61                  f[i | 1 << j] = std::min(f[i | 1 << j], f[i] + 1);
62              }
63          }
64
65          cout << f[(1 << n) - 1] << endl;
66      }
```

保持简洁

去繁就简

避免无谓的重复

举个例子：有的时候需要对两个不同的数组进行相同的操作。

这时不需要将代码复制两遍来完成这项工作，而应该采用右图的方式来将其抽离为单独的函数，便于维护。

```
12  int n1, n2, m, dist1[N], dist2[N];
13  bool vis[N];
14  std::vector<int> g[N];
15
16  void bfs(int s, int* dist) {      You, 12个月前 • D - Add One Edge
36
37  int main() {
38      std::ios::sync_with_stdio(false);
39      cin.tie(nullptr);
40
41      cin >> n1 >> n2 >> m;
42
43      for (int i = 1, u, v; i <= m; i++) {
44          cin >> u >> v;
45
46          g[u].emplace_back(&: v);
47          g[v].emplace_back(&: u);
48      }
49
50      bfs(1, dist1);
51      bfs(n1 + n2, dist2);
52
53      cout << *std::max_element(dist1 + 1, dist1 + 1 + n1)
```

精简函数

- 让一个函数只做一件事。

不做无用功

```
private bool IsEven(int number){
    if (number == 1) return false;
    else if (number == 2) return true;
    else if (number == 3) return false;
    else if (number == 4) return true;
    else if (number == 5) return false;
    else if (number == 6) return true;
    else if (number == 7) return false;
    else if (number == 8) return true;
    else if (number == 9) return false;
    else if (number == 10) return true;
    else if (number == 11) return false;
    else if (number == 12) return true;
    else if (number == 13) return false;
    else if (number == 14) return true;
    else if (number == 15) return false;
    else if (number == 16) return true;
    else if (number == 17) return false;
    else if (number == 18) return true;
    else if (number == 19) return false;
    else if (number == 20) return true;
    else if (number == 21) return false;
    else if (number == 22) return true;
```

别用火车头

- 当心被禁赛。

```
#pragma GCC diagnostic error "-std=c++11"
#pragma GCC target("avx")
#pragma GCC optimize(3)
#pragma GCC optimize("Ofast")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-ffast-math")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-funroll-loops")
#pragma GCC optimize("-fwhole-program")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-fstrict-overflow")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-skip-blocks")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("-funsafe-loop-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
```

回头看看

与文化课同样地，检查很重要

检查自己的代码

再聪明的人也会犯错。

不要盲目自信。

很多前辈都死在了「不检查」上面。

回头看看

- 不得不提 Windows 系统下不区分大小写的问题，但 Linux 下可不是这样（其实和文件系统有关）。

SCSY_OJ 返回比赛列表 20240723普及模拟赛3 题目列表 比赛成绩表

✕ 0 Compile Error

```
foo.cc:1:9: fatal error: bits/stdC++.H: No such file or directory
    1 | #include<bits/stdC++.H>
      |          ^~~~~~~~~~~~~~
compilation terminated.
```

代码

```
1  #include<bits/stdC++.H>
2  using namespace std;
3  int main(void){
4      freopen("pow.in","r",stdin);
```

回头看看

- 数组大小?
- Yes 还是 YES?
- 答案让你输出什么?
- ……

一份好代码是什么样的？

众说纷纭。每个人都有自己的评判标准。

但毋庸置疑的是，一份好代码不仅可以被编写者读懂，也是可以被其他人读懂的。

谢谢大家

Stay Hungry, Stay Foolish

https://oi.baoshuo.ren/code-style-guide/