

第 1 篇

语言入门

通过本篇熟悉并精通 C/C++ 语言的语法和编码,这是程序员的基本功。合格的程序员在写代码时,对于编程语言的语法、简单逻辑、常用系统函数,能不假思索地写出,且基本不出错。

具体要精通哪些内容?翻开一本 C 语言的教材,包括数据类型、运算符、输入输出、简单字符处理、选择结构、循环结构、数组、函数、文件等;还有递推、递归、二进制位运算等,是极为重要的编码基本功。

有读者问,C 语言我学了,C++也要学吗?听说 C++很复杂。

答案是:C 语言是算法竞赛的基础,必须扎实掌握全部核心内容;C++无须全学,重点攻克标准模板库(Standard Template Library,STL)中与竞赛相关的部分即可。C++作为面向对象语言,支持类、对象、封装、继承、多态等特性,在工程应用中需要熟练掌握。不过,这些在算法竞赛的编程中几乎用不上。在算法竞赛中,关键的是 C++的标准模板库,其中的核心数据结构(队列、栈、链表、集合等)和基础算法(排序、查找等),能极大地简化编码流程、提升解题效率。本书的代码示例会完整展示这些核心技术的用法。

下面给出算法竞赛中常用的 STL 清单。

数据结构:vector、queue、stack、set/multiset、map/multimap、priority_queue、string、deque、bitset 等。

算法:sort、next_permutation、prev_permutation、find、lower_bound/upper_bound、reverse、count 等。

工具:pair、swap、max/min、fill 等。

提升编码能力没有捷径,唯一的办法就是大量实操。从“我会写代码”进化到“我就是代码”,才能在竞赛中稳扎稳打。

1.1

结构化程序设计



题 1-1 P1001 A+B Problem

题目描述:输入两个整数 a 和 b ,输出它们的和($|a| \leq 10^9, |b| \leq 10^9$)。

输入格式:两个以空格分开的整数。

输出格式:一个整数。

输入样例

20 30

输出样例

50

【考点】 输入输出格式、代码测试。

【思路 and 证明】 用这道题说明做算法题的基本要求,包括输入输出的处理、代码的测试方法。

下面详细说明输入输出格式要求、测试过程、测试结果、C 和 C++语言相关问题。

(1) 输入和输出。

这是一道经典例题,在很多 C 语言教材中会给出类似下面的代码。编译运行后,根据提示输入整数 a 和 b ,输出 $a+b$ 的结果。

```
1 #include <stdio.h>
2 int main(){
3     int a,b;
4     printf("请输入 a 和 b: ");
5     scanf("%d%d",&a,&b);
6     printf("%d\n", a + b);
7     return 0;
8 }
```

然而,把代码提交到洛谷网站,返回的判题结果是 10 个 WA,表示 10 个测试结果都是 Wrong Answer,全部错误。

错误原因:第 4 行的 printf 语句是多余的。因为在“题目描述”中并没有要求输出这个提示。在线评测系统会严格按照题目给定的输入输出格式进行校验,任何额外的输出都会导致判定错误。

根据题目的输入输出说明,输入时要严格按题目给定的格式进行输入,不要写任何多余的提示;输出时也不要写除了要求以外的任何内容。

删除第 4 行重新提交,网站返回 10 个 AC,表示 10 个测试结果都是 Accept,代码是正确的。

(2) 判题系统的测试过程。

在洛谷做题时提交代码,洛谷的判题系统会自动测试代码是否正确。

如何测试?是直接看代码吗?并不是,而是采用更简单的方法——黑盒测试法。判题系统给每道题准备了若干测试,每个测试包括输入数据和输出数据,这些输入和输出对应了题目要求的正确计算。例如,本题的输入是两个整数,那么可能的一个测试数据是:输入“84 7”,输出“91”。

提交代码后,判题系统会运行代码,并用这 10 个测试检查代码。例如,输入“84 7”,如果运行结果是“91”,那么通过了这个测试;如果运行结果是“93”,那么没有通过这个测试。如果能通过所有 10 个测试,返回了 10 个 AC,说明代码通过了系统的所有测试,这个代码被认为完全正确。如果只通过部分测试,说明代码还有缺陷。

除了 WA 和 AC,还可能有其他测试结果^①,洛谷的所有测试结果有以下几种。

- AC: Accept,程序通过。
- CE: Compile Error,编译错误。
- WA: Wrong Answer,答案错误。
- RE: Runtime Error,运行时错误。
- TLE: Time Limit Exceeded,超出时间限制。
- MLE: Memory Limit Exceeded,超出内存限制。
- OLE: Output Limit Exceeded,输出超过限制。
- UKE: Unknown Error,出现未知错误。

(3) 测试结果部分正确。

有时提交的代码可以通过输入输出样例,但是只能通过部分测试。例如,下面的代码在读入输入样例“20 30”后输出“50”。

^① <https://help.luogu.com.cn/manual/luogu/problem/judging>。

```
1 #include <stdio.h>
2 int main(){
3     short a,b;
4     scanf("%d %d",&a,&b);
5     printf("%d\n", a+b);
6     return 0;
7 }
```

但是,提交代码后返回 4 个 AC 和 6 个 WA。错误在于,第 3 行的 short 应该改为 int。因为题目说明 a 和 b 最大可能是 10^9 ,而 short 能表示的数据范围小于 10^9 。

(4) C 和 C++ 语言。

虽然所有算法题都可以用纯 C 语言写代码并 AC,但是在竞赛中强烈建议用 C++ 的有关内容,特别是 C++ 的 STL。使用 STL,在很多情况下可以极大简化代码,节省大量编码时间。本题的 C++ 代码如下。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(){
4     int a,b;
5     cin >> a >> b;
6     cout << a+b << endl;
7     return 0;
8 }
```

第 1 行的 bits/stdc++.h 是“万能”头文件,它包含了绝大多数标准 C 和 C++ 的头文件。写代码时,只需要加上万能头文件,不用再写其他头文件。万能头文件仅在 GCC 编译器下可以使用,常见的 C++ IDE 如 Dev-C++、Code::Blocks 自带 GCC 编译器,可以直接引用该头文件。如果使用的是 Microsoft Visual Studio 等,则不支持万能头文件。

第 2 行的 using namespace std 是 C++ 的一个语句,用于简化代码中命名空间的使用。使用该语句后,可以直接使用标准命名空间 std 中的所有成员,而不需要在每个标识符前加上 std:: 前缀。例如本题代码,如果去掉第 2 行,那么第 5 行的 cin >> a 需要改为 std::cin >> a。

本书后续的例题都将用 C++ 写代码,请读者注意总结代码中用到的 STL。



题 1-2 B3685 计算

题目描述: 给定一个三位整数 n , 计算 n 的各位数字之和、 n 的各位数字之和的二次方、 n 的各位数字之和的三次方。例如, 当这个三位数是 123 时:

各位数字之和为 $1+2+3=6$;

各位数字之和的二次方为 $(1+2+3)^2 = (1+2+3) \times (1+2+3) = 36$;

各位数字之和的三次方为 $(1+2+3)^3 = (1+2+3) \times (1+2+3) \times (1+2+3) = 216$ 。

输入格式: 输入只有一行, 一个三位整数 n 。

输出格式: 输出共 3 行, 第一行为各位数字之和, 第二行为各位数字之和的二次方, 第三行为各位数字之和的三次方。

输入样例 817	输出样例 16 256 4096
数据规模与约定：对于全部的测试点，保证 $100 \leq n \leq 999$ 。	

【考点】 输入输出、整除、求余。

【思路 and 证明】 提取 n 的百位、十位、个位数字，然后计算各位数字之和的二次方、三次方。

如何提取一个整数的每个数位上的数字？方法是对 10 或 10 的幂求余、整除。

$n \% 10$ ，即 n 对 10 做除法得到的余数，结果是 n 的个位，如 $817 \% 10 = 7$ 。类似地， $n \% 100$ 的结果是 n 的后两位， $n \% 1000$ 的结果是 n 的后 3 位，等等。

$n / 10$ ，得到 n 去除个位之后的数值，如 $817 / 10 = 81$ 。这里的除法运算是整除（向零取整），结果仍是一个整数，不保留小数位。类似地， $n / 100$ 的结果是去掉 n 的后两位， $n / 1000$ 是去掉 n 的后 3 位，等等。

对于一个三位数 n ，用下面的方法求它的个位、十位、百位。

个位： $n \% 10$ ，如 $817 \% 10 = 7$ 。

十位： $n / 10 \% 10$ ，如 $817 / 10 \% 10 = 1$ 。

百位： $n / 100$ ，如 $817 / 100 = 8$ 。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int main(){
4      int n;
5      cin >> n;                //或 scanf("%d",&n);
6      int x = n / 100;          //百位数字
7      int y = n / 10 % 10;      //十位数字
8      int z = n % 10;           //个位数字
9      int s = x + y + z;        //数字之和
10     cout << s << '\n' << s * s << '\n' << s * s * s << '\n'; //或 printf("%d\n%d\n%d\n",s,s * s,s * s * s);
11
12     return 0;
13 }
```



题 1-3 B3894 开车旅行

题目描述：小明的学校组织了一场春游，春游的目的地是滨江森林公园。由于路途遥远，他们的行程会经过高速公路和地面道路。学校所承包的大巴，在高速公路上以 a 千米/小时的速度匀速行驶，一共行驶了 b 小时；在地面道路上以 c 千米/小时的速度匀速行驶，一共行驶了 d 小时。小明想知道以下两个问题的答案：

(1) 学校到滨江森林公园的路程有多长？

(2) 这辆大巴在旅途中的平均速度是多少？

输入格式：输入一行，4 个实数 a, b, c, d 。

输出格式：输出两行，两个实数，均保留到小数点后一位。第一行一个实数，表示学校到滨江森林公园的路程。第二行一个实数，表示这辆大巴在旅途中的平均速度。

只要你的答案与标准答案的差距不超过 0.1，就被判作正确回答。

输入样例 60 2 30 1	输出样例 150.0 50.0
数据范围：对于所有数据，保证 $0 \leq a, b, c, d \leq 1000$ ，且 $b + d \neq 0$ 。	

【考点】 小数处理、四舍五入。

【思路 and 证明】 用本题熟悉小数的处理。

高速公路上的路程为 $a \times b$ 千米，地面道路上的路程为 $c \times d$ 千米，总路程为 $a \times b + c \times d$ 千米，为第一问答案。总行驶时间为 $b + d$ 小时，平均速度为 $(a \times b + c \times d) / (b + d)$ 千米/小时。

本题的 a, b, c, d 是浮点数，浮点数有 float、double 类型，由于本题的数值不大，且小数点后保留位数只有一位，数值的有效位数少，用 float 类型可以满足，不需要用 double。不过，由于 double 比 float 精度更高，在浮点数运算中误差更小，所以本书的浮点数统一用 double，不用 float。

为什么浮点数计算会有精度问题？原因是计算机内的所有数字都是用二进制表示的，题目给的是十进制，计算机会先转换为二进制再计算。一个十进制有限小数转换为二进制后往往是无限小数，而计算机的浮点数有长度限制，超出的小数被丢弃，就产生了精度损失。例如，十进制数 0.2，转换成十位二进制小数是 0.0011001100，再转换回十进制是 0.19921875，不是 0.2。而整数没有这个问题，一个十进制整数可以准确地转换成任何进制的整数。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int main(){
4      double a,b,c,d;
5      scanf("%lf%lf%lf%lf",&a,&b,&c,&d); //注意是%lf. 或 cin>>a>>b>>c>>d;
6      double u = a * b;                      //计算高速公路路程
7      double v = c * d;                      //计算地面道路路程
8      double sum = u + v;
9      double avg = sum / (b + d);
10     printf("%.1lf\n",sum); //或 cout<<fixed<<setprecision(1)<<sum<<endl;
11     printf("%.1lf\n",avg); //注意是%lf
12     return 0;
13 }
```

第 5 行的输入，用 cin 比 scanf() 简洁。

第 10 行的小数输出，用 printf() 更简洁，用 cout 比较烦琐。

第 10 行使用 printf() 输出小数是四舍五入^①。例如， $sum = 66.78$ ，保留一位小数输出是 66.8。本题允许答案有 0.1 的误差，因此直接使用 printf() 输出即可。

变量用适合理解的名字命名，如用 sum 表示总路程，用 avg 表示平均速度。

注意，double 的格式用 %lf。

^① 在统计中，常使用“四舍六入五成双”原则。因为“四舍五入”中 5 一律进 1，会导致大量数据的统计结果偏大。用“四舍六入五成双”可以解决，方法是：若 5 后面还有非零数，则进 1，如 $1.2351 \rightarrow 1.24$ ；若 5 后面没有数字，看前一位，若前一位是偶数，舍 5，如 $1.245 \rightarrow 1.24$ ，若前一位是奇数，进 1，如 $1.235 \rightarrow 1.24$ 。这就是“五成双”，让 5 一半进、一半舍（前一位奇进偶舍），当批量计算时误差抵消，结果更准确。

如果处理小数时,要求使用去尾法,如 $\text{sum} = 66.78$,保留一位小数输出 66.7,可以这样做:

```
printf("%.1f\n", (int)(sum * 10)/10.0);
```

在这里, $\text{sum} * 10 = 667.8$,被强制类型转换为整数,保留了 667,接着/10.0(注意:如果是/10 会被认为是做整数除法而不保留小数),结果为 66.7。

同理,如果保留两位小数,则为 $(\text{int})(\text{sum} * 100)/100.0$ 。

保留 3 位或更多位小数,方法相同。



题 1-4 P5716 月份天数

题目描述: 输入年份和月份,输出这一年的这一月有多少天。需要考虑闰年。

输入格式: 输入两个正整数,分别表示年份 y 和月份 m ,以空格隔开。

输出格式: 输出一个正整数,表示这个月有多少天。

输入样例

1926 8

输出样例

31

数据范围: 对于所有数据,保证 $1583 \leq y \leq 2020, 1 \leq m \leq 12$ 。

【考点】 平/闰年判断。

【思路 and 证明】 闰年的判断是常见的考点。

闰年的判断: 能整除 4 且不能整除 100 的为闰年; 能整除 400 的为闰年。设年份变量为 y , 则第一个条件表达式为 $(y \% 4 == 0 \ \&\& \ y \% 100 != 0)$, 第二个条件表达式为 $(y \% 400 == 0)$ 。由于两个条件只需满足一个, 因此使用逻辑或运算 $\parallel$ 将它们连接, 一起写为: $(y \% 4 == 0 \ \&\& \ y \% 100 != 0) \parallel (y \% 400 == 0)$ 。

一年的 12 个月中, 只有 2 月的天数和闰年有关, 其他月份的天数是固定的。编码时特殊处理 2 月即可。

【代码展示】

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  int main(){
4      int y,m;
5      cin >> y >> m;
6      int feb = 28; //2 月
7      if((y%4 == 0 && y%100!= 0) || (y%400 == 0)) feb = 29; //闰年 2 月有 29 天
8      if(m==2) cout << feb;
9      else if(m== 4 || m== 6 || m== 9 || m== 11) cout << 30;
10     else cout << 31;
11     return 0;
12 }
```

第 6 行用 feb 表示 2 月的天数,平年的 2 月有 28 天,闰年的 2 月有 29 天。

需要注意运算符的优先级。为避免纠结于运算符优先级,建议使用括号处理逻辑表达式中的运算符。例如,第 7 行条件表达式这样写也是对的:

```
if(y%4 == 0 && y%100!= 0 || y%400 == 0)
```

用括号把前后两种判断条件分开不易出错,改为:

```
if((y%4 == 0 && y%100 != 0) || (y%400 == 0))
```



题 1-5 P5718 找最小值

题目描述: 给出 n 和 n 个整数 a_i , 求这 n 个整数中的最小值。

输入格式: 第一行输入一个正整数 n , 表示数字个数; 第二行输入 n 个非负整数 $a_1, a_2, \dots, a_n$, 以空格隔开。

输出格式: 输出一个非负整数, 表示这 n 个非负整数中的最小值。

输入样例

5

5 7 4 2 6

输出样例

2

数据范围: $n \leq 100, 0 \leq a_i \leq 1000$ 。

【考点】 打擂台法。

【思路 and 证明】 求最小值有多种方法, 这里用最简单的“打擂台法”。首先准备一个存储答案的变量 ans , 作为“擂主”, 初始化为读入的数的最大可能值(在本题中为 1000)。然后依次读入每个整数 a , 如果 $a < ans$, 则让 ans 变为 a ; 如果 $a \geq ans$, 则 ans 不变。当所有数都读完后, ans 就是最小值。

