异步图书
www.epubit.com

巷积传媒
AIRBOOK

适合零基础读者入门

编程竞赛宝典

C++语言和算法入门



张新华 编著

- 循序渐进掌握编程竞赛知识点
- 配有视频讲解，完整展示解题过程
- 配套代码经过多次校验，更有丰富测试数据

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

目录

版权信息

版 权

内容提要

本书编委会

前言

第01章 C++语言入门

- 1.1 我的第一个程序
  - 1.1.1 并排的树（tree）
- 1.2 数据类型及运算
  - 1.2.1 求两个整数的积和平均值（calc）
  - 1.2.2 3个浮点数相加（add2）
  - 1.2.3 捡石头（stone）
  - 1.2.4 简单解方程（equation）
  - 1.2.5 计算多项式的值（cal）
  - 1.2.6 混合运算（cal2）
  - 1.2.7 两个整数相乘（mul）
  - 1.2.8 巨型人造天体（aster）
- 1.3 字符和字符串

1.3.1 字符金字塔（Pyramid）
1.3.2 ASCII值转字符（ASCII1）
1.3.3 小写字母转大写字母
1.3.4 大写字母转小写字母（ASCII2）
1.3.5 恺撒加密术
1.3.6 改进的加密术（encryption）
1.3.7 输出特殊字符（special）
1.3.8 费解的对话（chat）
1.4 输入/输出及格式控制
1.4.1 精确到小数点（fixed）
1.5 一些运算规则
第02章 基本结构
2.1 顺序结构
2.1.1 三位数分解（check）
2.1.2 反向输出四位数（reverse）
2.1.3 小数的四舍五入（rounding）
2.1.4 求等差数列末项（series）
2.1.5 徽章（badge）
2.1.6 计算梯形的面积（trapezoid）
2.1.7 二进制存储（bin）
2.1.8 校庆日（day）
2.1.9 计算星期几（week）
2.1.10 求余数（remainder）
2.1.11 求二次方根（sqrt）
2.1.12 勾股定理（triangle）

2.1.13 计算两点间的直线距离（distance）
2.1.14 海伦公式（helen）
2.1.15 计算三角形的面积（helen2）
2.1.16 计算练习时间（training）
2.2 选择结构
2.2.1 求绝对值(abs)
2.2.2 奇偶数判断(odd)
2.2.3 两个实数排序(sort)
2.2.4 3个实数排序(sort3)
2.2.5 判断两数的大小(compare)
2.2.6 等级制(grade)
2.2.7 翻转硬币(coin)
2.2.8 邮寄包裹(post)
2.2.9 选班级代表(election)
2.2.10 角色识别(role)
2.2.11 判断闰年(leap)
2.2.12 正方形内的点(point)
2.2.13 判断数的整除(div)
2.2.14 五分制(five)
2.2.15 等级划分(gradation)
2.2.16 简单计算器(cal)
2.2.17 运费计算(freight)
2.2.18 某年某月有几天(day)
2.2.19 判断某日是一年的第几天(day)
2.3 循环结构
2.3.1 计算n的阶乘(factorial)

2.3.2 电文保密(message)
2.3.3 求数的总和(power)
2.3.4 整数猜想(guess)
2.3.5 数字反转（rererse）
2.3.6 级数求和（sum）
2.3.7 投资收益(investment)
2.3.8 找规律1（array）
2.3.9 找规律2(array)
2.3.10 求圆周率(pi)
2.3.11 求最大公约数和最小公倍数(euclid)
2.3.12 求Sn的值(sn)
2.3.13 找规律3(array)
2.3.14 计算皮球的弹跳高度(ball)
2.3.15 显示ASCII码值对应的字符(ASCII)
2.3.16 兔子“永生”（rabbit）
2.3.17 计算平均分(average)
2.3.18 计算数列和(drug)
2.3.19 奇数求和(odd)
2.3.20 求水仙花数(flower)
2.3.21 求最大跨度值(Max)
2.3.22 求e的近似值(e)
2.3.23 求分子序列和(sum)
2.3.24 计算圆面积(square)
2.3.25 不能被3整除的数(sacred)
2.3.26 火柴游戏
2.3.27 九九乘法表1
2.3.28 九九乘法表2(99)

2.3.29 执行任务
2.3.30 谁是小偷(thief)
2.3.31 绘制矩形（draw）
2.3.32 不定方程的解(equation)
2.3.33 1的次数(sum1)
2.3.34 算式成立1（formula）
2.3.35 算式成立2（formula2）
2.3.36 算式成立3（formula3）
2.3.37 同构数(num)
2.3.38 判断质数（prime）
2.3.39 质因数分解(factor)
2.3.40 完美数(perfect)
2.3.41 防护罩(safe)
2.3.42 换零钱1（money）
2.3.43 换零钱2(money)
2.3.44 埃及分数(Egypt)
2.3.45 除式还原(div1)

第03章 竞赛模拟

3.1 文件读写
3.1.1 求中间数（mid）
3.1.2 修改作文（word）
3.2 制作测试数据
3.3 随机数据与对拍
3.3.1 模拟骰子
3.4 效率分析及简单优化