下面用样例说明操作步骤。一开始令答案变量 ans 为 1000。

读入第一个数 5, 5 小于变量 ans 的数值 1000, 则变量 ans 的数值变为 5。

读入第二个数 7, 7 大于变量 ans 的数值 5, ans 不变。

读入第三个数 4, 4 小于变量 ans 的数值 5, 则变量 ans 的数值变为 4。

读入第四个数 2, 2 小于变量 ans 的数值 4, 则变量 ans 的数值变为 2。

读入第五个数 6, 6 大于变量 ans 的数值 2, ans 不变。

因此这 n 个非负整数的最小值是变量 ans 的数值 2。

【代码展示】 题目需要输入 n 个整数, 有以下两种方法。

一种方法是先全部读入 n 个数, 然后再找其中的最小值。需要定义一个大小为 n 的数组存储这 n 个数。

另一种方法是一边读一边比较, 这样就不需要定义数组了, 下面的代码使用了这种方法。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(){
4     int n;
5     cin >> n;
6     int ans = 1000;
7     while(n--){
8         int a;
9         cin >> a;
10        if (ans > a) ans = a;
11    }
12    cout << ans;
13    return 0;
14 }
```

//ans 记录最小值, 初始化为最大值
//依次处理 n 个数
//读一个数, 比较一次
//比较 ans 和 a, 打擂台

第 10 行的比较也可以用 `algorithm` 库中的 `min()` 函数,代码改为:

```
ans = min(ans, a);
```



题 1-6 P1085 不高兴的津津

题目描述: 津津上初中了。妈妈认为津津应该更加用功学习,所以津津除了上学之外,还要参加妈妈为她报名的各科复习班。另外,每周妈妈还会送她去学习朗诵、舞蹈和钢琴。但是津津如果一天上课超过 8 个小时就会不高兴,而且上得越久就会越不高兴。假设津津不会因为其他事不高兴,并且她的不高兴不会持续到第二天。请你帮忙检查一下津津下周的日程安排,看看下周她会不会不高兴;如果会的话,哪天最不高兴。

输入格式: 输入包括 7 行数据,分别表示周一到周日的日程安排。每行输入两个小于 10 的非负整数,用空格隔开,分别表示津津在学校上课的时间和妈妈安排她上课的时间。

输出格式: 一个数字。如果不会不高兴则输出 0,如果会不高兴则输出最不高兴的是周几(用 1,2,3,4,5,6,7 分别表示周一,周二,周三,周四,周五,周六,周日)。如果有两天或两天以上不高兴的程度相当,则输出时间最靠前的一天。

输入样例

```
5 3
6 2
7 2
5 3
5 4
0 4
0 6
```

输出样例

```
3
```

【考点】 打擂台法。

【思路 and 证明】 根据题意,循环读入 7 行正整数 a 和 b ,分别表示学校学习小时数和课外学习小时数。由于上课超过 8 个小时才会不高兴,且只需知道最靠前的最不高兴的一天即可,因此使用打擂台法,判断 $a + b$ 是否大于目前的最大学习总时间(变量 `max_time`),若大于则记录这是星期几(变量 `ans`)并更新最大值。

【代码展示】

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main() {
4     int max_time = 0;           //最大学习总时间
5     int ans = 0;                //答案
6     for(int i = 1; i <= 7; i++) { //读 7 组数据
7         int a, b; cin >> a >> b; //读星期 i 的学校学习小时数与课外学习小时数
8         if(a + b > max_time && a + b > 8) {
9             ans = i;           //更新答案
10            max_time = a + b;   //更新最大值
11        }
12    }
13    cout << ans;
14    return 0;
15 }
```



题 1-7 P1035 级数求和

题目描述：已知 $S_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$ 。显然对于任意一个整数 k ，当 n 足够大时， $S_n > k$ 。现给出一个整数 k ，要求计算出一个最小的 n ，使得 $S_n > k$ 。

输入格式：一个正整数 k 。

输出格式：一个正整数 n 。

输入样例

1

输出样例

2

数据范围：对于 100% 的数据， $1 \leq k \leq 15$ 。

【考点】 循环。

【思路 and 证明】 本题可以使用 while 循环编程，while 循环的括号内是循环进行的判断条件。从 $n=1$ 开始循环，逐步累计级数的项，直到和 sum 首次大于 k ，停止循环，因此循环判断条件为 $\text{sum} \leq k$ 。

由于是小数运算，因此 sum 应当使用 double 类型存储。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int main() {
4      int k;
5      cin >> k;
6      double sum = 1;
7      int n = 1;
8      while(sum <= k){                //判断终止条件
9          n++;
10         sum += 1.0/n;                //如果 1.0 写成 1, 因为 n 是 int 型, 导致 1/n 是整数
11     }
12     cout << n;
13     return 0;
14 }
```

第 10 行如果改成 $\text{sum} += 1/n$ ，由于 n 是 int 型，编译器会将 $1/n$ 理解为整数除法，不保留小数位，造成计算错误。因此需要将 1 写作 1.0 进行浮点数除法。

或者把第 7 行的 n 定义为 double 型，那么第 10 行可以写为 $\text{sum} += 1/n$ 。



题 1-8 B2068 统计满足条件的 4 位数

题目描述：给定若干四位数，求出其中满足以下条件的数的个数：个位数字减去千位数字，再减去百位数字，再减去十位数字，结果大于零。

输入格式：输入为两行，第一行为四位数的个数 n ，第二行为 n 个的四位数。

输出格式：输出为一行，包含一个整数，表示满足条件的四位数的个数。

输入样例

5

1234 1349 6119 2123 5017

输出样例

3

样例解释:

对于 1234, $4-3-2-1=-2<0$;

对于 1349, $9-4-3-1=1>0$;

对于 6119, $9-1-1-6=1>0$;

对于 2123, $3-2-1-2=-2<0$;

对于 5017, $7-1-0-5=1>0$;

因此满足条件的四位数有 1349、6119、5017, 共 3 个。

数据范围: $n \leq 100$ 。

【考点】 循环、数位拆分。

【思路 and 证明】 本题也是数位拆分, 方法和例题“B3685 计算”的数位拆分相同。

【代码展示】

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(){
4     int n;
5     cin >> n;
6     int cnt = 0;           //局部变量需要初始化为 0
7     while(n--){           //读 n 个数
8         int x;
9         cin >> x;
10        int q = x/1000;     //分离千位
11        int b = x/100%10;   //分离百位
12        int s = x/10%10;    //分离十位
13        int g = x%10;       //分离个位
14        if(g-q-b-s > 0) cnt++;
15    }
16    cout << cnt;
17    return 0;
18 }
```



题 1-9 P1980 计数问题

题目描述: 试计算 $1 \sim n$ 的所有整数中, 数字 x 共出现了多少次? 例如, 在 $1 \sim 11$ 中, 即在 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 中, 数字 1 出现了 4 次。

输入格式: 两个整数 n, x , 用空格隔开。

输出格式: 一个整数, 表示 x 出现的次数。

输入样例

11 1

输出样例

4

数据范围: 对于 100% 数据, $1 \leq n \leq 10^6, 0 \leq x \leq 9$ 。

【考点】 循环嵌套。

【思路 and 证明】 与例题“B3685 计算”不同的是, 这道题要拆分的并不是三位数, 而是长度不定的数字, 因此需要使用循环逐位进行数位拆分。

代码采用循环嵌套结构, 在外层使用循环变量 i , 用 i 遍历 $1, 2, \dots, n$; 在内层循环, 遍历 i 的各个数位, 统计数字 x 出现的个数。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int main(){
4      int n,x;
5      cin >> n >> x;
6      int ans = 0;                                //ans 是答案,注意要初始化为 0
7      for(int i = 1; i <= n; i++){                //外层循环
8          int d = i;
9          while(d > 0){                            //内层循环。数位拆分: 逐个检查 d 的每一位
10             if(d%10 == x) ans++;                //用求余计算个位
11             d = d/10;
12         }
13     }
14     cout << ans;
15     return 0;
16 }
```

在本题中,第 7 行的 for 循环循环 n 次,第 9 行的 while 循环最多循环 7 次(i 为 10^6 时执行 7 次),总循环计算量不超过 $7n$ 次,代入题目数据范围 $n \leq 10^6$,总计算量不超过 700 万。现代计算机一秒钟可以进行一亿次运算,因此可以在远小于 1 秒的时间内完成本题。

通过本题学习“数位拆分”的方法,具体实现见第 10、11 行。数位拆分用到求余计算,在每一位上用求余来取这一位的数字。注意学习这个技巧。

数位拆分的计算过程举例如下。假设第 9 行处理的 d 是 531,拆分 d 的每一位,步骤如下。

(1) 在 while 循环的第一次循环中, $d \% 10 = 1$,用求余取 d 的最后一位,此时得到 531 的个位数 1。接着 $d = d/10$, d 变为 53,去掉了原来的个位数 1。为什么 $d/10$ 不等于 53.1? 因为是整除,把小数丢掉了。

(2) 进入第二次循环,此时 $d \% 10 = 3$,相当于取到了原来 d 的十位数字。

(3) 以此类推……



题 1-10 B3841 自幂数判断

题目描述: 自幂数是指一个 N 位数,满足各位数字 N 次方之和是本身。例如,153 是三位数,其每位数的 3 次方之和为 $1^3 + 5^3 + 3^3 = 153$,因此 153 是自幂数;1634 是四位数,其每位数的 4 次方之和为 $1^4 + 6^4 + 3^4 + 4^4 = 1634$,因此 1634 是自幂数。现在,输入若干正整数,请判断它们是否是自幂数。

输入格式: 第一行输入一个正整数 M ,表示有 M 个待判断的正整数。约定 $1 \leq M \leq 100$ 。

从第二行开始的 M 行,每行输入一个待判断的正整数。约定这些正整数均小于 10^8 。

输出格式: 输出 M 行,如果对应的待判断正整数为自幂数,则输出英文大写字母 T,否则输出英文大写字母 F。

提示: 不需要等到所有输入结束再依次输出,可以输入一个数就判断并输出,再输入下一个数。

输入样例	输出样例
3	F
152	F
111	T
153	

【考点】 数位拆分, pow() 函数。

【思路 and 证明】 自幂数和位数有关, 所以输入一个整数 a 后, 需要先求 a 的位数。可以通过除以 10 并取整循环直到商变为 0 来计算, 见代码第 9~12 行。

然后将这个数分解为各位数字。可以用求余运算得到每位的值, 然后用整除操作去除当前位。对于每位数字, 计算它们的 N 次方 (N 即为该数的位数), 并累加到一个变量中, 见代码第 15~19 行。

第 17 行用 pow() 函数计算幂, 它的原型是 double pow(double x , double y), 返回 x^y 。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int main(){
4      int m;  cin >> m;
5      while(m-- ) {
6          int a;  cin >> a;                //输入一个数
7          int p = 0;                        //记录这个数有多少位
8          int n = a;
9          while (n) {                      //计算这个数的位数
10             n /= 10;
11             ++p;
12         }
13         int sum = 0;                      //记录和
14         n = a;
15         while(n) {
16             int b = n % 10;               //用求余得到最低位的值
17             sum += pow(b, p);              //p 为位数, 逐位运算求和
18             n /= 10;                      //用整除去掉最低位
19         }
20         if(sum == a) cout << 'T' << "\n";
21         else          cout << "F" << "\n";
22     }
23     return 0;
24 }
```

小心使用 pow() 函数, 它的计算结果可能有精度误差, 见后面例题“B3620 X 进制转 10 进制”“P10510 进制”对 pow() 函数精度的说明。



题 1-11 B3988 天然气计费

题目描述: 天然气采用阶梯计费。

第一档收费标准: 年用气量在 0~310(含)立方米, 价格为 3 元/立方米;

第二档收费标准: 年用气量在 310~520(含)立方米, 价格为 3.3 元/立方米;

第三档收费标准: 年用气量在 520 立方米以上, 价格为 4.2 元/立方米。

例如,小 Z 家今年用气量为 600 立方米,费用为 $310 \times 3 + (520 - 310) \times 3.3 + (600 - 520) \times 4.2 = 930 + 693 + 336 = 1959$ 元。

现在给出小 Z 家 1 月至 12 月的用气量,请输出小 Z 家每个月的天然气费用。

输入格式:共 12 行,其中第 i 行包含一个整数,表示 i 月的用气量,单位为立方米。

输出格式:共 12 行,其中第 i 行包含一个小数,表示这个月的天然气费用,结果保留至小数点后 1 位。

输入样例	输出样例
100	300.0
100	300.0
100	300.0
100	327.0
100	330.0
100	402.0
100	420.0
100	420.0
100	420.0
100	420.0
100	420.0
100	420.0

数据范围:对于 100% 的数据,满足一年的总用气量不超过 10^9 。

【考点】 循环。

【思路 and 证明】 每年使用的天然气的价格只在前 520 立方米中有变化。当超过 520 立方米天然气后,后续的天然气都可以按一个价格来计算。

在计算费用时,按照这样的步骤:

- (1) 如果当前使用的天然气总数量 $\text{cur} < 520$ 立方米,那么就逐立方米计算;
- (2) 否则,直接按照最高价来计算,不需要每立方米计算一次。

上述步骤使用 while 循环来完成:

```
while (cur < 520 && 当月还有没计算的天然气用量 > 0) { ... }
```

【代码展示】

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(){
4     int n = 12;
5     int cur = 0;                                //cur 记录当前使用的天然气总量
6     for(int i = 1; i <= n; i++) {                //读每月的天然气
7         int a;
8         cin >> a;
9         double ans = 0.0;                        //注意要定义为 double 型
10        while(cur < 520 && a > 0) {
11            if(cur < 310) ans += 3;                //少于 310 立方米的按 3 元单价算
12            else ans += 3.3;
```

```
13         a--; //当月还没计算的天然气用量
14         cur++;
15     }
16     ans += a * 4.2; //超过 520 立方米的按 4.2 元单价算
17     printf("%.11f\n", ans);
18     //cout << fixed << setprecision(1) << ans << '\n'; //cout 比较烦琐
19 }
20 return 0;
21 }
```

第 17 行用 printf() 输出浮点数, 比第 18 行用 cout 简洁。



题 1-12 B3865 小杨的 X 字矩阵

题目描述: 小杨想要构造一个 $N \times N$ 的 X 字矩阵 (N 为奇数), 这个矩阵的两条对角线都是半角加号+, 其余都是半角减号-。例如, 一个 5×5 的 X 字矩阵如下。

```
+---+
-+-+-
--+-
-+-+-
+---+
```

请你帮小杨根据给定的数字打印出对应的 X 字矩阵。

输入格式: 一个整数 N , $5 \leq N \leq 49$, 保证为奇数。

输出格式: 输出对应的 X 字矩阵。

请严格按格式要求输出, 不要擅自添加空格、标点、空行等任何符号。应该恰好输出 N 行, 每行除了换行符外恰好包含 N 个字符, 这些字符要么是+, 要么是一。

输入样例

5

输出样例

```
+---+
-+-+-
--+-
-+-+-
+---+
```

【考点】 循环。

【思路 and 证明】 输出矩阵这样的平面图形, 通常使用二重循环完成。外层循环表示目前正在输出第几行的内容, 内层循环表示目前正在输出第几列的内容。

设矩阵左上角坐标为 $(1, 1)$, 右下角坐标为 (n, n) 。对于矩阵内的坐标点 (i, j) , 有以下结论。

(1) 若 $i = j$, 则点位于主对角线 (从左上到右下的对角线) 上, 输出+。例如, 第 1 行第 1 列, 第 2 行第 2 列, 都应当输出+。

(2) 若 $i + j = n + 1$, 则点位于副对角线 (从左下到右上的对角线) 上, 输出+。例如, 当 $n = 5$ 时, 第 1 行第 5 列, 第 2 行第 4 列, 都应当输出+。

(3) 其他位置的点, 输出一。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int main(){
4      int n;
5      cin>>n;
6      for(int i=1; i<=n; i++) {
7          for(int j=1; j<=n; j++) {
8              if(i==j || i+j==n+1) cout<<'+';
9              else cout<<'-' ;
10         }
11         cout<<"\n";           //换行
12     }
13     return 0;
14 }

```



题 1-13 B3688 旋转排列

题目描述：我们称一个数列 p 是一个长度为 n 的排列，当且仅当 p 满足如下条件：

- (1) p 的长度为 n ；
- (2) $1, 2, \dots, n$ 这 n 个数在 p 中均恰好出现一次。

对于一个排列 p ，定义一次 shift 操作：将 p 里的每个数字都依次向后移动一位，并把 p 的最后一个数字移动到开头去。

例如，若排列 p 初始时为 $[1, 4, 2, 3]$ ，则一次 shift 操作以后将变为 $[3, 1, 4, 2]$ 。

现在，给定一个长度为 n 的排列 p ，请你按照如下规定循环操作。

- (1) 对当前的排列 p 做一次 shift 操作；
- (2) 输出本次 shift 操作以后的排列 p ；
- (3) 判断排列 p 的最后一个数字是否是 n ，如果是，则结束循环操作；否则回到步骤(1)继续操作。

提示：请严格按照题目给出的顺序进行循环操作。

输入格式：第一行输入一个整数，表示排列 p 的长度 n ；第二行输入 n 个整数，表示排列 p ，第 i 个整数表示 p_i 。

输出格式：对于每次操作的第二条“输出”操作，请输出一行 n 个整数，按顺序表示当前排列的每个数，一行中相邻两个数之间用一个空格隔开。

输入样例

4
1 4 2 3

输出样例

3 1 4 2
2 3 1 4

数据范围：对全部的测试点，保证 $1 \leq p_i \leq n \leq 2000$ ， p 是长度为 n 的排列。

【考点】 循环、数组。

【思路 and 证明】 根据题目中给出的循环操作规定，先进行循环操作处理，再判断是否要结束流程，因此使用 do-while 语句较为合适。

代码分为 3 部分。

首先，把所有数往后移动一位，然后把最后的数移动到第一位；使用令 i 从 n 到 1 的倒序循环完成，每次令 $p[i+1]=p[i]$ ，这样 $p[n+1]$ 存储的就是原本的最后一个数字 $p[n]$ ，

令 $p[1]=p[n+1]$ 即可完成 shift 操作。

以 $n=4$, 序列为 $\{4\ 1\ 2\ 3\}$ 的情况为例, 由于我们只输出 $p[1]\sim p[4]$ 的内容, 因此无论 $p[5]$ 存放了什么样的数字都无须清空, 如表 1-1 所示。

表 1-1 序列内容

下 标	1	2	3	4	5
原 始 序 列	4	1	2	3	0
往后移动一次的结果	4	4	1	2	3
$p[1]=p[n+1]$ 后的序列	3	4	1	2	3

接着, 使用一重循环输出这个序列。

最后, 判断 $p[n]$ 是否是 n , 若是则退出循环。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int p[2005];           //全局定义一个大数组,覆盖所有测试的需要
4  int main(){
5      int n;
6      cin >> n;
7      for(int i = 1; i <= n; i++)  cin >> p[i];
8      do {
9          for(int i = n; i >= 1; i--)           //把每个数往后移动一位
10             p[i+1] = p[i];                   //把最后的 p[n]移动到 p[n+1]
11         p[1] = p[n+1];
12         for(int i = 1; i < n; i++)             //打印这个排列的前 n-1 个
13             cout << p[i] << " ";
14         cout << p[n] << "\n";                 //打印最后一个,然后换行
15     } while (p[n] != n);
16     return 0;
17 }
```

定义 $p[]$ 时, 把数组开大一些, 保证不会溢出。题目中 $n \leq 2000$, 那么就定义一个更大的数组 $p[2005]$ 。



题 1-14 B4038 平衡序列

题目描述: 小杨有一个包含 n 个正整数的序列 a 。他认为一个序列是平衡的, 当且仅当存在一个正整数 i ($1 \leq i < n$) 使得序列中第 1 到第 i 个数字的总和等于第 $i+1$ 到第 n 个数字的总和。小杨想请你判断序列 a 是否是平衡的。

输入格式: 本题单个测试点内包含多组测试数据。第一行输入一个正整数 t , 表示测试用例组数。接下来是 t 组测试用例。每组测试用例输入两行: 第一行包含一个正整数 n , 表示序列长度; 第二行包含 n 个正整数, 代表序列 a 。

输出格式: 对每组测试用例输出一行(一个字符串)。如果 a 是平衡的, 输出 Yes, 否则输出 No。

输入样例

3

3

1 2 3

输出样例

Yes

Yes

No

```

4
2 3 1 4
5
1 2 3 4 5

```

数据范围：对全部的测试数据，保证 $1 \leq t \leq 100; 1 \leq n, a_i \leq 10000$ 。

【考点】 循环，数组。

【思路 and 证明】 题目的意思是，把序列分成左右两部分，分别求和，问是否相等。

方法 1：直接模拟题目的要求，循环枚举 i ，先累加 $a_1 \sim a_i$ ，再累加 $a_{i+1} \sim a_n$ ，判断两部分是否相等。编码需要用两重循环，第一重循环枚举 i ，在循环内部使用第二重循环，统计序列 $a_1 \sim a_i$ 之和，以及 $a_{i+1} \sim a_n$ 之和。这个方法的答案正确，但是无法通过本题，因为计算量太大。两重循环共循环 n^2 次，共 t 组测试，总计算量为 $t \times n^2 = 100 \times 10000^2$ ，计算时间远远大于 1 秒的判题时间。

如何减少计算量？如果能把 $t \times n^2$ 减少为 $t \times n$ ，就能通过判题了。

方法 2： $a_1 \sim a_i$ 的和与 $a_{i+1} \sim a_n$ 的和相等，相当于把原来序列的总和分成两个相等的一半。所以只要找到一个 i ，使得 $a_1 \sim a_i$ 之和等于总和的一半就行了。先对整个序列求总和得到 s ，然后做一次循环，找到一个 i ，计算 $a_1 \sim a_i$ 之和 $s/2$ ，就是答案。这个方法只需要一重循环。共 t 组测试，总计算量为 $t \times n$ ，能通过判题测试。

需要注意，本题有多组测试数据，因此在处理每组测试数据之前，记录总和、标记答案的相关变量都需要重新初始化，以免造成意料之外的情况。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int a[10005];
4  int main(){
5      int t;
6      cin >> t;
7      while(t--){
8          int n; cin >> n;
9          int s = 0; //序列总和
10         for(int i = 1; i <= n; i++){
11             cin >> a[i];
12             s += a[i];
13         }
14         int u = 0; //左半部分的和
15         int flag = 0; //若 flag == 1, 表示平衡
16         for(int i = 1; i <= n; i++){ //找到使左半部分和等于右半部分和的 i
17             u += a[i];
18             if(u * 2 == s) flag = 1; //如果写成 u == s/2 是错的
19         }
20         if(flag == 1) cout << "Yes\n";
21         else cout << "No\n";
22     }
23     return 0;
24 }

```

注意数据类型，序列和 s 应该是 `int` 还是 `long long` 类型？当 $n = 10000, a = 10000$ 时， $s = 10^8$ ，用 `int` 类型即可。当然，如果不想分析，直接用 `long long` 类型也行。

第 18 行判断 u 是否等于 $s/2$, 应该写成 $u * 2 == s$, 而不是 $u == s/2$, 避免由于整数除法取整造成错误, 如 $s=21$, $s/2$ 的结果是 10。



题 1-15 B2104 矩阵加法

题目描述: 输入两个 n 行 m 列的矩阵 A 和 B , 输出它们的和 $A+B$, 矩阵加法的规则是对两个矩阵中对应位置的值进行加和, 具体参照样例。

输入格式: 第一行输入两个整数 n 和 m , 表示矩阵的行数和列数。

接下来 n 行, 每行 m 个整数, 表示矩阵 A 的元素。

接下来 n 行, 每行 m 个整数, 表示矩阵 B 的元素。

相邻两个整数之间用单个空格隔开, 每个元素均为 $1 \sim 1000$ 。

输出格式: n 行, 每行 m 个整数, 表示矩阵加法的结果。相邻两个整数之间用单个空格隔开。

输入样例

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

输出样例

```
2 4 6
5 7 9
8 10 12
```

数据范围: $1 \leq n \leq 100, 1 \leq m \leq 100$ 。

【考点】 二维数组, 矩阵加法。

【思路 and 证明】 两个矩阵 A 、 B 能相加的条件是它们有相同的行数和相同的列数, 把对应位置的元素相加, 结果存在新的矩阵 C 中。

定义二维数组存储矩阵 A 、 B 、 C 。首先, 使用两个二重循环读入存储矩阵 A 和矩阵 B , 再令矩阵 C 为矩阵 A 和矩阵 B 的对应位置相加的结果, 然后输出矩阵 C 。

【代码展示】 下面的代码用了 3 个二维数组。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 110;
4 int a[N][N], b[N][N], c[N][N];
5 int main(){
6     int n, m;    cin >> n >> m;
7     for(int i = 1; i <= n; i++)                //输入矩阵 A
8         for(int j = 1; j <= m; j++)
9             cin >> a[i][j];
10    for(int i = 1; i <= n; i++)                //输入矩阵 B
11        for(int j = 1; j <= m; j++){
12            cin >> b[i][j];
13            c[i][j] = a[i][j] + b[i][j];        //计算矩阵 C
14        }
15    for(int i = 1; i <= n; i++) {                //输出矩阵 C
16        for(int j = 1; j <= m; j++)
```

```

17         cout << c[i][j] << ' ';
18         cout << "\n";           //换行
19     }
20     return 0;
21 }

```

矩阵的存储非常耗费空间,因为矩阵是 $n \times m$ 的二维数组,上面的代码用了 3 个二维数组分别定义 **A**、**B**、**C** 矩阵,需要“ $3 \times n \times m \times$ 单个该数据类型变量”所需空间的内存。如果题目的 n 、 m 很大,3 个矩阵可能超出空间限制。例如, $n = m = 10000$,每个矩阵元素是 4B 的 int 型,那么需要的存储空间为 $4 \times 3 \times n \times m = 1.2\text{GB}$ 。

可以使用技巧减少空间。下面的代码只定义一个矩阵 $a[][]$,内存空间只是上面代码的 1/3。 $a[][]$ 用于存储矩阵 **A**,先读取矩阵 **A**,存在 $a[][]$ 中;矩阵 **B** 不存储,而是每读一个矩阵 **B** 的元素,就立刻和矩阵 **A** 的元素相加,结果就是矩阵 **C** 的元素,并立刻输出,这样也不需要存储矩阵 **C**。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int a[110][110];           //只定义一个矩阵
4  int main(){
5      int n,m;    cin>>n>>m;
6      for(int i=1; i<=n; i++)           //输入矩阵 A
7          for(int j=1; j<=m; j++)
8              cin>>a[i][j];
9      for(int i=1; i<=n; i++) {           //输入矩阵 B
10         for(int j=1; j<=m; j++) {
11             int b;    cin>>b;           //读矩阵 B 元素
12             cout << a[i][j] + b << ' '; //计算和输出矩阵 C 元素
13         }
14         cout << "\n";           //换行
15     }
16     return 0;
17 }

```



题 1-16 B2106 矩阵转置

题目描述: 输入一个 n 行 m 列的矩阵 **A**, 输出它的转置 A^T 。

输入格式: 第一行包含两个整数 n 和 m , 表示矩阵 **A** 的行数和列数。接下来 n 行, 每行 m 个整数, 表示矩阵 **A** 的元素。相邻两个整数之间用单个空格隔开, 每个元素取值范围为 $1 \sim 1000$ 。

输出格式: m 行, 每行 n 个整数, 为矩阵 **A** 的转置。相邻两个整数之间用单个空格隔开。

输入样例

```

3 3
1 2 3
4 5 6
7 8 9

```

输出样例

```

1 4 7
2 5 8
3 6 9

```

数据范围: $1 \leq n \leq 100, 1 \leq m \leq 100$ 。

【考点】 二维数组。

【思路 and 证明】 矩阵转置是将矩阵的行和列互换。对于一个 $m \times n$ 的矩阵 A ，其转置矩阵 A^T 是一个 $n \times m$ 的矩阵，其中 A^T 的第 i 行第 j 列元素等于 A 的第 j 行第 i 列元素。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int a[105][105];
4  int main(){
5      int n,m;
6      cin>>n>>m;
7      for(int i=1;i<=n;i++)                //输入矩阵,n行m列
8          for(int j=1;j<=m;j++)
9              cin>>a[i][j];
10     for(int j=1;j<=m;j++){                //颠倒 i,j 后输出
11         for(int i=1;i<=n;i++)
12             cout<<a[i][j]<<" ";          //注意: 这里是 a[i][j],不是 a[j][i]
13         cout<<"\n";                      //换行
14     }
15     return 0;
16 }
```



题 1-17 P2615 神奇的幻方

题目描述：幻方是一种很神奇的 $N \times N$ 矩阵，它由数字 $1, 2, \dots, N \times N$ 构成，且每行、每列及两条对角线上的数字之和都相同。

当 N 为奇数时，通过以下方法构建一个幻方：首先将 1 写在第一行的中间，之后，按如下方式从小到大依次填写每个数 $K (K=2, 3, \dots, N \times N)$ 。

(1) 若 $(K-1)$ 在第一行但不在最后一列，则将 K 填在最后一行， $(K-1)$ 所在列的右一列；

(2) 若 $(K-1)$ 在最后一列但不在第一行，则将 K 填在第一列， $(K-1)$ 所在行的上一行；

(3) 若 $(K-1)$ 在第一行最后一列，则将 K 填在 $(K-1)$ 的正下方；

(4) 若 $(K-1)$ 既不在第一行，也不在最后一列，如果 $(K-1)$ 的右上方还未填数，则将 K 填在 $(K-1)$ 的右上方，否则将 K 填在 $(K-1)$ 的正下方。

现给定 N ，请按上述方法构造 $N \times N$ 的幻方。

输入格式：一个正整数 N ，即幻方的大小。

输出格式：共 N 行，每行 N 个整数，即输出按上述方法构造出的 $N \times N$ 的幻方，相邻两个整数之间用单空格隔开。

输入样例

3

输出样例

8 1 6

3 5 7

4 9 2

数据范围： $1 \leq N \leq 39$ 且 N 为奇数。

【考点】 楼梯法填矩阵。

【思路 and 证明】 题目描述的方法是经典的“楼梯法”，用于构造奇数阶幻方。填幻方的

过程就像“走楼梯”，一直向右上角走，如果走不通或遇到边界，则按规则重新走楼梯。读者可以尝试证明为什么这个方法能填出一个奇数阶的幻方。对于偶数阶幻方、通用阶幻方，也有构造方法，如对角线法、斯特雷奇法等。

编程时，根据题目中给出的构建幻方的规则，将数字逐一填入。假设当前填写第 x 行第 y 列。由于 1 填写在第一行的中间，因此初始值 $x=1, y=N/2+1$ 。之后用循环从 2 到 N^2 逐一填数。

判断在第一行但不在最后一列： $x==1 \ \&\& \ y \neq N$ ，填写位置为 $x=N, y=y+1$ 。

判断在最后一列但不在第一行： $y==N \ \&\& \ x \neq 1$ ，填写位置为 $x=x-1, y=1$ 。

判断在第一行最后一列： $x==1 \ \&\& \ y==N$ ，填写位置为 $x=x+1$ 处。

不在第一列但在最后一行： $x \neq 1 \ \&\& \ y \neq N$ ，其右上角位置为 $x=x-1, y=y+1$ ，下方位置为 $x=x+1$ 。

【代码展示】 根据规则填写幻方后输出。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int a[45][45];
4  int main(){
5      int N; cin>>N;
6      a[1][N/2+1]=1;           //数字1,位于第一行中间
7      int x=1,y=N/2+1;        //当前位于x行,y列
8      for(int i=2; i<=N*N; i++) {           //逐个填数字
9          if(x==1 && y!=N) { a[N][y+1]=i,x=N,y++; }
10         else if(y==N && x!=1) { a[x-1][1]=i,x--,y=1; }
11         else if(x==1 && y==N) { a[x+1][y]=i,x++; }
12         else if(x!=1 && y!=N){
13             if(a[x-1][y+1]==0) { a[x-1][y+1]=i,x--,y++; }
14             else { a[x+1][y]=i,x++; }
15         }
16     }
17     for(int i=1; i<=N; i++) {
18         for(int j=1; j<=N; j++)
19             cout<<a[i][j]<<' ';
20         cout<<"\n";
21     }
22     return 0;
23 }
```



题 1-18 B3976 交题解

题目描述：小 S 想要在洛谷上提交一篇非常美妙的题解，但是小 Q 不希望看到小 S 的题解比自己的点赞更多，于是她故意在小 S 的题解中加入了一些奇怪的字符。

具体来说，原本的题解可以看作一个只包含小写或大写字母的字符串 S ，而小 Q 在其中若干位置插入了一些随机字符，这些字符可能是数字或标点符号，但一定不是字母，且一定在键盘上出现。我们称被添加字符后的字符串为 S' 。

你的任务是，将 S' 中非大小写字母的字符全部去除，然后将得到的字符串输出。

输入格式：输入仅一行，一个字符串 S' ，表示被添加字符后的字符串。

输出格式：输出仅一行，一个字符串 S ，表示原字符串。

输入样例 I * Love * Luogu ** 333 # \$ \$	输出样例 ILoveLuogu
数据范围：对于 100% 的数据，保证 S' 的长度为 $1 \sim 10^6$ ， S' 中只出现键盘上主键盘区中的字母、标点符号、数字，不含空格、换行符等结束符号。	

【考点】ASCII 码。

【思路 and 证明】给定一个字符串，删除其中的非英文字母，然后输出结果。

用本题熟悉 ASCII 码。ASCII 码是一种字符编码标准，用于将字符（如字母、数字、标点符号等）转换为计算机可以理解的二进制数值。ASCII 码使用 7 位二进制数（十进制是 $0 \sim 127$ ）表示 128 个字符。