3.4.1 排行榜（list）

第04章 数组

4.1 一维数组

4.1.1 上楼梯（stairs）

4.1.2 逆序输出数组元素（reverse）

4.1.3 数组元素前移（move）

4.1.4 分糖（candy）

4.1.5 开关灯1（light1）

4.1.6 开关灯2（light2）

4.1.7 放花炮（fire）

4.1.8 冒泡排序法（sort）

4.1.9 车厢重组（train）

4.1.10 统计各数据的个数（number）

4.1.11 狼抓兔子（mystery）

4.1.12 求分数精确值1（exact）

4.1.13 求分数精确值2（exact）

4.1.14 复杂除式还原（div2）

4.1.15 求质数（prime）

4.2 二维数组

4.2.1 最强魔法师（powerful）

4.2.2 矩阵加法（matrix1）

4.2.3 杨辉三角（YangHui）

4.2.4 矩阵转置（matrix3）

4.2.5 扫雷游戏（game）

4.2.6 矩阵乘法（matrix2）

4.2.7 神奇矩阵
4.2.8 蛇形矩阵1（snake1）
4.2.9 回型方阵（matrix）
4.2.10 蛇形矩阵2（snake）
4.2.11 蛇形矩阵3（snake）
4.3 字符数组
4.3.1 求逆序字符串（reverse）
4.3.2 求数的和（sumN）
4.3.3 查找子串（findchar）
4.3.4 字符串游戏（string）
4.3.5 柱状图（chart）
4.3.6 字符串排序（charsort）
4.3.7 最大整数（BigNum）
4.3.8 统计单词数（stat）
4.3.9 子串包含问题（substr）
4.3.10 命名（namenum）
4.3.11 古风排版（typesetting）
4.4 滚动数组
4.4.1 截铁丝（line）
4.4.2 太阳能电池（battery）
第05章 阶段检测1
5.1 笔试检测
5.2 上机检测
5.2.1 猴子吃桃（peach）
5.2.2 卖宝石（stone）

5.2.3 取石子游戏（stone）
5.2.4 打电话（telephone）
5.2.5 神秘字符串（ride）
5.2.6 合法C++标识符（c）
5.2.7 求立方根（cuberoot）
5.2.8 插入排序（insert）
5.2.9 排名次（billing）
5.2.10 ISBN（isbn）
5.2.11 救援顺序（rescue）
5.2.12 比例简化（ratio）
5.2.13 黑色星期五（friday）
5.2.14 序列变换（change）
5.2.15 打猎（hunt）
5.2.16 矩阵排序（sort）
5.2.17 “语言之争”（language）
5.3 头脑风暴
5.3.1 五位数
5.3.2 六位数
5.3.3 计算S的值（tower）
5.3.4 求0的个数（factorial）
5.3.5 多项式求值（poly）
5.3.6 二分法求方程的解（equation）
5.3.7 求最大子序列和（sum）
5.3.8 猴子选大王（monkey）
5.3.9 NOI（NOI）
5.3.10 NOI字符串（NOI）

5.3.11 组合数问题（problem）

第06章 函数

6.1 初识函数

6.1.1 浮点数求最大值（Max）

6.1.2 判断平方数（square）

6.1.3 哥德巴赫猜想（Goldbach）

6.1.4 约分（comm）

6.1.5 素数回文数（Prime4）

6.1.6 丑数（ugly）

6.1.7 三质数（prime5）

6.2 库函数简介

6.3 常用的变量类型

6.4 数组作为函数参数

6.5 函数的递归调用

6.5.1 逆序字符（reverse）

6.5.2 逆序数（reverse）

6.5.3 求阶乘（factorial）

6.5.4 最大公约数和最小公倍数

6.5.5 复杂算式（f）

6.5.6 母牛数（cow）

6.5.7 矩阵行走（walk）

6.5.8 计算函数值（f）

6.5.9 汉诺塔1（hanoi1）

6.5.10 汉诺塔2（hanoi2）

第07章 阶段检测2
7.1 笔试检测
7.2 上机检测
7.2.1 RSA加密算法（RSA）
7.2.2 单词排序（word）
7.2.3 回文日期（date）
7.2.4 多项式输出（poly）
7.2.5 最大公约数和最小公倍数问题（b）
7.2.6 子矩阵求和（matrix）
7.2.7 石头剪刀布（rps）
7.2.8 公约数最大（com）
7.2.9 无线网络发射器选址（wireless）
7.2.10 机器翻译（translate）
7.2.11 玩具谜题（toy）
7.2.12 cantor表（cantor）
7.2.13 计算器的改良（calc）
7.2.14 维吉尼亚密码（vigenere）
7.2.15 计算系数（factor）
7.2.16 大炮互攻（cannon）
7.2.17 解方程（equation）
7.2.18 shlqsh数（shlqsh）
第08章 指针
8.1 地址和指针
8.2 指针变量的应用

8.3 数组与指针

第09章 结构体

9.1 结构体及其应用

9.2 结构体与指针

9.3 课后练习

9.3.1 选举（kind）

9.3.2 统计成绩（score）

9.3.3 生日（birthday）

第10章 位运算与进制

10.1 位运算

10.1.1 整数幂（power）

10.1.2 二进制半整数（bin）

10.2 进制转换

10.2.1 N进制数加法（add）

10.2.2 二进制数分类（classify）

10.2.3 确定进制（num）

10.2.4 K进制数转L进制数

第11章 STL编程

11.1 sort排序算法

11.1.1 单词排序(WordSort)

11.1.2 志愿者选拔（voluntary）

11.1.3 奖学金（scholar）

11.1.4 导弹拦截（missile）
11.2 lower bound/upper bound
11.3 vector向量容器
11.3.1 电话列表(telephone)
11.3.2 普通平衡树（BalanceTree）
11.4 pair容器
11.4.1 奇怪的排序（OddSort）
11.5 set集合容器
11.5.1 问卷调查（random）
11.5.2 两倍（Double）
11.6 multiset多重集合容器
11.6.1 12!配对(12)
11.6.2 01串排序(Sort01)
11.6.3 卡片游戏（CardGame）
11.7 deque双端队列容器
11.7.1 鸡蛋队列(egg)
11.8 list双向链表容器
11.8.1 队列训练（train）
11.9 map映照容器
11.9.1 射箭（toxophily）
11.9.2 漂亮数字(pretty)
11.9.3 彩色石头(stone)
11.9.4 水果（fruit）
11.9.5 A－B(sub)