0~31：控制字符，如换行、回车、退格等。

32~126：可打印字符，包括空格、字母、数字、标点符号等。

127：删除字符 DEL。

ASCII 码表中的英文字母，所有大写字母的值按照字母顺序单调递增，所有小写字母的值按照字母顺序单调递增。

如何判断一个字符 c 是英文字母？如果是英文字母，那么 c 既可以是大写也可以是小写，满足下面任意一条即可：

如果是大写字母，那么 c 一定满足 $'A' \leq c \ \&\& \ c \leq 'Z'$ ；

如果是小写字母，那么 c 一定满足 $'a' \leq c \ \&\& \ c \leq 'z'$ 。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int main(){
4      string s;
5      cin >> s;
6      for(int i = 0; i < s.size(); i++)
7          if((s[i] >= 'a' && s[i] <= 'z') || (s[i] >= 'A' && s[i] <= 'Z'))
8              cout << s[i];
9      return 0;
10 }
```

读者可以用 ASCII 码表做练习：输入小写字母，转换成大写字母输出。



题 1-19 P7106 双生独白

题目描述：为了形式化地描述颜色，我们引入 RGB 颜色值，用三元组 (r, g, b) 表示一种颜色，其中 r, g, b 分别为该颜色的 R 值、G 值、B 值，满足 $0 \leq r, g, b \leq 255$ 且皆为十进制整数。

显然，这套颜色系统一共可以表示 $256 \times 256 \times 256 = 16777216$ 种不同的颜色。对于颜色 (r, g, b) ，定义其反色的 RGB 为 $(255 - r, 255 - g, 255 - b)$ 。

然而人们发现，单纯地使用 RGB 颜色值很不方便，复制颜色时要复制 3 个值，于是诞生了十六进制颜色码，即形如 #EBA932 的长度为 7 的字符串。具体而言：

字符串的第一位是 #，为颜色码标识符；

字符串的第二、三位是十六进制数码,拼成的十六进制数等于十进制下颜色的 R 值;
字符串的第四、五位是十六进制数码,拼成的十六进制数等于十进制下颜色的 G 值;
字符串的第六、七位是十六进制数码,拼成的十六进制数等于十进制下颜色的 B 值。
十六进制数码从小到大包含 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,注意 A,B,C,D,E,F 均为大写。

现在你收到了一组十六进制颜色码,请你输出其反色的十六进制颜色码。

输入格式:一行,输入长度为 7 的字符串,表示原色的十六进制颜色码。

输出格式:一行,输出长度为 7 的字符串,表示反色的十六进制颜色码。

输入样例

#EBA932

输出样例

#1456CD

数据范围:对于所有数据,保证给定字符串为合法十六进制颜色码。

【考点】 数据类型的输入/输出。

【思路 and 证明】 用本题熟悉 scanf()、printf() 函数。scanf()、printf() 函数在输入、输出数字时,能按格式处理八进制、十进制、十六进制。%d 是十进制,%o 是八进制,%x 或 %X 是十六进制。例如:

```
1 #include <stdio.h>
2 int main() {
3     int n;
4     scanf("%o", &n);                //输入: 12
5     printf("%o %d %x\n", n, n, n);   //输出: 12 10 a
6     scanf("%x", &n);                //输入: a
7     printf("%o %d %x\n", n, n, n);   //输出: 12 10 a
8     return 0;
9 }
```

【代码展示】 本题最简单的方法是利用 scanf() 和 printf() 函数的输入和输出,按十六进制读入,然后取反,用十六进制输出。

```
1 #include <bits/stdc++.h>
2 int main(){
3     int a;
4     scanf("#%X",&a);                //输入有#,这里不要忘记#
5     printf("#%06X", 0xffffffff - a); //用 0xffffffff 减去 a,结果就是 a 按位取反
6     return 0;
7 }
```

本题还可以模拟,读入每个字符,输出对应的反数。例如,E(二进制 1110)的反数是 1(二进制 0001),B(二进制 1011)的反数是 4(二进制 0100),等等。

```
1 #include <bits/stdc++.h>
2 int main(){
3     getchar();                      //读#
4     putchar('#');
5     for(int i = 1; i <= 6; i++) {
6         char c; c = getchar();
7         if(c == '0')putchar('F'); if(c == '1')putchar('E');
8         if(c == '2')putchar('D'); if(c == '3')putchar('C');
9         if(c == '4')putchar('B'); if(c == '5')putchar('A');
```

```

10     if(c == '6') putchar('9');   if(c == '7') putchar('8');
11     if(c == '8') putchar('7');   if(c == '9') putchar('6');
12     if(c == 'A') putchar('5');   if(c == 'B') putchar('4');
13     if(c == 'C') putchar('3');   if(c == 'D') putchar('2');
14     if(c == 'E') putchar('1');   if(c == 'F') putchar('0');
15 }
16     return 0;
17 }
```



题 1-20 P8115 Table

题目背景：小 A 想要打表过题，但是代码长度过长。他想到了用十六进制表示数字有时比用十进制表示更短，决定把答案表用十六进制进行改写。

题目描述：我们定义下文中的十六进制均用数字 0~9 和大写英文字母 A~F 表示，且每个十六进制数字都包含前缀 0x。

给出若干用花括号包裹，用逗号分隔的十进制整数，代表小 A 的答案表。你需要对答案表内的每个数字进行改写：若该整数使用十六进制表示所占字符数小于或等于十进制表示，则将该数改写为十六进制；否则，该数保留十进制不变。

输出改写后的答案表。

输入格式：输入一行，一个字符串，字符串中仅包含花括号、逗号和数字，不包含空格和其他字符等，代表小 A 的答案表。具体格式可以参考 C++ 语言的数组定义格式，保证答案表格式正确。

输出格式：输出一行，一个字符串，表示改写后的答案表。

输入样例

{1,314159,3141592653589793}

输出样例

{1,314159,0xB29430A256D21}

数据范围：答案表中共有 n 个整数，这些整数分别为 $a_1, a_2, \dots, a_n$ 。 $0 \leq n \leq 10^3$ ， $0 \leq a_i < 2^{64}$ 。

【考点】 格式化输入输出的应用，scanf() 函数的返回值，sprintf() 函数。

【思路 and 证明】 本题最简单的方法是利用 sprintf() 函数，它能把输出的内容转换为字符串，并返回字符串长度。sprintf() 函数的用法见后面的代码。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int main(){
4      unsigned long long n;        //必须用 unsigned long long
5      char a, comma = 0;           //a 用于读取符号(括号或逗号), comma 用于判断是否输出逗号
6      putchar('{');                //输出前括号 {
7      while(scanf("%c%llu", &a, &n) == 2){        //一次输入两个: 符号、数字
8          if(comma) putchar(comma);              //输出逗号: 第一个数字前没逗号, 其他数字前都有
9          char s1[20], s2[20];
10         int len1 = sprintf(s1, "%llu", n);        //返回值是 s1 的长度
11         int len2 = sprintf(s2, "0x%llx", n);      //转换为十六进制
12         if(len1 < len2) cout << s1;              //输出长度短的数字
```

```

13         else          cout << s2;
14         comma = ',';
15     }
16     putchar('}');      //输出后括号 }
17     return 0;
18 }

```

本题的小麻烦是符号(括号和逗号)的输入和输出。

输入符号,代码第7行用字符变量 a 读取符号。

输出逗号时,如果在数字后面输出逗号,由于最后一个数字后面没有逗号,难以处理。可以反过来,在数字前面输出逗号。注意到第一个数字前没有逗号,其他数字前有逗号,用 $comma$ 变量判断是否需要在数字前输出逗号。

第7行还用到了 $scanf()$ 函数的返回值。正常情况下, $scanf()$ 函数的返回值等于输入项数量,如 $scanf("%d%d",&a,&b)$ 成功读入两个整数,返回2。异常情况下,如遇到输入流结束,手动按 $\text{Ctrl}+\text{Z}$ 键,返回 $\text{EOF}(-1)$ 。

如果不用 $\text{sprintf}()$ 函数,本题可以模拟,手工把十进制转换为十六进制。请在学习后面题1-45“B2143 进制转换”后,自己练习。



题 1-21 B4060 位运算 1

题目描述: 基本的位运算共6种,分别为按位与、按位或、按位异或、按位取反、左移和右移。现要求输入3个正整数 a, b, k , 请你输出6行整数, 每行整数分别为以下结果值。

- a 按位与 b ;
- a 按位或 b ;
- a 按位异或 b ;
- a 按位取反;
- a 左移 k 位;
- a 右移 k 位。

输入格式: 输入3个正整数 a, b, k 。

输出格式: 输出6行整数, 含义如题要求。

输入样例

15 12 2

输出样例

12
15
3
-16
60
3

数据范围: $1 \leq a, b \leq 2^{15}$; $1 \leq k \leq 15$ 。

【考点】 基本的位运算。

【思路 and 证明】 表1-2列出了C语言的位运算, 注意负数的位运算结果。

表 1-2 C 语言的位运算

运算符	名 称	示 例
&	按位与	5&3,5 的二进制是 101,3 的二进制是 011,运算结果为 001
	按位或	5 3 的运算结果是二进制 111
^	异或	异或的核心特性: $a \wedge 0 = a$ 、 $a \wedge a = 0$ 、 $(a \wedge b) \wedge b = a$ 经典应用: 交换数、找唯一数。这里的 a 、 b 不论正负,以上都成立
~	按位取反	5 的二进制是 00000101,取反后~5 为 11111010,在有符号数中表示为-6(补码形式)
<<	左移	正整数左移 1 位的结果是乘以 2,左移 k 位是乘以 2^k 。例如, $5 \ll 2$,5 的二进制是 101,左移 2 位后为 10100,十进制是 20 负整数左移 k 位也是乘以 2^k 。例如 $-5(11111011)$, $-5 \ll 1$ 结果为 11110111,十进制是-10
>>	右移	正整数右移 1 位的结果是除以 2,右移 k 位是除以 2^k 。例如, $5 \gg 1$,5 的二进制是 101,右移 1 位后为 010,十进制是 2 负整数右移 k 位也是除以 2^k 。例如 $-5(11111011)$, $-5 \gg 1$ 结果为 11111101,十进制是-3,注意不是一2

【代码展示】 注意每行中的括号不能少,如第 6 行中的 $(a \& b)$ 不能写成 $a \& b$ 。 $\ll$ 的优先级比 $\&$ 高。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int main(){
4      int a,b,k;
5      cin >> a >> b >> k;
6      cout << (a & b) << '\n';    //括号不能少,否则报语法错误
7      cout << (a | b) << '\n';
8      cout << (a ^ b) << '\n';
9      cout << (~a) << '\n';
10     cout << (a << k) << '\n';
11     cout << (a >> k) << '\n';
12     return 0;
13 }
```



题 1-22 B4061 位运算 2

题目描述: 给定正整数 a 和整数 b 。定义正整数 a 的二进制的第 b 位为将 a 转换为二进制后从右往左数的第 b 位,且规定最右侧位是第 0 位。

例如,假设 $a=11$,则对应关系如下,此时我们称 11 的二进制的第 2 位为 0。

位数	4	3	2	1	0
对应数位	0	1	0	1	1

现在需要你依次输出下列 6 个计算结果,一行输出一个整数:

$a \times 2^b$;

$a \div 2^b$,除法向下取整;

a 的二进制的第 b 位;

将 a 的二进制的第 b 位设置为 0 后得到的十进制结果;

将 a 的二进制的第 b 位设置为 1 后得到的十进制结果;

将 a 的二进制的第 b 位取反后得到的十进制结果。

输入格式：输入正整数 a 和整数 b 。

输出格式：输出 6 行，一行一个整数，表示相应问题的答案。

输入样例

11 2

输出样例

44

2

0

11

15

15

数据范围： $1 \leq a \leq 2^{20}, 1 \leq 2^b \leq a$ 。

【考点】基本位运算。

【思路 and 证明】

$a \times 2^b$ ：把 a 左移 b 位。

$a \div 2^b$ ：把 a 右移 b 位。

a 的二进制的第 b 位：先把 a 右移 b 位，这样 a 的第 b 位就移动到了最后一位，再与 1 进行与运算，就得到了它。

将 a 的二进制的第 b 位设置为 0：先把数字 1 左移 b 位，然后取反，再与 a 进行与运算，就把 a 的第 b 位置 0 了。

将 a 的二进制的第 b 位设置为 1：先把数字 1 左移 b 位，再与 a 进行或运算，就把 a 的第 b 位置 1 了。

将 a 的二进制的第 b 位取反：先把数字 1 左移 b 位，再与 a 进行异或运算，就把 a 的第 b 位置取反了。

【代码展示】

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(){
4     long long a,b; cin >> a >> b;
5     cout << (a << b) << '\n';           //a 左移 b 位
6     cout << (a >> b) << '\n';           //a 右移 b 位
7     cout << ((a >> b) & 1) << '\n';       //a 的第 b 位
8     cout << (a & ~(1 << b)) << '\n';     //a 的第 b 位置 0
9     cout << (a | (1 << b)) << '\n';     //a 的第 b 位置 1
10    cout << (a ^ (1 << b)) << '\n';     //a 的第 b 位取反
11    return 0;
12 }
```



题 1-23 P1100 高低位交换

题目描述：给出一个小于 2^{32} 的非负整数，这个数可以用一个 32 位的二进制数表示（不足 32 位用 0 补足）。我们称这个二进制数的前 16 位为“高位”，后 16 位为“低位”。将它的高低位交换，可以得到一个新的数。试问这个新的数是多少（用十进制表示）。

例如,十进制数 1314520 用二进制表示为 0000 0000 0001 0100 0000 1110 1101 1000 (添加了 11 个前导 0 补足为 32 位),其中前 16 位为高位,即 0000 0000 0001 0100;后 16 位为低位,即 0000 1110 1101 1000。将它的高低位进行交换,得到一个新的二进制数 0000 1110 1101 1000 0000 0000 0001 0100,即为十进制的 249036820。

输入格式:一个小于 2^{32} 的非负整数。

输出格式:高低位交换后新的十进制数。

输入样例

1314520

输出样例

249036820

【考点】位运算。

【思路 and 证明】题目要求把低 16 位移动到高 16 位,把高 16 位移动到低 16 位。这可以简单地通过位运算中的左移、右移运算符实现。对于无符号类型整数 a ,进行左移、右移运算时,超过数字表示范围的部分会被丢弃。

例如,对于 `unsigned int a = 7`, $a \ll 30$ 的结果是 $2^{31} + 2^{30}$,而非 7×2^{30} (即 $2^{32} + 2^{31} + 2^{30}$)。使用这个性质, $a \ll 16$ 可以理解为提取出低 16 位并且放到高 16 位上,因为原本的高 16 位恰好被低 16 位向左移动“挤占”而丢弃,而 $a \gg 16$ 可以理解为提取出高 16 位并且放到低 16 位上,因为原本的低 16 位恰好被高 16 位向右移动“挤占”而丢弃。两者相加即为答案。

【代码展示】

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(){
4     unsigned int a;        //确保恰为 32 位无符号整数
5     cin >> a;
6     cout << (a << 16) + (a >> 16);
7 }
```



题 1-24 P1469 找筷子

题目描述:经过一段时间的紧张筹备,计算机小组的“RP 餐厅”终于开业了,这天,经理 LXC 接到了一个订餐大单,可把大家乐坏了!员工们齐心协力按要求准备好了套餐正准备派送时,突然碰到一个棘手的问题——筷子!

CX 小朋友找出了餐厅中所有筷子,但遗憾的是这些筷子长短不一,而我们都知两根筷子需要长度一样才能组成一双,更麻烦的是 CX 找出来的这些筷子数量为奇数,但是巧合的是,这些筷子中只有一根是落单的,其余都成双,善良的你,可以帮 CX 找出这根落单的筷子的长度吗?

输入格式:第一行输入一个整数,表示筷子的数量 n ;第二行输入 n 个整数,第 i 个整数表示第 i 根筷子的长度 a_i 。

输出格式:输出一个整数表示答案。

提示:请注意数据读入对程序效率造成的影响;本题的空间限制为 4MB。

输入样例 9 2 2 1 3 3 3 2 3 1	输出样例 2
数据范围: $1 \leq n \leq 10^7 + 1, 1 \leq a_i \leq 10^9$ 。	

【考点】 异或。

【思路 and 证明】 题目看起来很容易解决,例如先排序,然后统计每个数字的数量。不过,由于题目仅给了 4MB 的空间,而整数有 10^7 个,如果定义一个大小为 10^7 的数组存储所有数字,这远远超过了 4MB。也就是说,不能用数组存数字,只能一边读一边处理。

本题的考点是异或,异或有两个性质:

(1) 自己异或自己等于 0,即 $X \oplus X = 0$;

(2) 0 异或 X 等于 X ,即 $0 \oplus X = X$ 。

根据第一条性质, n 个数连续异或,出现偶数次的数字会被异或为 0。由于本题除了一个整数外其他整数的数量都是偶数个,因此只要逐一异或所有数字,最后的结果就是那个唯一的整数。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int main(){
4      int n;
5      cin >> n;
6      int ans = 0;                //记得把 ans 初始化为 0
7      while(n--){
8          int x;
9          scanf("%d",&x);        //scanf() 比 cin 快
10         ans ^= x;
11     }
12     cout << ans;
13     return 0;
14 }
```

用这道题继续熟悉输入/输出的处理: C++ 语言的输入/输出(cin/cout)和 C 语言的输入/输出(scanf()/printf())。C++ 代码兼容 C 的 scanf()/printf()。

很多队员喜欢用 cin/cout,因为它比 scanf()/printf()写起来要简洁一些。不过在一般情况下,cin/cout 比 scanf()/printf()要慢很多,特别是与 endl 一起用时,更慢。因为 endl 会调用 flush()函数强制刷新缓冲区,而\n 只是输出换行,缓冲区会自动批量刷新,性能差一个数量级。如果输入/输出在几万个以上,scanf()/printf()的速度优势明显。

cin/cout 为什么比 scanf()/printf()慢? 默认情况下,C++会让 cin 与 scanf()、cout 和 printf()保持同步,这个同步机制会带来大量性能开销。cout 默认开启缓冲区刷新,如每次输出都立刻写入终端,也会降低效率。

如果在代码中加入下面两行的优化,可以提升 cin/cout 的效率。

```

ios::sync_with_stdio(false);    //关闭 C++标准流与 C 标准流的同步
cin.tie(0);                     //解除 cin 和 cout 的绑定
```

此时 cin、cout 和 scanf()、printf()一样快。需要注意,此时不应将 cin 与 scanf()、cout

与 printf() 混用, 否则可能造成输入/输出异常。

如果要换行, 用“\n”替换 endl, 更快一些。

本书大部分题目的输入/输出量不大, 不需要优化输入/输出, 直接使用 cin/cout 即可。当输入/输出量很大时, 本书或者使用 scanf()/printf(), 或者加上这两行优化后再使用 cin/cout。



题 1-25 B3650 求和

题目描述: 输入一个整数; 每行输出一个数, 第 i 行表示数字 1 到 i 的和。

输入格式: 一个整数 n 。

输出格式: 输出 n 行, 每行一个整数, 第 i 行表示数字 1 到 i 的和。

输入样例

2

输出样例

1

3

数据范围: $1 \leq n \leq 10^7$ 。

提示: 使用 `cout << endl` 输出换行会导致 TLE(超过时间限制)。请使用 `cout << "\n"` 或者 `printf("\n")` 输出换行。

$1+2+3+\dots+10000000$ 的值超出了 int 类型可以存储的最大值。

【考点】 cout 的效率, 数据类型。

【思路 and 证明】 本题输出有 n 行, 而 n 可能是 10^7 , 非常大。如果输出使用 `cout << endl` 换行会导致 TLE(超时), 应该用 `cout << "\n"` 或 `printf("\n")` 来换行。cout 慢主要是由于它与 endl 一起用, 单纯 cin/cout 效率还可以接受。这是个坑点, 当用 cout 输出大量数据时, 不要用 endl, 而是用“\n”代替。

另外, 要注意计算结果的大小。1 到 i 的和 sum 约等于 $n^2/2$, 当 $n=10^7$ 时, sum 超过了 int 类型的范围, 需要用 long long 类型。

【代码展示】

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(){
4     int n; cin >> n;
5     long long sum = 0;
6     for(int i = 1; i <= n; i++){
7         sum += i;
8         cout << sum << "\n"; //或: printf("%lld\n", sum);
9     }
10    return 0;
11 }
```



题 1-26 B3661 排队

题目描述: 幼儿园里有 n 个小朋友, 每个小朋友有一个学号。老师要求学号是奇数的小朋友站一排, 学号是偶数的小朋友站一排。奇数是个位数为 1、3、5、7、9 的数字, 满足除以 2 的余数为 1。偶数是个位数为 0、2、4、6、8 的数字, 满足除以 2 的余数为 0。

输入格式: 共两行, 第一行输入一个数字 n , 表示有 n 个小朋友; 第二行共输入 n 个数字, 其中第 i 个数字 a_i 表示第 i 个小朋友的学号。保证小朋友的学号既有奇数, 又有偶数。

输出格式：共两行，第一行数字表示学号为奇数的小朋友学号，按照输入的顺序输出。即，如果一个数字输入时在前，那么输出时也应当在前。第二行数字表示学号为偶数的小朋友学号，按照输入的顺序输出。

输入样例

5

5 2 3 1 4

输出样例

5 3 1

2 4

数据范围：对于所有数据，满足 $n \leq 10000, 0 \leq a_i \leq 10^9$ 。

【考点】 数组的应用、奇偶判断。

【思路 and 证明】 定义数组 $a[]$ 存 n 个数。读入 n 个数后，分两次输出，第一次输出 $a[]$ 中的奇数，第二次输出 $a[]$ 中的偶数。

奇偶的判断有两种方法：

(1) 除 2 求余。如果余数是 1，则为奇数，否则为偶数。

(2) 位运算。如果 $a \& 1$ 等于 1，则为奇数，否则为偶数。

【代码展示】

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int a[10005]; // 定义为全局变量
4 int main(){
5     int n;    cin >> n;
6     for(int i = 1; i <= n; i++) cin >> a[i];
7     for(int i = 1; i <= n; i++)
8         if(a[i] % 2 == 1) // 或 if (a[i] & 1)
9             cout << a[i] << " ";
10    cout << endl;
11    for(int i = 1; i <= n; i++)
12        if(a[i] % 2 == 0) // 或 if ((a[i] & 1) == 0), 注意 & 和 == 的优先级
13            cout << a[i] << " ";
14    return 0;
15 }
```

第 3 行定义全局数组 $a[10005]$ 存 n 个数。题目的 n 最大是 10000， $a[]$ 需要定义得比 10000 大一点，防止溢出。

需要特别注意，大数组 $a[10005]$ 应该定义在全局，而不是函数内部。全局变量存储在静态存储区，空间很大。局部变量在栈空间上分配，栈空间可能没那么大，数组太大了可能溢出。

另外，可以用动态数组存数，用 STL 的 `vector` 定义动态数组 b, c ，其中 b 存奇数， c 存偶数。这里提前演示 `vector` 的用法，供读者参考。后续的题目会更详细地介绍 `vector`。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(){
4     int n;
5     cin >> n;
6     vector<int> b, c; // 用 vector 定义动态数组
7     for(int i = 1; i <= n; i++) {
8         int x;    cin >> x;
9         if (x & 1) b.push_back(x); // b[] 存奇数
```

```

10     else        c.push_back(x);           //c[]存偶数
11     }
12     for(int i : b)    cout << i << " ";
13     //或 for(int i = 0; i < b.size(); i++)    cout << b[i] << " ";
14     cout << endl;
15     for(int i : c)    cout << i << " ";
16     //或 for(int i = 0; i < c.size(); i++)    cout << c[i] << " ";
17     return 0;
18 }

```



题 1-27 P1601 A+B Problem(高精)

题目描述：高精度加法，相当于 $A+B$ Problem，不用考虑负数。

输入格式：分两行输入 a, b 。

输出格式：输出只有一行，代表 $a+b$ 的值。

输入样例

1001

9099

输出样例

10100

数据范围： $a, b \leq 10^{500}$ 。

【考点】 高精度加法。

【思路 and 证明】 所谓精度，是指能准确表示多少位的数字。高精度加法，就是编程计算特别大、超出普通整数类型范围的大整数相加。

C 语言的整数类型，最大的 long long 型只有 64 位，如果用来计算超过 64 位的整数，会因为溢出导致精度损失。更大的整数，只能用数组来存储和计算。例如，一个 500 位的十进制数，可以用 `char a[501]` 来存储，其中 $a[i]$ 是它的第 i 位上的数字。两个大整数相加，简单地模拟按位加即可，注意处理进位。

【代码展示】

先把数字输入字符串中，然后用 `add()` 函数求和。先把字符串转换成数字，做加法后再转换回字符。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  char a[1005], b[1005];           //加数和被加数,结果仍存在 a[] 中
4  string add(string sa, string sb){
5      memset(a, 0, sizeof(a));
6      memset(b, 0, sizeof(b));
7      int lena = sa.size(), lenb = sb.size();
8      for(int i = 0; i < lena; i++)
9          a[lena - 1 - i] = sa[i] - '0';           //把字符转换成数字然后翻转,使 a[0] 为最低位
10     for(int i = 0; i < lenb; i++)
11         b[lenb - 1 - i] = sb[i] - '0';
12     int lmax = lena > lenb ? lena : lenb;
13     for(int i = 0; i < lmax; i++) {
14         a[i] += b[i];
15         a[i + 1] += a[i] / 10;           //处理进位
16         a[i] %= 10;
17     }
18     if(a[lmax]) lmax++;           //若最高位相加后也有进位,数字长度加 1

```

```

19     string ans;
20     for(int i = lmax - 1; i >= 0; i--)           //把数字转换成字符,然后翻转
21         ans += a[i] + '0';
22     return ans;
23 }
24 int main(){
25     string sa, sb;
26     cin >> sa >> sb;
27     cout << add(sa, sb);
28     return 0;
29 }

```



题 1-28 P1303 A * B Problem

题目描述：给出两个非负整数，求它们的乘积。

输入格式：输入共两行，每行一个非负整数。

输出格式：输出一个非负整数，表示乘积。

输入样例

1
2

输出样例

2

数据范围：每个非负整数不超过 10^{2000} 。

【考点】高精度乘法。

【思路 and 证明】乘法模拟小学学过的竖式乘法操作，按位做乘法最后相加。例如 87×99 ，计算过程如下。

$$\begin{array}{r}
 87 \\
 \times 99 \\
 \hline
 63 \\
 72_ \\
 63_ \\
 + 72 \\
 \hline
 8613
 \end{array}$$

计算结果用 `int c[]` 存储，首先算出 $c[0] = 7 \times 9 = 63$, $c[1] = 8 \times 9 + 7 \times 9 = 72 + 63 = 135$, $c[2] = 8 \times 9 = 72$ ，然后处理进位，得到乘积 8613。

【代码展示】下面是 C/C++ 代码。注意第 3 行的定义，其中的 $a[]$ 、 $b[]$ 可以定义为 `char`，但是乘法结果 `int c[]`，不能定义为 `char c[]`。例如上面例子中 $c[1] = 135$ ，超出了 `char` 的范围（-128~127），即使改为 `unsigned char`，也可能超出范围。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int a[2005], b[2005], c[4005];           //c = a * b
4  string mul(string sa, string sb){
5      memset(a, 0, sizeof(a));
6      memset(b, 0, sizeof(b));
7      memset(c, 0, sizeof(c));
8      if(sa == "0" || sb == "0") return "0";
9      int lena = sa.size(), lenb = sb.size();
10     for(int i = 0; i < lena; i++) a[lena - i] = sa[i] - '0';
11     for(int i = 0; i < lenb; i++) b[lenb - i] = sb[i] - '0';

```

```

12     for(int i = 1; i <= lena; i++)
13         for(int j = 1; j <= lenb; j++)
14             c[i+j-1] += a[i] * b[j];
15     for(int i = 1; i <= lena + lenb; i++)
16         c[i+1] += c[i]/10, c[i] %= 10;
17     string ans;
18     if(c[lena + lenb])
19         ans += c[lena + lenb] + '0';
20     for(int i = lena + lenb - 1; i >= 1; i--)
21         ans += c[i] + '0';           //倒过来,把 c 的高位放在 ans 前面
22     return ans;
23 }
24 int main(){
25     string sa, sb;
26     cin >> sa >> sb;
27     cout << mul(sa, sb);
28     return 0;
29 }

```



题 1-29 B3842 春游

题目描述：老师带领同学们春游。已知班上有 N 位同学，每位同学有从 0 到 $N-1$ 的唯一编号。到了集合时间，老师确认是否所有同学都到达了集合地点，就让同学们报出自己的编号。到达的同学都会报出自己的编号，不会报出别人的编号，但有的同学很顽皮，会多次报出。你能帮老师找出有哪些同学没有到达吗？

输入格式：输入两行，第一行包含两个整数 N 和 M ，表示班级有 N 位同学，同学们共有 M 次报出编号；第二行包含 M 个整数，分别为 M 次报出的编号。约定所有编号是小于 N 的非负整数。

输出格式：输出一行。如果所有同学都到达，则输出 N ；否则由小到大输出所有未到达的同学编号，空格分隔。

输入样例

3 5
0 0 0 0 0

输出样例

1 2

数据范围：对于所有数据， $2 \leq N \leq 1000$ ， $2 \leq M \leq 1000$ 。

【考点】“桶”的思想。

【思路 and 证明】 定义 $\text{int } a[]$ ，若 $a[i]=1$ ，表示编号为 i 的学生到达了；若 $a[i]=0$ ，表示编号为 i 的学生未到达。读取 M 个编号，对相应的 $a[i]$ 赋值，最后输出 $a[i]=0$ 的编号。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int a[1005];
4  int main(){
5      int n, m;
6      cin >> n >> m;
7      while(m--){int p; cin >> p; a[p] = 1;}           //编号为 p 的同学来了
8      int cnt = 0;                                       //统计没到的人数
9      for(int i = 0; i < n; i++){                       //输出没到的同学

```

```

10         if(a[i] == 0){
11             cnt++;
12             cout << i << " ";           //输出没到的同学的编号
13         }
14     }
15     if(cnt == 0) cout << n;           //如果都到了,输出 n
16     return 0;
17 }
```

用本题介绍“桶”的思想。

将数组 $\text{int } a[]$ 视作“桶”，用其下标表示对应编号的学生是否到达。把每个 $a[i]$ 看成一个桶，若 $a[i]=1$ ，表示这个桶中有数据（编号 i 的学生到了）；若 $a[i]=0$ ，表示这个桶中没有数据（编号 i 的学生没来）。注意桶的大小是所有可能的编号，任何一个学生的编号都能找到对应的桶。

$a[i]$ 的取值可以是 0、1 以外的值，可能如下。

(1) 统计次数。例如， $a[i]=5$ 表示编号 i 的学生出现了 5 次。可以应用在分数段人数统计中。

(2) 存储数据。例如，桶排序中，每个桶（数组元素）可以是一个列表，存放落在该区间的的所有数据。

(3) 状态标记。例如， $a[i]=2$ 表示编号 i 的学生“迟到”， $a[i]=1$ 表示“正常”， $a[i]=0$ 表示“旷课”等。



题 1-30 P1008 三连击

题目背景：本题为提交答案题，您可以写程序或手算，在本机上算出答案后，直接提交答案文本，也可提交答案生成程序。

题目描述：将 1,2,...,9 共 9 个数分成 3 组，分别组成 3 个三位数，且使这 3 个三位数构成 1:2:3 的比例，试求出所有满足条件的 3 个三位数。

输入格式：无

输出格式：若干行，每行 3 个数字。按照每行第 1 个数字升序排列。

输入样例
无

输出样例
192 384 576

...