11.10 排列组合关系算法
11.10.1 进制位（system）
11.11 stable sort 稳定排序
11.11.1 给朋友排序（friends）
11.12 multimap多重映照容器
11.12.1 银行业务(bank)
11.13 stack堆栈容器
11.13.1 字符栈（CharStack）
11.13.2 表达式括号匹配1（bracket1）
11.13.3 表达式括号匹配2（bracket2）
11.13.4 表达式求值（expr）
11.14 queue队列容器
11.14.1 海港（port）
11.14.2 团体队列（queue）
11.15 priority queue优先队列容器
11.15.1 推销员（salesman）
11.15.2 有序表最小和（MinSum）

附录A ASCII 对照表

附录B C++语言的关键字

附录C C++语言运算符及其优先级

附录D 常用函数库

附录E 常用函数

附录F 如何使用在线评测网站

版权信息

书名：编程竞赛宝典：C++语言和算法入门

ISBN：978-7-115-55461-1

本书由人民邮电出版社发行数字版。版权所有，侵权必究。

您购买的人民邮电出版社电子书仅供您个人使用，未经授权，不得以任何方式复制和传播本书内容。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

版　权

编　　著　张新华

责任编辑　赵祥妮

人民邮电出版社出版发行　　北京市丰台区成寿寺路11号

邮编　100164　　**电子邮件**　315@ptpress.com.cn

网址　http://www.ptpress.com.cn

读者服务热线：(010)81055410

反盗版热线：(010)81055315

内容提要

编程类竞赛活动受各级各类学校重视，受青少年学生欢迎。本书以Dev-C++为C++语言的开发环境，首先带领读者入门C++语言，然后循序渐进、由浅入深地讲解C++语言的基本结构、数组、函数、指针、结构体、位运算等知识，并编排了竞赛模拟、阶段检测等内容，使读者能及时评估自己的学习效果。

本书在介绍C++语言的同时，更加侧重于计算思维的培养，通过“一题多解”及“数学求解”等方法，拓展读者对题目的本质和内涵的思考与理解。本书还配备了参考程序、习题解答、测试数据、讲解视频等资源供读者参考学习。

本书由具有丰富经验的编程竞赛教练编写，适合作为全国青少年编程竞赛及大学生程序设计竞赛的培训用书，也可作为计算机专业学生、算法爱好者的参考和学习用书。

本书编委会

主任： 严开明　　胡向荣

成员： 张义丰　　李守志

陈　科　　王　震

谢春玫　　梁靖韵

葛　阳　　徐景全

刘路定　　黎旭明

张就序　　郭继飞

伍婉秋　　热则古丽

前　　言

编程竞赛介绍

随着计算机逐步深入人类生活的各个方面，利用计算机及其程序设计来分析、解决问题的算法在计算机科学领域乃至整个科学界的作用日益明显。相应地，各类以算法为主的编程竞赛也层出不穷：在国内，有全国青少年信息学奥林匹克联赛（National Olympiad in Informatics in Provinces，NOIP），该联赛与全国中学生生物学联赛、全国中学生物理竞赛、全国高中数学联赛、全国高中学生化学竞赛并称为国内影响力最大的“五大奥赛”；在国际上，有面向中学生的国际信息学奥林匹克竞赛（International Olympiad in Informatics， IOI）、面向亚太地区在校中学生的信息学科竞赛即亚洲与太平洋地区信息学奥林匹克（Asia-Pacific Informatics Olympiad，APIO）以及由国际计算机 学会主办的面向大学生的国际大学生程序设计竞赛（International Collegiate Programming Contest，ICPC）等。

各类编程竞赛要求参赛选手不仅具有深厚的计算机算法功底、快速并准确编程的能力以及创造性的思维，而且有团队合作精神和抗压能力，因此编程竞赛在高校、IT公司和其他社会各界中获得越来越多的认同和重视。编程竞赛的优胜者更是微软、谷歌、百度、Facebook

等全球知名IT公司争相高薪招募的对象。因此，除了各类参加编程竞赛的选手外，很多不参加此类竞赛的研究工作者和从事IT行业的人士，也都希望能获得这方面的专业训练并从中得到一定的收获。

为什么要学习算法

经常有人说：“我不学算法也照样可以编程开发软件。”那么，为什么还要学习算法呢？

首先，算法（Algorithm）一词源于算术（Algorism），具体地说，算法是一个由已知推求未知的运算过程。后来，人们把它推广到一般过程，即把进行某一工作的方法和步骤称为算法。一个程序要完成一个任务，其背后大多会涉及算法的实现，算法的优劣直接决定了程序的优劣。因此，算法是程序的“灵魂”。学好了算法，就能够设计出更加优异的软件，以非常有效的方式实现复杂的功能。例如，设计完成一个具有较强人工智能的人机对弈棋类游戏，程序员没有深厚的算法功底是根本不可能实现的。

其次，算法是对事物本质的数学抽象，是初等数学、高等数学、线性代数、计算几何、离散数学、概率论、数理统计和计算方法等知识的具体运用。真正懂计算机的人（不是“编程匠”）都在数学上有相当高的造诣，既能用科学家的严谨思维来求证，也能用工程师的务实手段来解决问题——这种思维和手段的最佳演绎之一就是“算法”。学习算法，能锻炼我们的思维，使思维变得更清晰、更有逻辑、更有深度和广度。学习算法更是培养逻辑推理能力的非常好的方

法之一。因此，学会算法思想，其意义不仅在于算法本身，更重要的是，对以后的学习生活和思维方式也会产生深远的影响。

最后，学习算法本身很有意思、很有趣味。所谓“技术做到极致就是艺术”，当一个人真正沉浸到算法研究中时，他会感受到精妙绝伦的算法的艺术之美，也会为它的惊人的运行速度和构思而深深震撼，并从中体会到一种不可言喻的美感和愉悦。当然，算法的那份“优雅”与“精巧”虽然吸引人，却也令很多人望而生畏。事实证明，对很多人来说，学习算法是一件非常有难度的事情。