（剩余部分不予展示）

【考点】 循环、数位拆分。

【思路 and 证明】

方法 1：直接模拟，求 9 个数的全排列，然后判断这 9 数组成的 3 个三位数是否合法。笨方法是写 9 个 for 循环，每个 for 循环选一个数位的数字。第一位 a 枚举范围为 1~9，第二位 b 枚举范围为 1~9，且 b 与 a 不重复，等等。

方法 2：本题有更简洁的枚举方法。枚举第一个三位数 a ，那么第二个三位数是 $2a$ ，第三个三位数是 $3a$ ，然后判断这 3 个数是否包含所有 $1\sim 9$ 。判断方法可以用“桶”，用 $v[1]\sim v[9]$ 这 9 个桶，每用到一个数字 i ，就置 $v[i]=1$ ；如果 $v[1]\sim v[9]$ 都是 1，说明 $1\sim 9$ 都用到了。第一个三位数的范围为 $123\sim 333$ 。

【代码展示】 编码时用到数位拆分，对于每个三位数，拆出每位的数字。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int v[10]; //桶: v[i] = 1 表示数字 i 用到了
4  int main(){
5      for(int a = 123; a < 333; a++) { //枚举第一个三位数: 从 123 到 333
6          memset(v, 0, sizeof(v)); //注意: 每次枚举时把 v[] 清空
7          int b = 2 * a, c = 3 * a;
8          v[a%10] = v[a/10%10] = v[a/100%10] = 1; //第一个数的个、十、百位
9          v[b%10] = v[b/10%10] = v[b/100%10] = 1; //第二个数的个、十、百位
10         v[c%10] = v[c/10%10] = v[c/100%10] = 1; //第三个数的个、十、百位
11         int cnt = 0;
12         for(int j = 1; j <= 9; j++)
13             if(v[j]) cnt++;
14         if(cnt == 9) cout << a << " << b << " << c << "\n"; //9 个数字都用到了, 输出
15     }
16     return 0;
17 }
```

求全排列有更好的方法：用深度优先搜索(depth-first search, DFS)求全排列，或者用 `next_permutation()` 函数求全排列。“第 2 篇 模拟与枚举”将会讲解 `next_permutation()` 函数。



题 1-31 P5741 旗鼓相当的对手

题目描述：现有 N 名同学参加了期末考试，并且获得了每名同学的信息，包括姓名（不超过 8 个字符的字符串，没有空格），语文、数学、英语成绩（均为不超过 150 的自然数）。如果某对学生 $\langle i, j \rangle$ 的每科成绩的分差都不大于 5，且总分分差不大于 10，那么这对学生就是“旗鼓相当的对手”。现在我们想知道这些同学中，哪些是“旗鼓相当的对手”？请输出他们的姓名。

所有人的姓名是按照字典序给出的，输出时也应该按照字典序输出所有对手组合。也就是说，这对组合的第一个名字的字典序应该小于第二个名字；如果两个组合中第一个名字不同，则第一个名字字典序小的先输出；如果两个组合的第一个名字相同但第二个名字不同，则第二个名字字典序小的先输出。

输入格式：第一行输入一个正整数 N ，表示学生个数。第二行开始，往下 N 行，每行首先输入一个字符串表示学生姓名，再输入 3 个自然数表示语文、数学、英语的成绩。均用空格相隔。

输出格式：输出若干行，每行两个以空格隔开的字符串，表示一组旗鼓相当的对手。注意题目描述中的输出格式。

输入样例

```

3
fafa 90 90 90
lxl 95 85 90
senpai 100 80 91
```

输出样例

```

fafa lxl
lxl senpai
```

数据范围: $1 \leq N \leq 1000$, 姓名为长度不超过 8 的字符串, 语文、数学、英语成绩均为不超过 150 的自然数。

【考点】 结构体。

【思路 and 证明】 定义结构体存储学生的信息。定义 $\text{cmp}(x, y)$ 函数比较 x, y 是否旗鼓相当。

【代码展示】 题目要求按字典序输出, 由于输入时已经是字典序, 所以输出自然是按字典序的, 代码不需要特殊处理。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  struct{
4      string name;                //名字
5      int a,b,c,t = 0;            //三门课的分数与总分
6  } s[1005];
7  int cmp(int x,int y){           //比较 x,y 是否旗鼓相当
8      if(abs(s[x].a - s[y].a) <= 5 &&
9         abs(s[x].b - s[y].b) <= 5 &&
10         abs(s[x].c - s[y].c) <= 5 &&
11         abs(s[x].t - s[y].t) <= 10) return 1;
12     else return 0;
13 }
14 int main(){
15     int n; cin>>n;
16     for(int i = 1;i <= n;i++){
17         cin>> s[i].name>> s[i].a>> s[i].b>> s[i].c;
18         s[i].t = s[i].a + s[i].b + s[i].c;
19     }
20     for(int i = 1;i <= n;i++)        //按顺序进行循环, 比较
21         for(int j = i+1;j <= n;j++)
22             if(cmp(i,j))            //判断 i,j 是否是旗鼓相当
23                 cout<< s[i].name<<" "<< s[j].name<<'\n';
24     return 0;
25 }
```



题 1-32 B3957 完全平方数

题目描述: 小杨同学有一个包含 n 个非负整数的序列 A , 他想要知道其中有多少对下标组合 $\langle i, j \rangle$ ($1 \leq i < j \leq n$), 使得 $A_i + A_j$ 是完全平方数。如果 x 是完全平方数, 则存在非负整数 y 使得 $y^2 = x$ 。

输入格式: 第一行输入一个非负整数 n , 表示非负整数个数。第二行输入 n 个非负整数 $A_1, A_2, \dots, A_n$, 表示序列 A 包含的非负整数。

输出格式: 输出一个整数表示答案。

输入样例

5

1 4 3 3 5

输出样例

3

数据范围: $1 \leq n \leq 1000, 1 \leq A_i \leq 10^5$ 。

【考点】 循环、枚举。

【思路 and 证明】 通过本题了解“枚举”的解题思路。枚举就是对所有情况进行考查,从这些情况中找到符合题目要求的解。

枚举所有 $\langle i, j \rangle$ 组合, i 从 1 到 n , j 从 $i+1$ 到 n , 共有约 $n^2/2$ 种组合。对每个组合, 判断是否为完全平方数。判断方法: 若一个正整数 a 是完全平方数, 则其平方根必然是正整数 b , 此时 $b^2 = a$; 若 a 不是完全平方数, 其平方根下取整的值为 c , 则 c^2 必然不是 a 。使用整数变量 $b = \text{sqrt}(a)$, 判断 b^2 是否等于 a 即可。sqrt() 是开平方根函数, 在 cmath.h 头文件中。

【代码展示】 下面代码第 8、9 行, 用两重 for 循环枚举了所有 $\langle i, j \rangle$ 组合。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int a[1005];
4 int main(){
5     int n;    cin >> n;
6     for(int i = 1; i <= n; i++)    cin >> a[i];
7     int ans = 0;
8     for(int i = 1; i <= n; i++)
9         for(int j = i + 1; j <= n; j++){           //用两重 for 循环枚举所有< i, j>组合
10             int m = a[i] + a[j];
11             int t = sqrt(m);                       //开方后转换为整数赋值给 t
12             if(t * t == m)    ans++;               //找到一个完全平方数
13         }
14     cout << ans;
15     return 0;
16 }
```

枚举又称为穷举, 由于需要计算所有情况, 计算量往往非常大。枚举是一种低效的算法, 不过因为思路和编码很简单, 当计算规模不大时, 可以使用它。有时找不到更好的算法, 只能用枚举法。

我们将在第 2 篇中学习更多关于枚举的知识点。



题 1-33 B3674 标题修改

题目描述: 某 E 写了一篇文章, 标题为字符串 S , 由小写英文字符和空格组成。

某 E 恪守规范, 因此, 字符串 S 是由若干单词组成的, 单词之间由一个空格隔开。

编辑人员要求对单词中字母的大小写作调整。具体地, 对每个单词, 编辑人员要求将每个单词中的第奇数个字母改成大写, 第偶数个字母改成小写。

关于单词下标的定义: 以 1 为起始下标。也就是说, 对一个长度为 n 的单词, 我们将字母从头到尾记作第 1 个, 第 2 个, ..., 第 n 个。

举例说明, 对于单词 apple:

第一个字母为 a;

第二个字母为 p;

第三个字母为 p;

第四个字母为 l;

第五个字母为 e。

某 E 要赶进度,请你帮她完成这个任务。

输入格式:输入一行,一个字符串 S 。

输出格式:输出一行,一个字符串,代表修改后的标题。

输入样例

i like eat apple

输出样例

I LiKe EaT ApPlE

数据范围:去除空格后的 S 长度 $\leq 10^6$ 。

【考点】 字符串的读入处理,奇偶判断。

【思路 and 证明】 逐个读入单词,然后把单词的奇数位置改成大写。

【代码展示】 注意,题目要求把奇数位置的字母改成大写,又说明单词以 1 为起始下标,所以 $s[0]$ 是第 1 个。那么当 $s[i]$ 的 i 是偶数时,把 $s[i]$ 改成大写。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(){
4     string s;
5     while(cin>>s) {                //逐个读入单词
6         int len = s.length();
7         for(int i = 0; i < len; i++)
8             if(i%2 == 0)            //i 是偶数
9                 s[i] = s[i] - 'a' + 'A'; //改成大写。或者: s[i] = toupper(s[i])
10        cout << s << " ";          //注意补上单词间的空格
11    }
12    return 0;
13 }
```

大小写转换还可以用 STL 函数 `toupper()`、`tolower()`。

`toupper()`: 小写字母转大写字母,如果输入的不是小写字母(如大写字母、数字、符号),则返回原字符。

`tolower()`: 大写字母转小写字母,如果输入的不是大写字母(如小写字母、数字、符号),则返回原字符。

要简便地操作字符串,需要学习 STL string 库。部分函数如下。

`length()`: 返回字符串的长度。

`size()`: 返回字符串的长度,和 `length()` 功能一样。

`push_back()`: 在字符串尾部添加一个字符。

`append()`: 在字符串尾部添加一个字符串。

`find(str,pos)`: 查找 `str` 在 `pos`(含)之后第一次出现的位置。

`substr(pos,len)`: 返回从 `pos` 位置开始截取最多 `len` 个字符组成的字符串。

`insert(index,count,c)`: 在 `index` 处连续插入 `count` 次字符 `c`。

`insert(index,str)`: 在 `index` 处插入字符串 `str`。

`erase(index,count)`: 将字符串 `index` 位置开始(含)的 `count` 个字符删除。

`replace(pos,count,str)`: 把从 `pos` 位置开始 `count` 个字符的子串替换为 `str`。



题 1-34 B3998 小苏的键盘

题目描述:小苏同学有一个文本编辑器和一个奇怪的键盘。这个键盘有 26 个小写

英文字母和退格键(backspace),一共 27 个键。每次当她按下任何一个小写英文字母键时,文本编辑器就会在当前编辑文本的末端添加对应的字母。例如,假设当前文本是 luog,当她按下 u 键时,文本就会变成 luogu。当她按下退格键时,文本编辑器就会删除当前文本的最后一个字母。如果当前文本是空的,则什么都不会发生。例如,如果当前文本是 luogu,当她按下退格键后,文本就会变成 luog。现在,给定小苏的按键情况,已知在初始时文本为空,请你求出小苏按完给定的所有键后的文本是什么。

输入格式: 第一行输入一个整数 n ,表示按键的次数;第二行输入 n 个用空格隔开的字符串,依次表示小苏按下的每个按键。输入的每个字符串要么是一个小写字母,表示对应的按键,要么是字符串 <bs>,表示退格键。

输出格式: 输出一行,一个字符串,表示小苏按完给定的所有按键后的文本。数据保证输出不是空串。

输入样例

3

a <bs> b

输出样例

b

数据范围: $1 \leq n \leq 100$ 。

【考点】 字符串处理。

【思路 and 证明】 模拟题目的要求,读入 n 个字符串,根据读入的是字符还是退格键逐个处理输入的字符串。当读到退格键时,调用 substr() 函数。

【代码展示】

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  int main(){
4      int n;
5      cin>>n;
6      string s;
7      while(n--){
8          string t;
9          cin>>t;
10         if(t=="<bs>") s = s.substr(0,s.size()-1); //把 s 的最后一个字符删除
11         else s += t; //或者 else s.append(t);
12     }
13     cout<<s;
14     return 0;
15 }
```

使用 cin 读入 string 类型字符串会自然地以空格隔断,因此不用处理空格带来的问题。

注意第 11 行,把字符串 t 加到字符串 s 的末尾,简单地用 + 即可,比用 append() 函数简洁。

第 10 行的 substr() 函数的计算复杂度为 $O(n)$,第 7 行输入 n 个,代码的总计算复杂度为 $O(n^2)$ 。



题 1-35 B3769 制糊串

题目描述: 在这个问题中,我们用 $s[x,y]$ 表示从字符串 s 的第 x 个字符到第 y 个字符连起来构成的字符串。例如,若 $s=abcdef$,则 $s[2,4]=bcd$ 。

给出两个字符串 s 和 t , 有 q 次询问。每次给出 l_1, r_1 和 l_2, r_2 , 请判断 $s[l_1, r_1]$ 和 $t[l_2, r_2]$ 谁的字典序更小。

输入格式: 第一行输入一个字符串 s 。第二行输入一个字符串 t 。第三行输入一个整数, 表示询问次数 q 。接下来 q 行, 每行 4 个整数 l_1, r_1, l_2, r_2 , 表示一次询问。

输出格式: 对每次询问, 输出一行一个字符串:

如果 $s[l_1, r_1]$ 的字典序更小, 请输出 yifusuyi;

如果 $t[l_2, r_2]$ 的字典序更小, 请输出 erfusuer;

如果两者的字典序一样大, 请输出 ovo。

输入样例

Yifusuyi

yifusuYi

3

1 2 7 8

1 2 1 2

7 8 7 8

输出样例

ovo

yifusuyi

erfusuer

数据范围: $1 \leq |s|, |t|, q \leq 10^3$; $1 \leq l_1 \leq r_1 \leq |s|$; $1 \leq l_2 \leq r_2 \leq |t|$ 。输入字符串仅含大小写英文字母。

【考点】 取子串, 字符串比较。

【思路 and 证明】 用 `substr(pos, len)` 函数取子串, 返回从 `pos` 位置开始截取最多 `len` 个字符组成的字符串。

取子串时注意范围。题目要求字符串下标从 1 开始, 但字符串 s 的第一个字符是 $s[0]$, 所以应该是 $s.substr(l_1 - 1, r_1 - l_1 + 1)$ 。

【代码展示】 字符串的比较有两种常见方法: 一种是使用 `compare()` 函数, 另一种是使用关系运算符 `>`、`<`、`==`、`!=`。下面的代码用最简单的关系运算符 `>`、`<`、`==` 进行比较。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main() {
4     string s, t;
5     cin >> s >> t;
6     int q;
7     cin >> q;
8     while(q--) {
9         int l1, r1, l2, r2;
10        cin >> l1 >> r1 >> l2 >> r2;
11        string a = s.substr(l1 - 1, r1 - l1 + 1);
12        string b = t.substr(l2 - 1, r2 - l2 + 1);
13        if(a < b) cout << "yifusuyi" << endl; //直接用关系运算符比较字符串
14        if(a > b) cout << "erfusuer" << endl;
15        if(a == b) cout << "ovo" << endl;
16    }
17    return 0;
18 }
```

**题 1-36 B3997 小洛的字符串分割**

题目描述：对于一个字符串 S ，小洛定义它为回文串，当且仅当字符串 S 从左往右读和从右往左读一样。例如， $abcba$ 是回文串，而 $abcca$ 不是回文串。

小洛现在有一个字符串 S ，他想将这个字符串分为若干段，段长度分别为 $1, 2, \dots$ 。具体而言，他会先将第一个字符拿出来作为字符串 S_1 ，再将第 2、3 个字符拿出来作为 S_2 ，再将第 4、5、6 个字符拿出来作为 S_3 ，以此类推。最后若还有多余的字符，则单独作为一段。

例如，对于字符串 $aaababcaacd$ ，会被分为如下 5 个字符串：

$$S_1 = a$$
$$S_2 = aa$$
$$S_3 = bab$$
$$S_4 = caac$$
$$S_5 = d$$

字符串 $aaababcaacd$ 分割出的 5 个字符串都是回文串。

小洛想要知道，对于读入的字符串 S ，这些被分割出来的字符串，有多少个是回文串呢？

输入格式：输入一行，一个字符串 S 。

输出格式：输出一个整数，表示答案。

输入样例

$aaababcaacd$

输出样例

5

数据范围： $1 \leq |S| \leq 10^6$ ，字符串仅包含小写英文字母。

【考点】 字符串翻转。

【思路 and 证明】 逐个读取字符串，判断是否为回文串。

【代码展示】

本题的核心考点是判断一个字符串是否为回文串。回文串的定义是：一个字符串从左至右读取的结果与从右至左读取的结果完全一致（如 $"abba"$ 和 $"abcba"$ 为回文串）。基于这一定义，可设计如下判断方法：将待判断的字符串 s 进行翻转得到新字符串 t ，若 s 与 t 的内容完全相同，说明 s 是回文串。

在 C++ 的 `algorithm` 库中，提供了 `reverse()` 函数用于翻转容器（包括字符串）的内容。需要注意的是，`reverse()` 函数会直接修改原字符串，而非返回新的翻转字符串。翻转字符串 s 的代码为 `reverse(s.begin(), s.end())`。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main(){
4     int p = 0;                //当前处理到第 p 个字符
5     int len = 1;              //当前处理的子串长度,从长度 1 开始
6     int cnt = 0;              //统计回文串
7     string s;
8     cin >> s;
```

```

9      while(p < s.length()) {
10         string subs = s.substr(p, len);
11         p += len;
12         len++;
13         string t = subs;
14         reverse(t.begin(), t.end());           //翻转字符串
15         if(t == subs)    cnt++;                //判断是否为回文串
16     }
17     cout << cnt;
18 }
```



题 1-37 B3868 进制判断

题目描述： N 进制数指的是逢 N 进一的计数制。例如，人们日常生活中大多使用十进制计数，而计算机底层一般则使用二进制。除此之外，八进制和十六进制在一些场合也是常用的计数制（十六进制中，一般使用字母 A~F 表示 10~15）。

现在有 N 个数，请分别判断它们是否可能是二进制数、八进制数、十进制数、十六进制数。例如，15A6F 就只可能是十六进制数，而 1011 则是 4 种进制数皆有可能。

输入格式：第一行输入一个十进制整数 N 。接下来 N 行，每行输入一个字符串，表示需要判断的数。保证所有字符串均由数字和大写字母组成，可能以 0 开头。保证不会出现空行。

输出格式：输出 N 行，每行 4 个数，用空格隔开，分别表示给定的字符串是否可能表示一个二进制数、八进制数、十进制数、十六进制数。1 表示可能，0 表示不可能。

例如，对于只可能是十六进制数的 15A6F，就需要输出 0 0 0 1；而对于四者皆有可能的 1011，则需要输出 1 1 1 1。

输入样例

2

15A6F

1011

输出样例

0 0 0 1

1 1 1 1

数据范围：保证 $1 \leq N \leq 1000$ ，保证所有字符串长度不超过 10。

【考点】字符串处理。

【思路 and 证明】 本题并不需要进制转换，只需要判断能不能转换。判断以下情况：

- (1) 二进制只有 0、1，如果有大于 1 的字符，不是二进制。
- (2) 八进制只有 0~7，如果有大于 7 的字符，不是八进制。
- (3) 十进制只有 0~9，如果有大于 9 的字符，不是十进制。
- (4) 十六进制有 0~9、A~F，如果有大于 F 的字符，不是十六进制。

【代码展示】 定义数组 a ，用 $a[0]$ 、 $a[1]$ 、 $a[2]$ 、 $a[3]$ 分别存储二进制、八进制、十进制、十六进制的判断结果。例如， $a[0]=1$ 说明能表示为二进制， $a[0]=0$ 说明不能表示为二进制。

本题需要对 n 个字符串进行判断，对每个字符串，在第 6 行用局部变量重新定义并初始化判断结果 $a[]$ ，保证不同的字符串的 $a[]$ 互不影响。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int main(){
4      int n;    cin >> n;
5      while(n-- ) {
6          int a[4] = {1, 1, 1, 1};    //定义并赋初值,默认能表示为某进制
7          string s;
8          cin >> s;
9          for(int i = 0; i < s.length(); i++) {    //依次判断字符串中每个字符
10             if(s[i] > '1')    a[0] = 0;    //大于 1,不是二进制
11             if(s[i] > '7')    a[1] = 0;    //大于 7,不是八进制
12             if(s[i] > '9')    a[2] = 0;    //大于 9,不是十进制
13             if(s[i] > 'F')    a[3] = 0;    //大于 F,不是十六进制
14         }
15         cout << a[0] << ' ' << a[1] << ' ' << a[2] << ' ' << a[3] << endl;
16     }
17     return 0;
18 }

```



题 1-38 P1308 统计单词数

题目描述：一般的文本编辑器都有查找单词的功能,该功能可以快速定位特定单词在文章中的位置,有的还能统计出特定单词在文章中出现的次数。

现在,请你编程实现这一功能,具体要求是给定一个单词,输出它在给定的文章中出现的次数和第一次出现的位置。注意,匹配单词时,不区分大小写,但要求完全匹配,即给定单词必须与文章中的某一独立单词在不区分大小写的情况下完全相同,如果给定单词仅是文章中某一单词的一部分则不算匹配。

输入格式：共两行。第一行为一个字符串,其中只含字母,表示给定单词;第二行为一个字符串,其中只可能包含字母和空格,表示给定的文章。

输出格式：如果在文章中找到给定单词则输出两个整数,两个整数之间用一个空格隔开,分别是单词在文章中出现的次数和第一次出现的位置(即在文章中第一次出现时,单词首字母在文章中的位置,位置从 0 开始);如果单词在文章中没有出现,则直接输出一个整数-1。

注意：空格占一个字母位。

输入样例

To

to be or not to be is a question

输出样例

2 0

数据范围：第一行单词长度 ≤ 10 ,文章长度 $\leq 10^6$ 。

【考点】 STL 字符串处理函数 getline()、substr()、tolower()。

【思路 and 证明】 本题的文章不长,可以按题目要求简单模拟:从前到后逐一检查每个位置开始的单词,查询是否为特定单词。

是否可以这样编码:用 cin 逐个读入单词(cin 会自动跳过单词之间的空格,也就是忽略空格分隔符),并把每个单词与给定单词匹配。这样做虽然可以统计得到特定单词的出现次数,但是无法得到第一次出现的位置。因为有一个容易被忽视的关键问题——单词之间的空格数量是未知的。而特定单词的位置与单词的长度相关,也和空格数量有关,单词长度可

以知道,但是单词之间的空格数量未知。

所以,必须把文章中的空格也当成普通字符处理。此时需要用 `getline()` 函数,它能完整读取包括空格的整个字符串。

还有一个问题是,要求文章内的单词和特定单词完全匹配,不能只匹配一部分。这里采用一个技巧:把每个单词前后的空格也看成单词的一部分,把特定单词的头尾也加上空格,连同空格一起匹配,这样就避免了部分匹配。另外,文章的头和尾是特殊情况,第一个单词前面和最后一个单词后面没有空格,需要补上空格。

例如,文章是 $s = \text{"to be or too to"}$,头尾加上空格是 $s' = \text{"to be or too to"}$ 。查特定单词 $p = \text{"to"}$,头尾加上空格是 $p' = \text{"to"}$ 。在 s' 中查询 p' ,避免了部分匹配问题。

【代码展示】 用 `getline()` 函数读包含空格在内的所有字符串,用 `tolower()` 函数转换成小写,用 `substr()` 函数匹配。请自己熟悉其他字符串处理函数。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  string turn(string s) {                                //转换成小写,并且头尾加空格
4      for(int i = 0; i < s.size(); i++)
5          s[i] = tolower(s[i]);
6      return ' ' + s + ' ';
7  }
8  int main() {
9      string s1, s2;
10     getline(cin, s1); s1 = turn(s1);
11     getline(cin, s2); s2 = turn(s2);
12     int pos = -1;                                       //记录第一次出现的位置
13     int cnt = 0;                                       //统计匹配的数量
14     for(int i = 0; i < s2.size(); i++)                 //暴力查询每个位置是否有特定单词
15         if(s2.substr(i, s1.size()) == s1){           //找到了
16             if(pos < 0) pos = i;                       //记录第一次出现的位置
17             cnt++;
18         }
19     if(cnt) cout << cnt << ' ' << pos;
20     else cout << "-1";
21     return 0;
22 }
```



题 1-39 P1055 ISBN 码

题目描述: 每本正式出版的图书都有一个 ISBN 码与之对应,ISBN 码包括 9 位数字、1 位识别码和 3 位分隔符,其规定格式如 $x\text{-}xxx\text{-}xxxxx\text{-}x$,其中符号-就是分隔符(键盘上的减号),最后一位是识别码。例如 $0\text{-}670\text{-}82162\text{-}4$ 就是一个标准的 ISBN 码。ISBN 码的首位数字表示书籍的出版语言,如 0 代表英语;第一个分隔符-之后的 3 位数字代表出版社,如 670 代表维京出版社;第二个分隔符后的 5 位数字代表该书在该出版社的编号;最后一位为识别码。识别码的计算方法如下。

首位数字乘以 1 加上次位数字乘以 2,以此类推,用所得的结果 $\text{mod } 11$,所得的余数即为识别码,如果余数为 10,则识别码为大写字母 X。例如,ISBN 码 $0\text{-}670\text{-}82162\text{-}4$ 中的识别码 4 是这样得到的:对 067082162 这 9 个数字,从左至右,分别乘以 $1, 2, \dots, 9$ 再求和,即 $0 \times 1 + 6 \times 2 + \dots + 2 \times 9 = 158$,然后取 $158 \text{ mod } 11$ 的结果 4 作为识别码。

你的任务是编写程序判断输入的 ISBN 码中识别码是否正确,如果正确,则仅输出 Right; 如果错误,则输出你认为是正确的 ISBN 码。

输入格式: 一个字符序列,表示一本书的 ISBN 码(保证输入符合格式要求)。

输出格式: 一行,假如输入的 ISBN 码的识别码正确,那么输出 Right,否则,按照规定的格式,输出正确的 ISBN 码(包括分隔符-)。

输入样例

0-670-82162-4

输出样例

Right

【考点】 字符串处理。

【思路 and 证明】 模拟题目的要求,对 ISBN 字符串进行处理。

【代码展示】

```
1 #include <iostream>
2 using namespace std;
3 int main(){
4     string mod = "0123456789X";
5     string a;
6     cin >> a;
7     int t = 0, cnt = 1;
8     for(int i = 0; i < 12; i++){
9         if(a[i] == '-') continue;
10        t += (a[i] - '0') * cnt;
11        cnt++;
12    }
13    if(mod[t%11] == a[12]) cout << "Right";
14    else{ a[12] = mod[t%11]; cout << a; }
15    return 0;
16 }
```



题 1-40 P11227 扑克牌

题目描述: 小 P 从同学小 Q 那儿借来一副包含 n 张牌的扑克牌。

本题中我们不考虑大小王,此时每张牌具有两个属性:花色和点数。花色共有 4 种:方片、草花、红桃和黑桃。点数共有 13 种,从小到大分别为 A 2 3 4 5 6 7 8 9 T J Q K。注意:点数 10 在本题中记为 T。

我们称一副扑克牌是完整的,当且仅当对于每种花色和每种点数,都恰好有一张牌具有对应的花色和点数。由此,一副完整的扑克牌恰好有 $4 \times 13 = 52$ 张牌。

小 P 借来的牌可能不是完整的,为此小 P 准备再向同学小 S 借牌若干。可以认为小 S 每种牌都有无限张,因此小 P 可以任意选择借来的牌。小 P 想知道他至少得向小 S 借多少张牌,才能让从小 S 和小 Q 借来的牌中,可以选出 52 张牌构成一副完整的扑克牌。

为了方便输入,我们使用字符 D 代表方片,字符 C 代表草花,字符 H 代表红桃,字符 S 代表黑桃,这样每张牌可以通过一个长度为 2 的字符串表示,其中第一个字符表示这张牌的花色,第二个字符表示这张牌的点数。例如,CA 表示草花 A,ST 表示黑桃 T (黑桃 10)。

输入格式：第一行输入一个整数 n ，表示牌数。接下来 n 行，每行输入一个长度为 2 的字符串描述一张牌，其中第一个字符描述其花色，第二个字符描述其点数。

输出格式：输出一个整数，表示最少还需要向小 S 借几张牌才能凑成一副完整的扑克牌。

输入样例	输出样例
1 SA	51

【考点】 数组的应用。

【思路 and 证明】 题意是有 n 张扑克牌，有一些是重复的。去掉重复的牌，问还需要多少张牌，才能凑出一副完整的牌。如果学过 STL set，去重非常简单。这里不用 set，而是使用桶思想，用数组记录所有牌，最后统计缺少的牌。

在“第 4 篇 数据结构”中，将重新用 set 求解本题。

【代码展示】 定义 $p[][]$ 表示所有花色的牌，第一维是花色，第二维是点数。读取已经有的牌，记录在 $p[][]$ 中。最后统计没有记录的牌，就是缺失的牌。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int p[5][15];                                //p[][], 第一维是花色, 第二维是点数
4  int num(char x){                              //返回点数
5      if('2' <= x && x <= '9') return x - '0';
6      if(x == 'A') return 1;
7      if(x == 'T') return 10;
8      if(x == 'J') return 11;
9      if(x == 'Q') return 12;
10     if(x == 'K') return 13;
11 }
12 int main() {
13     int n; cin >> n;
14     for(int i = 1; i <= n; i++){                //记录每张牌
15         char c1, c2;
16         cin >> c1 >> c2;
17         if(c1 == 'D') p[1][num(c2)] = 1;
18         if(c1 == 'C') p[2][num(c2)] = 1;
19         if(c1 == 'H') p[3][num(c2)] = 1;
20         if(c1 == 'S') p[4][num(c2)] = 1;
21     }
22     int cnt = 0;
23     for(int i = 1; i <= 4; i++)                  //4 种花色
24         for(int j = 1; j <= 13; j++)              //13 张牌
25             if(!p[i][j]) cnt++;                  //缺这张牌, 统计数量
26     cout << cnt;
27     return 0;
28 }
```



题 1-41 B3918 图像变换

题目描述：一张字符图片由 n 行 m 列共 $n \times m$ 个字符组成，第 i 行第 j 列的字符为 $s_{i,j}$ 。一张 4×3 的字符图片如下。

%%%

```
$ $ $
```

```
@w@
```

```
!!!
```

现在,需要将图像放大 k 倍,得到 $kn \times km$ 的图片。原图片的每个字符都需要重复 k^2 次,作为新图像中 $k \times k$ 的一个区域,各字符的相对位置不变。

将上面给出的例子放大 2 倍,将得到如下图片。

```
% % % % % %
```

```
% % % % % %
```

```
$ $ $ $ $ $
```

```
$ $ $ $ $ $
```

```
@ @ ww @ @
```

```
@ @ ww @ @
```

```
!!!!!!
```

```
!!!!!!
```

输入格式: 输入 $n+1$ 行,第一行为 3 个整数 n, m, k ; 接下来 n 行,每行 m 个字符,表示字符图片。

输出格式: 输出 nk 行,每行 mk 个字符,表示变换后的字符图片。

输入样例

```
4 3 2
```

```
% % %
```

```
$ $ $
```

```
@w@
```

```
!!!
```

输出样例

```
% % % % % %
```

```
% % % % % %
```

```
$ $ $ $ $ $
```

```
$ $ $ $ $ $
```

```
@ @ ww @ @
```

```
@ @ ww @ @
```

```
!!!!!!
```

```
!!!!!!
```

数据范围: $1 \leq n, m \leq 100$; $1 \leq k \leq 10$; 输入的字符仅包含 ASCII 码不超过 127 的可见字符。

【考点】 二维数组。

【思路 and 证明】 用字符数组 $s[i][j]$ 存原字符图片第 i 行第 j 列的字符。把原字符图片扩大 k 倍,就是将一个字符,横向重复 k 次,纵向重复 k 次,一共重复 k^2 次。下面给出的代码直接模拟了这种做法。

【代码展示】

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 char s[105][105];
4 int main(){
5     int n,m,k;
6     cin >> n >> m >> k;
7     for(int i = 1; i <= n; i++)           //输入 n 行
```

```

8      for(int j = 1; j <= m; j++)          //输入 m 列
9          cin >> s[i][j];
10     for(int i = 1; i <= n; i++)          //原来有 n 行
11         for(int x = 1; x <= k; x++) {    //行扩展 k 次, 现在有 n * k 行
12             for(int j = 1; j <= m; j++)    //原来有 m 列
13                 for(int y = 1; y <= k; y++) //列扩展 k 次, 现在有 m * k 列
14                     cout << s[i][j];      //这一行有 m * k 个, 打印出来
15                 cout << "\n";            //换行
16             }
17     return 0;
18 }
```

下面介绍另一种方法,理解起来更容易一些。

设扩展后的新字符数组为 $t[][]$, $t[x][y]$ 与 $s[i][j]$ 的对应关系是

$$(i-1) * k + 1 \leq x \leq i * k$$

$$(j-1) * k + 1 \leq y \leq j * k$$

这种方法请读者自己编码。

1.2

递 归



题 1-42 P5739 计算阶乘

题目描述: 求 $n!$, 也就是 $1 \times 2 \times 3 \times \cdots \times n$ 。

挑战: 不用循环语句(for、while)完成这个任务。

输入格式: 第一行输入一个正整数 n 。

输出格式: 输出一个整数, 表示 $n!$ 。

输入样例

3

输出样例

6

数据范围: 保证 $1 \leq n \leq 12$ 。

【考点】 递归。

【思路 and 证明】 用本题练习递归。当一个问题可以被拆解为若干规模更小、结构完全相同的子问题时,递归是解决这类问题的标准编程方法。

递归有以下要点。

(1) 问题规模逐步减小。递归过程中,原问题会被持续拆解为规模更小的同类型子问题,直到问题规模缩小到“无须再拆解”的程度。递归的核心思想可概括为“自顶向下拆解问题,从宏观到微观求解”。

(2) 逻辑结构复用。无论问题规模大小,求解子问题的核心逻辑(计算规则)完全一致,因此可以将这一通用逻辑封装为一个函数(递归函数),通过反复调用自身实现逻辑复用,避免重复编写代码。

(3) 明确的终止条件。当问题被拆解到“最小可解单元”(即递归的终止条件)时,该最小问题的解是已知的、可直接返回的,无须继续递归调用,这是避免递归无限循环的关键。

(4) 结果回溯合并。当得到最小子问题的解后,会沿着递归调用的逆序,将子问题的解逐步回代、合并,最终推导出原问题的完整解,这个过程称为递归的回溯。

下面按递归的思路分析 $n!$ 的计算过程。

阶乘的数学定义:终止条件为 $1! = 1$,递推关系为 $n! = n \times (n-1)!$ 。

具体计算过程如下。

(1) 计算 $n!$ 。根据递推关系,将原问题拆解为 $n! = n \times (n-1)!$,把求 $n!$ 减小到求 $(n-1)!$ 。

(2) 递归计算 $(n-1)!$ 。同理, $(n-1)! = (n-1) \times (n-2)!$,继续拆解为求 $(n-2)!$ 。

(3) 持续拆解知道触发终止条件,最后递归计算 $2!$, $2! = 2 \times 1!$,而 $1! = 1$ 是已知的,停止递归并返回 1。

(4) 回溯合并结果。将 $1! = 1$ 代入 $2!$ 的计算,得到 $2! = 2 \times 1 = 2$;再将 $2! = 2$ 代入 $3!$ 的计算,得到 $3! = 3 \times 2 = 6$,以此类推,逐步回代最终得到 $n!$ 的值。

【代码展示】 $n!$ 是个很大的数,需要考虑用什么数据类型来表示。注意到本题的 n 最大是 12, $12! = 479001600$,用 `int` 类型表示即可。如果 n 更大,就需要用 `long long` 或高精度类型了。

用递归函数 `fac()` 计算阶乘。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int fac(int n){
4      if(n==1) return 1;           //递归的最底层,终止条件
5      return n * fac(n-1);         //继续递归,并回溯
6  }
7  int main(){
8      int n;  cin >> n;
9      cout << fac(n);
10     return 0;
11 }
```

递归是非常基础的编程技术,在所有编程语言教材中,都作为必备的知识点出现。不过对刚学习编程的新手来说,递归往往是个难点。



题 1-43 B3860 pip install

题目描述: `pip install` 是 Python 安装包的重要指令。Python 的包具有依赖关系,在安装某些包之前,需要先安装这些包的依赖。假定共有依次编号为 $1 \sim N$ 的 N 个包,第 i 个包共有 K_i 个依赖。现在,小 F 想要安装编号为 1 的包。完成 1 号包及其全部依赖的安装,一共安装了多少个包?