本书的特色

本书尽可能详尽而全面地介绍编程竞赛中用到的大多数知识，读者如果能按照书中的安排，认真做好每一道题，必然能在各类编程竞赛中一展身手。

本书中部分题目采用了“多向思考”“一题多解”“一题多变”的解决方法，其目的主要有3点：一是充分调动读者思维的积极性，提高读者综合运用已学知识解答问题的技能；二是锻炼读者思维的灵活性，促进读者知识和智慧的增长；三是增加读者思维的深度和广度，引导读者灵活地掌握知识的纵横联系，培养和发挥读者的创造性。因此，绝不能简单地将这种训练看作编程技巧的花哨卖弄，相反，它能培养读者思维的敏捷性，促进读者智力和思维的发展，提高读者的变通能力与综合运用所学知识的能力。读者若能坚持这种思维训练方法，必能获得意想不到的良好学习效果。

如何使用本书

本书采用的是循序渐进、由浅入深的教学方法。一开始讲解引入新知识点的题目时，书中会提供该题目的完整参考代码以供读者参考，但随着读者对此知识点的理解逐步加深，后续的同类型题目将逐步向仅提供算法思路、提供伪代码和无任何提示的方式转变。此外，对于一些思维跨度较大的题目，本书会酌情给予读者一定的提示。

本书在“第02章 基本结构”后直接安排了“第03章 竞赛模拟”，是为了使读者尽快熟悉竞赛环境，做到在平时的训练中以竞赛的标准要求自己，例如，制作全面而完善的测试数据验证程序的正确性、确保提交的程序一次通过而不是反复提交才通过等。但这种思维习惯的养成，非一朝一夕之功，所以读者无须循规蹈矩，只需大致了解第03章的内容后就可以继续后面章节的学习，因为第03章的内容是贯穿于整本书的学习过程，并在不断做题实践中加深理解的。

第08章是关于指针的内容，通常初学者在竞赛中应用极少，读者可以根据自身情况确定是否学习。

题目后括号内的英文为提交的源代码名。

资源与支持

为了让读者有更好的使用体验，本书提供了多种资源与支持的方式。

（1）增值讲解视频

- 扫描书上二维码即可在线观看视频。
- “内容市场”微信小程序或App中搜索本书书名，即可在线观看视频。
- 异步社区网站搜索本书书名，单击在线课程，即可在线观看视频。

（2）测试数据

登录本书作者开发维护的在线评测网站www.magicoj.com，单击【课程分类】-【语言和算法入门】，选择对应题目即可查看，还可解锁海量题库并在线提交代码获得反馈。

（3）例题完整参考代码

在异步社区网站搜索本书书名，单击下载资源，即可下载例题完整参考代码。

适合阅读本书的读者

本书可作为NOIP的复赛教程和ICPC的参考和学习用书，还可作为计算机专业学生、算法爱好者的参考和学习用书。

致谢

感谢安徽省安庆市石化第一中学胡向荣老师，感谢浙江省宁波市北仑区霞浦学校张义丰老师，感谢浙江省慈溪市周巷镇潭北小学陈科老师，感谢贵州省贵阳市第一中学李守志老师，感谢全国各省市中学、大学的信息学奥赛指导老师，他们给本书提了许多真诚而有益的建议，并对笔者在写书过程中遇到的一些困惑和问题给予了热心的解答。

本书在完成过程中，使用了NOIP的部分原题、在线评测网站的部分题目，并参考和收集了其他创作者发表在互联网、杂志等媒体上的相关资料，无法一一列举，在此一并表示衷心感谢。

感谢卷积文化传媒（北京）有限公司CEO高博先生和他的同事，他们成功地将现代AI技术与传统的图书完美地结合起来，降低了读者学习编程的门槛。

最后要说的话

由于笔者水平所限，书中难免存在不妥之处，欢迎同仁或读者赐正。读者如果在阅读中发现任何问题，请发送电子邮件到hiapollo@sohu.com，更希望读者对本书提出建设性意见，以便修订再版时改进。

本书对应的题库网站正在不断完善中，网址为www.magicoj.com。

希望本书的出版，能够给学有余力的中学生、计算机专业的大学生、程序算法爱好者以及IT行业从业者提供学习计算机科学的帮助。

张新华

2021年5月

第01章　C++语言入门

计算机编程语言能够实现人与计算机之间的沟通与交流，使计算机能够根据人编写的代码一步一步地工作，完成某些特定的任务。C++语言是使用最广泛的编程语言之一，也是许多编程竞赛指定的编程语言之一。

1.1　我的第一个程序



扫码看视频

我们使用Dev-C++这个适合初学者使用的C/C++集成开发环境来学习C++语言，读者可上网搜索“Dev C++ 下载”等关键词从网上下载该软件，或者使用浏览器访问www.magicoj.com（或者www.razxhoi.com），下载它的改进版本——Dev-CPP智能开发平台，如图1.1所示。

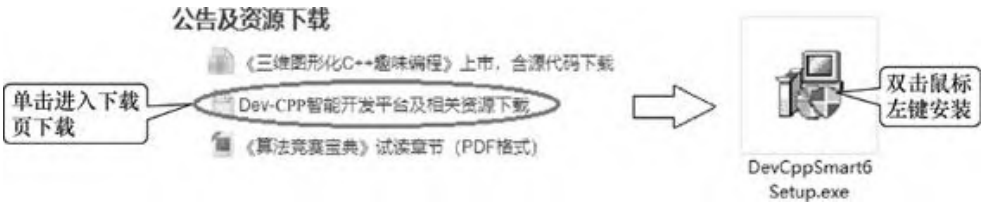


图1.1

安装界面如图1.2（a）所示。单击其中的“下一步”按钮，出现选择目标位置的界面，建议继续单击“下一步”按钮默认安装，如图1.2（b）所示。



（a）　　（b）

图1.2

安装完成后运行软件，将出现初始设置界面。

如果没有出现初始设置界面或者弹出错误对话框（因为未正确卸载旧版Dev-C++）等，可在软件的“工具”菜单里选择“环境选项”，在“文件和路径”选项卡中单击“删除设置并退出”按钮，再重新运行软件即可恢复正常，如图1.3所示。



图1.3

设置好的软件界面（Dev-CPP智能开发平台支持换肤功能）如图1.4所示。



图1.4

存放代码的文件称为源代码文件，编写代码之前要新建一个源代码文件，图1.5所示为新建源代码文件的两种方法。



图1.5

如图1.6所示，由模板自动生成的代码（若使用的非Dev-CPP智能开发平台则用户需手动输入）显示在代码编辑区。

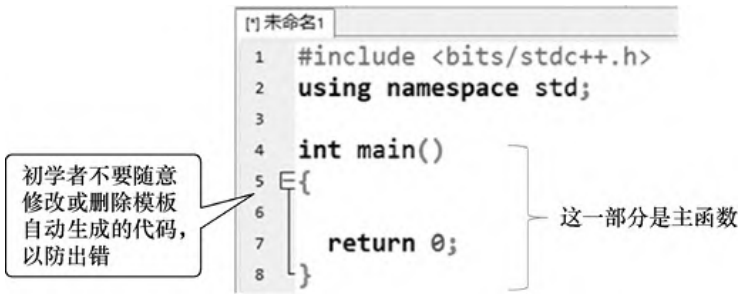


图1.6



如果不想要自动生成代码，想自己手动输入全部代码，只要在“工具”→“环境选项”中将新建文件时选中的“自动加载模板”选项取消即可。

#include <bits/stdc++.h> 表示编写的代码要包含子目录bits下的一个名为“stdc++.h”的头文件。之所以叫作头文件，是因为这类文件一般放在代码的开头。C++有许多头文件，它们可以实现一些特定的功能。stdc++.h头文件又称“万能”头文件，因为代码中只要包含该头文件就基本无须再包含其他头文件了。

using namespace std;表示使用的命名空间为“std”（std是英文单词“standard”即“标准”的缩写），这主要是为了解决名字冲突的问题，初学者暂不必深究。

“int”是英文单词“integer”即“整数”的缩写，和第7行的return 0相呼应。“return”的中文含义是“返回”，而0是整数，意思是给系统（调用者）返回整数0。0表示一切运行正常，即“没有消息就是好消息”，显然代码能运行到return 0，说明代码是正常结束而不是运行半途出错、异常退出的。

“main”的中文含义是“主要的”，C++程序里，必须有且只能有一个名为“main”的函数（主函数）。函数是指能完成一定功能的程序块，函数后面有圆括号，随后函数体用花括号标注，即第5～第8行。

C++程序的运行总是由main()函数的函数体里的第一条语句开始，到main()函数的函数体的最后一条语句结束。

现在编写代码，使程序运行后显示一行字符“Hello,world”，这需要在代码编辑区第6行空白处加入cout<<"Hello,world\n"；，如图1.7所示。

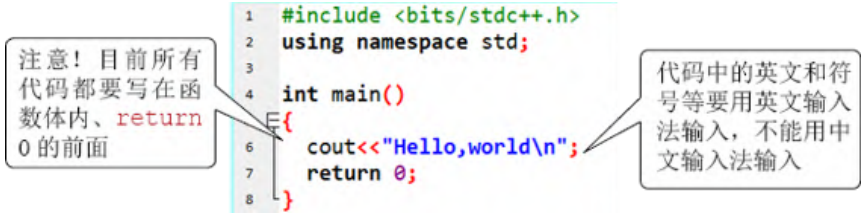


图1.7

cout用于输出紧随操作符“<<”后的双引号中的字符串，输出的字符串必须包含在双引号之中，其中“\n”表示换行。



C++语言对字母大小写敏感，例如字母“a”和“A”不同。

除输出内容有中文，需要用中文输入法在双引号中输入中文字符外，其他字符都必须用英文输入法输入，否则代码可能无法运行。

C++语言中，每条语句末尾应以分号（;）表示结束，但预处理、函数头及花括号之后一般不加分号。

写好的代码要及时保存，保存的文件的扩展名为.cpp，如图1.8所示。

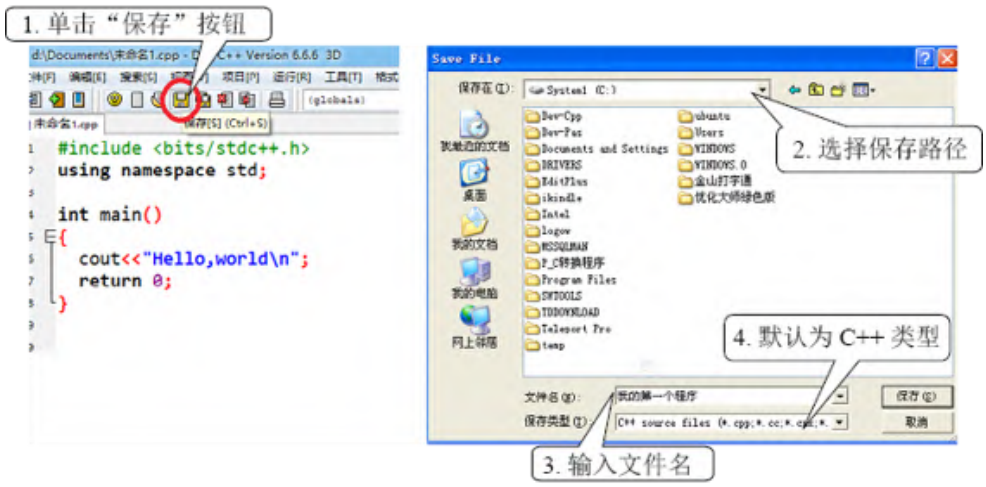


图1.8

以.cpp为扩展名保存的文件叫作源文件，源文件不能直接运行，要编译成可执行的EXE文件后方可运行。如图1.9所示，打开“运行”菜单后单击“编译”，无论编译成功还是失败，都会在软件界面下方功能区的信息栏输出相应的信息。