输入格式: 输入共 $N+1$ 行。第一行为一个正整数 N 。接下来 N 行,第 i 行描述了编号为 i 的包的依赖:

- 第 i 行共有 $K_i + 1$ 个数;
- 第一个数为 K_i ;
- 接下来 K_i 个数,描述了所依赖的包的编号。

输出格式: 输出一个整数,表示安装的包的数量。

输入样例

```
5
1 2
1 3
1 4
0
0
```

输出样例

4

数据范围： $1 \leq N \leq 5000, 0 \leq K_i < N$, 保证不会构成循环依赖。

【考点】 递归, 记忆化。

【思路 and 证明】 用图 1-1 的例子说明本题的要求。有 7 个包,

1 依赖 2、3, 2、3 依赖 4, 4 依赖 5。要安装 1, 需要安装 2、3、4、5。

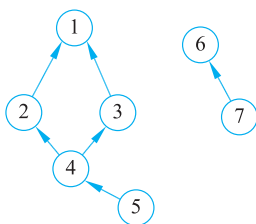


图 1-1 安装包依赖

如何找到 1 号包的所有依赖安装包? 首先找到 1 号包的直接依赖 2 号包和 3 号包, 然后接着找 2 号包和 3 号包的依赖包。这是一个递归过程, 从 1 开始递归, 递归到任何一个点后, 都继续往下递归, 直到最后一层, 递归终止。在递归回溯的过程中, 统计递归过的点, 累加这些点的数量。最后回溯到 1 时, 结束递归, 统计也结束, 得到答案。

【代码展示】

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int ans = 0;
4 int a[5005][5005]; //a[i][j]: i号包的第j个依赖
5 int install[5005];
6 void f(int k){
7     if(install[k] == 1) return; //k号包已经安装了,不用继续递归
8     for(int i = 1; i <= a[k][0]; i++) //递归k号包所依赖的软件包
9         f(a[k][i]);
10    install[k] = 1; //软件包递归结束,设置为需要安装
11    ans++;
12    return;
13 }
14 int main(){
15     int n;
16     scanf("%d", &n);
17     for(int i = 1; i <= n; i++){
18         scanf("%d", &a[i][0]); //a[i][0]记录i号包依赖的数量
19         for(int j = 1; j <= a[i][0]; j++){
20             scanf("%d", &a[i][j]);
21         }
22         f(i);
23     }
24     printf("%d", ans);
25 }
```

$f()$ 为递归函数。第 7 行是重要的递归优化技术, 称为记忆化。例如, 图 1-1 中, 从 1 先递归到 2、4、5, 此时 2、4、5 都统计了, 用 $install[] = 1$ 标记为已经安装。后面再从 1 递归到 3, 3 继续递归, 发现 4 已经标记过, 那么就不用再递归 4 了, 而且 4 后面的 5 也隐含不用递

归了。通过记忆化的优化,可以大大减少递归的计算量。如果去掉这一行,每个点会重复递归很多次,导致超时。

本题的输入量很大,所以用 `scanf()/printf()` 进行输入/输出。如果要用 `cin/cout`,需要加上优化语句“`ios::sync_with_stdio(false); cin.tie(0);`”。



题 1-44 P5461 赦免战俘

题目描述:借助反作弊系统,一些在月赛有抄袭作弊行为的选手被抓出来了!

现有 $2^n \times 2^n$ 名作弊者站成一个方阵等候 kkksc03 的发落。kkksc03 决定赦免一些作弊者。他将方阵均分为 4 个更小的方阵,每个更小的方阵的边长是原方阵的一半。其中左上角方阵的所有作弊者都将得到赦免,剩下 3 个小方阵中,每个方阵继续分为 4 个更小的方阵,然后通过同样的方式赦免作弊者,直到方阵无法再分下去为止。所有没有被赦免的作弊者都将被处以棕名处罚。

给出 n , 请输出每名作弊者的命运,其中 0 代表被赦免,1 代表不被赦免。

输入格式: 一个整数 n 。

输出格式: $2^n \times 2^n$ 的 0/1 方阵,代表每个人是否被赦免。数字之间有一个空格。

输入样例

3

输出样例

```
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 1
0 0 0 0 0 1 0 1
0 0 0 0 1 1 1 1
0 0 0 1 0 0 0 1
0 0 1 1 0 0 1 1
0 1 0 1 0 1 0 1
1 1 1 1 1 1 1 1
```

数据范围: $n \leq 10$ 。

【考点】 递归。

【思路 and 证明】 处理当前方阵时,把每个方阵分成 4 块:左上角标记为赦免,递归剩下 3 块。对每个方阵都做同样的递归操作,最后的方阵为 1×1 ,不能继续递归,直接返回。

【代码展示】

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 bool a[1030][1030];
4 void f(int x1, int y1, int x2, int y2){
5     if(x1 == x2 && y1 == y2) return;
6     int tx = (x1 + x2)/2, ty = (y1 + y2)/2;
7     for(int i = x1; i <= tx; i++)
8         for(int j = y1; j <= ty; j++)
9             a[i][j] = 0;
10    f(x1, ty + 1, tx, y2);
11    f(tx + 1, y1, x2, ty);
12    f(tx + 1, ty + 1, x2, y2);
13 }
```

//一行最多 $2^{10} = 1024$ 个
//左上角(x1,y1),右下角(x2,y2)
//递归的终点,返回
//计算中间点坐标
//左上角矩形被赦免
//递归左下角矩阵
//递归右上角矩阵
//递归右下角矩阵

```

14 int main(){
15     int n;    cin >> n;
16     n = 1 << n;                                //2~n
17     for(int i = 1; i <= n; i++)                //初始化,每人都不赦免
18         for(int j = 1; j <= n; j++)
19             a[i][j] = 1;
20     f(1,1,n,n);                                //开始是整个正方形:左上角(1,1),右下角(n,n)
21     for(int i = 1; i <= n; i++)                //输出结果矩阵
22         for(int j = 1; j <= n; j++) {
23             if(a[i][j] == 0)    cout << 0;
24             else                cout << 1;
25             if(j == n)          cout << "\n";    //换行
26             else                cout << ' ';
27         }
28
29     return 0;
30 }

```

1.3

进 制 转 换



题 1-45 B2143 进制转换

题目描述：用递归算法将一个十进制整数 X 转换成任意 M 进制数。

输入格式：一行两个数，第一个数为十进制整数 X ，第二个数为进制 M 。

输出格式：输出结果。

输入样例

431 16

输出样例

1AF

数据范围： $1 \leq X \leq 10^9$, $2 \leq M \leq 16$, M 为整数。

【考点】 进制转换、短除法、递归。

【思路 and 证明】 进制转换是计算机基础知识。

如何把十进制转换成其他进制？设十进制 X 的 M 进制是 $b_k \cdots b_3 b_2 b_1 b_0$ ，根据 M 进制的定义，有

$$X = b_k M^k + \cdots + b_3 M^3 + b_2 M^2 + b_1 M^1 + b_0 M^0$$

为了求十进制 X 的 M 进制 $b_k \cdots b_3 b_2 b_1 b_0$ ，进行以下步骤。

- (1) 用 X 除以 M ，商为 a_0 ，余数为 b_0 ；
 - (2) 继续用 a_0 除以 M ，商为 a_1 ，余数为 b_1 ；
 - (3) 重复步骤(2)直至商为 0，倒序输出余数 $b_k \cdots b_3 b_2 b_1 b_0$ ，即为 X 的 M 进制表示。
- 这种方法称为“短除法”。

下面以输入样例把十进制整数 431 转换为十六进制为例。十六进制有 16 个数码：0, 1, 2, ..., 9, A, ..., F。

第 1 步：431 除以 16，商为 26，余数为 F。

第 2 步：26 除以 16，商为 1，余数为 A。

第 3 步：1 除以 16，商为 0，余数为 1，结束。

答案是 1AF,即倒序输出的余数。计算过程表示为

$$\begin{array}{r}
 16 \overline{) 431} \\
 \underline{16 26} \quad \dots\dots F \\
 16 \overline{) 1} \quad \dots\dots A \\
 \underline{16 0} \quad \dots\dots 1
 \end{array}$$

【代码展示】 下面代码中第 9 行的 func() 函数给出了短除法的代码。由于要倒序输出余数,需要先用 s[] 存储余数,计算完后在第 17 行倒序输出。

倒序输出如果用递归来处理,编码非常简单,见第 4 行的 func() 函数。

如何倒序输出余数? 在递归函数 func() 中,先继续递归,递归返回后再输出,就倒序输出了余数。如果先输出再递归,就是正序输出,结果是错的。

初学者对递归的掌握可能有些困难,请一定要掌握好递归。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  char ch[] = "0123456789ABCDEF";      //存进制的数码,能转 2~16 进制
4  void func(int x, int m){               //必须满足 2≤m≤16,否则会出错
5      if(x/m) func(x/m,m);              //先递归:继续除
6      cout << ch[x%m];                  //再输出:倒序输出余数
7  }
8  /* 下面是非递归写法
9  void func(int x, int m){
10     string s;
11     int len = 0;
12     while(x!=0){
13         s[len] = ch[x%m];                //存余数
14         x /= m;
15         len++;
16     }
17     for(int i = len - 1; i >= 0; i--) cout << s[i];    //倒序输出余数
18 }
19 */
20 int main(){
21     int x, m;   cin >> x >> m;
22     func(x,m);    //短除法
23     return 0;
24 }
```

本题是把十进制转换为其他进制,如果要求在任意进制间互相转换,可以用十进制中转。例如,五进制转换为九进制,先把五进制转换为十进制,再把十进制转换为九进制。

下一题是把其他进制转换为十进制。



题 1-46 B3620 X 进制转换为十进制

题目描述: 给出一个小整数 x 和一个 x 进制的数 S ,将 S 转换为十进制数,对于超过十进制的数码,用 A,B,...表示。

输入格式: 第一行输入一个整数 x ; 第二行输入一个字符串 S 。

输出格式: 输出仅包含一个整数,表示答案。

输入样例 16 7B	输出样例 123
数据范围：保证目标数在十进制下不超过 $10^9, 1 \leq x \leq 36$ 。	

【考点】进制转换, pow() 函数的精度问题。

【思路 and 证明】如何把 M 进制转换成十进制？设 M 进制数是 $b_k \cdots b_3 b_2 b_1 b_0$, 转换成十进制数 X 。根据进制的定义, 有

$$X = b_k M^k + \cdots + b_3 M^3 + b_2 M^2 + b_1 M^1 + b_0 M^0$$

按上述公式计算即可得到 X 。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  int main() {
4      int m;    cin >> m;                //输入进制
5      string s; cin >> s;                //输入数字 s, 用字符串形式输入
6      int ans = 0;                        //答案
7      int len = s.length();              //数字的长度
8      for(int i = len - 1; i >= 0; i--) { //从高位往低位处理数字, 和第 15 行配合
9          char ch = s[len - 1 - i];
10         int num;                        //把这个字符转换成整数
11         if(ch >= '0' && ch <= '9') num = ch - '0';
12         if(ch >= 'A' && ch <= 'Z') num = ch - 'A' + 10;
13         if(ch >= 'a' && ch <= 'z') num = ch - 'a' + 10;
14         //ans += num * pow(m, i);        //累加对应的十进制值。用 pow() 函数可能有误差
15         ans = ans * m + num;            //这样不会有误差, 建议使用
16     }
17     cout << ans;
18     return 0;
19 }
```

第 9 行提取出 M 进制数的每位, 转换成数字 num , 就是公式中的 b_i , 然后在第 14 行乘以 M^i , 并累加得到十进制数 X 。

第 14 行用到库函数 $\text{pow}(m, i)$, 本题也是对的。不过要注意, 当 i 较大时, $\text{pow}()$ 函数有精度误差, 如 3^{34} 和 $\text{power}(3, 34)$ 不相等。

改成第 15 行的写法, 就没有问题。但是要注意, 第 8 行应该从高位向低位处理数字, 这样第 15 行的累乘 m , 才能使高位得到更大的幂次。

参考下一题“P10510 进制”的类似做法。



题 1-47 P10510 进制

题目描述：小洛正在学习三进制。他定义三进制数为每位只包括 0、1、2 的无穷长的数字串 a 。与一般的三进制数不同的是, 小洛的三进制数从左往右书写。例如, 在常规表述下, 4 的三进制表示为 $(0000\ 0011)_3$, 而在小洛的三进制数表示为 $(1100\ 0000)_3$ 。

小洛特别喜欢从 0 开始计数, 因此他规定三进制的第 0 位为最左侧的数位。

下面给出了小洛的三进制数表示下, 数字 37 的最低 8 位数位以及位权:

三进制数表示	1	0	1	1	0	0	0	0
位数	0	1	2	3	4	5	6	7
位权	3^0	3^1	3^2	3^3	3^4	3^5	3^6	3^7

现在小洛有一个十进制正整数 V (显然,需要将其转换为小洛的三进制数表示),并且给定 3 种操作。

操作 1,对第 i 位上的数进行操作: 0 变为 1,1 变为 2,2 变为 0。

操作 2,对第 i 位上的数进行操作: 0 变为 2,1 变为 0,2 变为 1。

操作 3,对第 i 位上的数进行操作: 1 变为 2,2 变为 1,0 不变。

小洛一共要进行 q 次操作。每次操作后,需要得到三进制串代表的数值,请你告诉他。

如果题意描述有不清晰之处,请查看样例解释。

输入格式: 第一行输入两个正整数 V, q 。接下来 q 行,每行一个操作,形如 $op\ i$ ($op=1,2,3$)。

请注意,你需要将 V 转换为三进制串并作为初始的三进制串。

输出格式: 输出共 q 行,第 i 行表示第 i 个操作后的答案。

输入样例

4 3

1 1

2 0

1 2

输出样例

7

6

15

数据范围: $0 \leq V \leq 10^{18}, 1 \leq q \leq 10^5$, 任意获取到的答案不超过 2×10^{18} 。

【考点】 进制转换, $\text{pow}()$ 函数的精度。

【思路 and 证明】 模拟题目的要求,先将十进制转换为三进制,然后执行每个操作,最后将三进制转换为十进制。

【代码展示】

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  ll s[50];           //18 位十进制数(<264)转成三进制不会超过 50 位
5  ll power[50];       //预计算 3 的倍数
6  int main(){
7      ll V;  int q;
8      cin>> V>> q;
9      power[0] = 1;
10     for(int i = 1; i <= 50; ++i)  power[i] = power[i-1] * 3;    //预计算 3 的倍数
11     ll ans = 0;
12     int len = 0;           //三进制数的位数
13     while(V) {             //十进制转三进制
14         s[len++] = V%3;
15         V/= 3;
16     }
17     while(q-- ) {
18         ans = 0;
19         int op,x;  cin>> op>> x;

```

```
20     if(op == 1) s[x] = (s[x] + 1)%3;
21     if(op == 2) s[x] = (s[x] + 2)%3;
22     if(op == 3) s[x] = (3 - s[x])%3;
23     for(int i = 49; i >= 0; i--) //三进制转十进制。注意本题的 i 可以大于 len
24         ans = ans * 3 + s[i]; //从高位往低位处理数字,得到 3 的幂
25         //ans += s[i] * power[i]; //这样写也行,用到第 9、10 行预计算的 power[]
26         //ans += s[i] * pow(3, i); //pow()有误差
27     cout << ans << '\n';
28 }
29 return 0;
30 }
```

十进制与三进制转换仍然用短除法。题 1-45“B2143 进制转换”用递归函数求进制转换的余数的倒序输出。不过,本题是低位在前,高位在后,余数是正序的,不需要用递归。第 13 行的 while 循环是十进制转三进制。

三进制转十进制,见第 24 行。第 25 行的写法也是对的,用到了预计算出的 `power[]`。第 26 行用库函数 `pow()`,它有精度误差,导致错误。