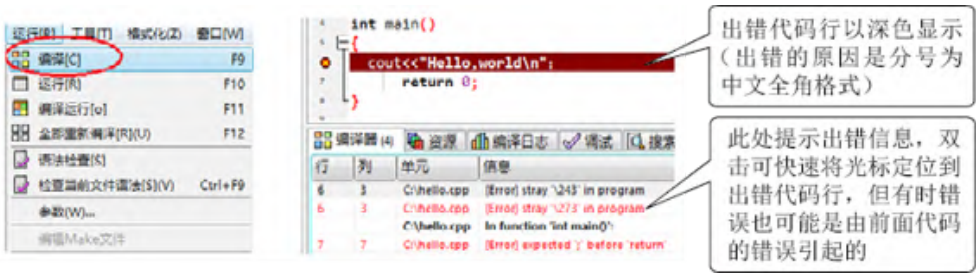


图1.9

编译成功后，打开“运行”菜单单击“运行”，将出现如图1.10所示的命令提示符窗口，显示输出的文字，第一个C++程序编写成功了！



图1.10

如图1.11所示，程序编译生成的EXE文件可独立运行而无须编译器的环境支持。



图1.11

使用cout可以输出各类字符，例如，输出多行中文字符串的代码如下。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  cout<<"你好，C++语言\n\n";
  cout<<"我是初学者\n";
  cout<<"我喜欢编程\n";
  return 0;
}
```

每一行输出字符串的末尾均有“\n”表示换行，输出结果如图1.12所示。

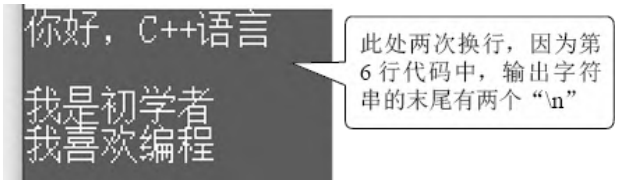


图1.12

1.1.1　并排的树（tree）

【题目描述】

请尝试编程输出如下的两棵“并排的树”。

```
   *          *
  ***        ***
 *****      *****
*******    *******
   *          *
   *          *
```

1.2　数据类型及运算



扫码看视频

编程需要用到各种类型的数据，例如整数、浮点数（包含小数）、字符串等。Dev-C++是Windows平台下的编译器，常用数据类型的字节长度和取值范围如表1.1所示。

表1.1

数据类型		字节长度	取值范围
整数类型	short（短整型）	2（16位）	-32 768～32 767
	int（整型）	4（32位）	-2 147 483 648～2 147 483 647
	long（长整型）	4（32位）	-2 147 483 648～2 147 483 647
	long long（超长整型）	8（64位）	-2^{63}～$2^{63}-1$
布尔类型	bool	1（8位）	（真）true或（假）false
字符类型	char	1（8位）	-128～127
浮点类型	float（单精度浮点型）	4（32位）	-3.4E-38～3.4E+38，有效位6～7位
	double（双精度浮点型）	8（64位）	-1.7E+308～1.7E+308，有效位15～16位
	long double （长精度浮点型）	16（128位）	-3.4E+4 932～1.1E+4 932，有效位18～19位

浮点数是表示小数的一种方法，简单理解就是小数点可以任意浮动的数字，例如3.14可以写成3.14×10^{0}、31.4×10^{-1}、314×10^{-2}等多种表现形式。而3.4×10^{38}这种表示法也可以用科学记数法表示为3.4E+38，例如2E-2即2×10^{-2}。

现代数码产品使用二进制数（0和1）来存储各类数据。例如，有一张存储视频的CD，图1.13（a）所示的螺旋形轨道表示存储的视频数据，图1.13（b）所示是在显微镜下显示出来的CD表面一个个的“坑”。我们将有坑的地方表示为1，没有坑的地方表示为0，这就是CD用二进制数存储数据的原理。

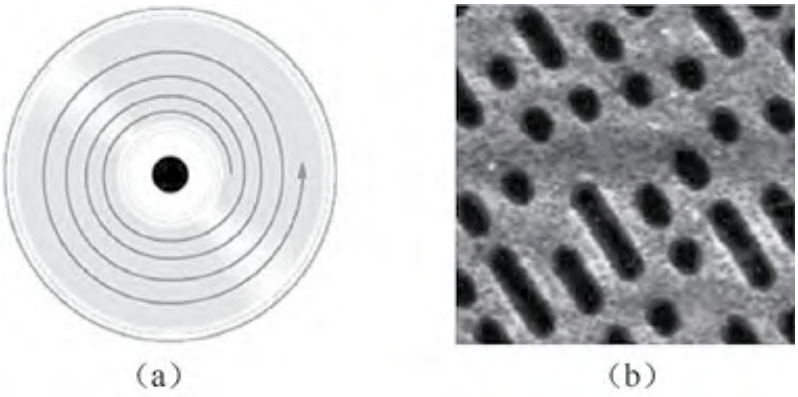


图1.13

位（bit）是计算机存储信息的最小单位，代表1个二进制位，其值为0或1，可以表示2种状态/2个数值；2个二进制位可以表示4个数值，即00、01、10、11；n个二进制位可以表示2^n个数值。例如，短整型的字节长度为16位，可以表示2^{16}即65 536个数值，考虑到正数、负数及0的表示，所以取值范围为-32 768～32 767。



8个二进制位为1字节(Byte)；1 024字节为1KB；1 024KB为1MB；1 024MB为1GB；1 024
GB为1TB ……

不同的编译器支持的各数据类型的字节长度可能不同，sizeof()可用于获取各数据类型的字节长度，例如输出各数据类型的字节长度的代码如下所示。

```
cout<<"int的字节长度为"<<sizeof(int)<<"\n";
cout<<"short的字节长度为"<<sizeof(short)<<"\n";
cout<<"long long的字节长度为"<<sizeof(long long)<<"\n";
```

C++程序用到的算术运算符如表1.2所示。

表1.2

算术运算符	含义	说明
+	加/正号	两数相加，例如2+3的值为5
-	减/负号	两数相减，例如2-3的值为-1
*	乘号	两数相乘，例如2*3的值为6
/	除号	两数相除，若两个整数相除则值取整数部分，例如3/2的值为1
%	求余（取模）	两个整数相除求余数，例如5%3的值为2
（）	圆括号	形式和使用方法同数学中的圆括号



C++语言以圆括号代替数学中的方括号和花括号，程序运行时将按由内到外的顺序计算圆括号里的值。例如数学表达式2×{4-[3-(2-9)]}，转换成C++语言中的表达式为2*(4-(3-(2-9)))。

诸如x^2+y^2这样的表达式，转换成C++语言中的表达式为x*x+y*y。

数学中常用x、y等字符代表未知数或未定数，例如代数方程$x+y$等。类似地，C++语言也用字符代表某个数值，其在程序运行中没有固定值、可以改变的数值称为变量。

代表变量的字符只能由字母、数字、美元符号和下划线等字符组成，且第一个字符必须为字母、下划线或美元符号。例如，可以定义变量名为_sum、abc、Day1、school_name、lotus_1_2、$sum，而Mr.Wang、234NUM、#34、a＞＝b、¥1234这些变量名是错误的。



为增加程序的可读性，应注意做到“见名知意”，即选择有含义的英文单词（或其缩写）作为变量名，例如count、total、price等，尽量少用简单的符号，例如a、b、c、u1、v1等。

1.2.1　求两个整数的积和平均值（calc）

【题目描述】

试编写一个程序，使得程序运行时，用键盘输入两个整数（两个整数之间以一个或多个空格间隔）后按Enter键，计算机显示器上输出这两个整数的积和平均值。

【输入格式】

输入两个整数。

【输出格式】

第一行输出两个整数相乘的完整表达式。

第二行输出平均值（仅输出整数部分），注意最后应以换行结束。

【输入样例】

5 6

【输出样例】

5*6=30

5

参考程序如下所示。

```
 1 /*
 2   这个程序的名字叫作求两个整数的积和平均值，它真的太强大了！好吧，
 3   这么说有点无聊，但反正是注释语句，不影响程序运行
 4 */
 5 #include <bits/stdc++.h>
 6 using namespace std;
 7
 8 int main()                              //这也是注释语句，不影响程序运行
 9 {
 10   int a,b,c;                           //定义变量 a、b、c，此时a、b、c的值
未知
 11   cin>>a>>b;                           //输入a和b的值
 12   c=a*b;                               //计算a和b的乘积，把结果赋值给c
 13   cout<<a<<"*"<<b<<"="<<c<<endl;//输出的变量和字符串要用"<<"隔开
 14   cout<<(a+b)/2<<endl;                 //整数除整数会舍去小数部分，自动取整
 15   return 0;
 16 }
```

第1～第4行中的“/*…… */”用作C++代码的多行注释，表示从“/*” 开始，到“*/”结束的这一段语句均为注释部分。

“//”用作C++代码的单行注释，表示从“//”开始，一直到本行结束的语句均为注释部分。



注释语句只是方便阅读代码的人更好地理解代码，编译程序时，编译器会自动忽略注释语句，它不会对程序的运行产生任何影响。

---

int a,b,c;定义了a、b、c这3个整型变量，C++语言规定变量必须要先定义才能使用。定义变量相当于向计算机内存申请“房子”，变

量只有住进“房子”里才允许使用，数据类型规定的存储空间有多大，分配的“房子”就有多大。例如定义了两个变量，如图1.14所示。因为a是整型变量，所以内存给它分配了4字节的空间；因为x是超长整型变量，所以内存给它分配了8字节的空间。

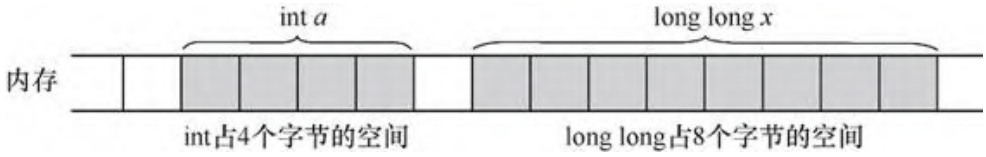


图1.14

可以在定义变量数据类型的同时给它赋值（初始化），例如：

```
int a=5;          //定义a为整型变量，初始值为5
float f=3.45;     //定义f为单精度浮点型变量，初始值为3.45
```

诸如int a=5；这样的语句，相当于下面两条语句：

```
int a;
a=5;
```

也可以对定义的变量的其中一部分变量赋值，例如：

```
int a，b，c=5;      //定义了整型变量a、b、c，其中c的初始值为5，a和b的初始值未知
```

注意，在程序中a、b、c之间要用逗号分隔。

如果对几个变量均赋值为5，应该写成int a=5,b=5,c=5;，而不能写成int a=b=c=5;。



变量数据类型的定义应视题目的要求而定，选取的数据类型的字节长度数值过大会降低运算速度、浪费内存空间，过小可能会导致数据溢出。

第11行是输入语句，cin表示程序运行时，用键盘输入“>>”右边的变量的值。 cin>>a>>b；表示用键盘输入两个整数。

第12行中的“=”和数学运算中用到的等号不同，它是赋值运算符，表示将右边的值赋给左边的变量，所以如果将代码写成a+b=c，程序将无法编译。

第13行中的endl代表换行，类似于换行符“\n”。



用键盘输入数据时，要严格按输入语句要求的格式输入。例如上例中，如果a=1，b=2，那么cin>>a>>b语句的正确输入方式应该是用英文输入法输入半角字符1 2后按Enter键，其中1和2之间以一个或多个空格分隔，但不能是其他符号。如果用键盘输入a=1,b=2，或者1，2之类的均是错误的。

输入/输出语句应严格按照格式编写，例如不能写成cin>>a,b,c或者cout<<a+b=c等。输入/输出的各变量之间应使用“>>”或“<<”分隔。

1.2.2　3个浮点数相加（add2）

【题目描述】

输入3个双精度类型的浮点数，计算3个浮点数相加的和。

【输入格式】

输入3个浮点数（整数部分不超过2位，小数部分不超过3位）。

【输出格式】

输出3个浮点数相加的和，注意最后应以换行结束。

【输入样例】

1.2 3.2 1

【输出样例】

5.4

浮点数在内存中是用有限的存储单元存储的，所以能提供的有效数字总是有限的，在有效位以外的数字将会被舍去，由此可能会产生一些误差。

下面的程序演示了浮点数的舍入误差。

```
 1 //浮点数的舍入误差演示
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   float a=2345678900.001;          //定义一个很大的浮点数
 8   float b=3;                       //定义一个很小的浮点数
 9   cout<<fixed<<a<<endl;            //fixed表示可禁止以科学记数法形式
输出
 10   float c=a+b;
 11   cout<<c<<endl;
 12   return 0;
 13 }
```

输出结果如图1.15所示。

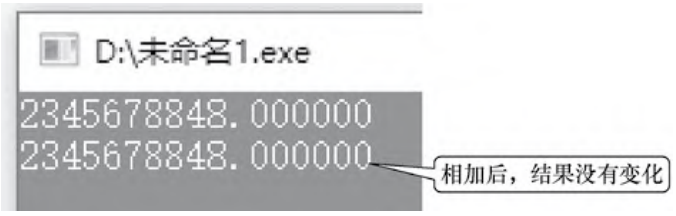


图1.15

可以看到，两个浮点数相加后的结果没有变化，这是由于单精度浮点型的精度为6～7位有效数字，后面的数字被忽略，并不能准确地

表示该数。例如定义float f=3456.12345;，当运行cout<<f语句输出f时，结果为3456.12。

所以在使用浮点数时要避免因浮点数运算产生的误差而导致结果错误。例如，应避免将一个很大的浮点数和一个很小的数直接运算，否则会“丢失”小的数。

1.2.3 捡石头（stone）

【题目描述】

小光捡了3块石头，这3块石头的质量的输入在代码中用cin语句实现，他想再捡一块石头，让这4块石头的质量正好为30千克，请编程用cout语句输出第4块石头的质量。

【输入格式】

输入3个数，数与数之间以空格间隔，表示3块石头的质量。

【输出格式】

输出一个数，表示第4块石头的质量，注意最后应以换行结束。

【输入样例】

1 1 1

【输出样例】

27

1.2.4 简单解方程（equation）

【题目描述】

设$S=(a+b)/2$，给定a与S的值，求b的值。

【输入格式】

输入两个整数a和S（$-1\,000 \leqslant a$，$S \leqslant 1\,000$）。

【输出格式】

输出一个整数b。

【输入样例】

4 3

【输出样例】

2

1.2.5　计算多项式的值（cal）

【题目描述】

试计算多项式$y=ax^3+bx^2+cx+d$的值，其中a、b、c、d、x的值由键盘输入。

【输入格式】

输入5个整数，分别代表a、b、c、d、x的值。

【输出格式】

输出一个整数，即多项式的值，注意输出最后应以换行结束。

【输入样例】

3 4 5 6 5

【输出样例】

506



初学者容易将表达式ax^3+bx^2+cx+d写成ax^3+bx^2+cx+d这样的错误形式，其正确形式应该为a*x*x*x+b*x*x+c*x+d。

程序写好后，试输入1999 1999 1999 1999 1999，观察输出结果有什么异常。

每种类型的数据都有各自的取值范围，请务必确保所定义的数据不超过该类型数据的取值范围，否则会造成数据的溢出而导致结果错误。数据的溢出在编译和运行程序时并不报错，完全要靠编程者的细心和经验来保证结果的正确性。

下面的程序显示了整型数据溢出的错误。

```
 1 //整型数据的溢出
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a=2147483647;
 8   a=a+1;
 9   cout<<"a="<<a<<endl;                    //输出a=-2147483648
10   return 0;
11 }
```

1.2.6 混合运算（ca12）

【题目描述】

试编程用键盘输入a、b、c、d、e的值后计算图1.16所示的算式的值。

$$\frac{ab+ac}{b+\frac{c}{d-e}}$$

图1.16

【输入格式】

输入5个双精度浮点数，分别为a、b、c、d、e的值。

【输出格式】

输出混合运算的结果。

【输入样例】

2.1 2.2 2.3 2.4 2.5

【输出样例】

-0.454327

如表1.3所示，如果程序中定义的某个数据类型的值永远不可能出现负数时，可以使用unsigned将其定义为无符号的数据类型，这样就使正整数的取值范围大约比原先扩大了一倍。

表1.3

数据类型	字节长度	取值范围
unsigned short	2	0～65 535
unsigned long	4	0～4 294 967 295
unsigned int	4	0～4 294 967 295
unsigned long long	8	0 ～$2^{64}-1$

1.2.7　两个整数相乘（mul）

【题目描述】

使用cin语句输入两个整数a和b，计算两个整数的乘积并输出。

【输入格式】

输入两个整数a和b（$1 \leqslant a$，$b \leqslant 4\,000\,000\,000$）。

【输出格式】

输出两个整数的乘积。

【输入样例】

100 10

【输出样例】

1000

1.2.8 巨型人造天体（aster）

【题目描述】

制造巨型人造天体需要A部件和B部件，其质量分别为2 012 345 678克、1 912 345 678克。

（1）试计算这两种部件的质量之和的最后4位数。

（2）试计算12 345 678个A部件的总质量的最后4位数。

【输入格式】

无输入。

【输出格式】

输出两行，第一行为两种部件的质量之和的最后4位数，第二行为12 345 678个A部件的总质量的最后4位数。



使用“内容市场”
App扫描看视频

【输入样例】

略。

【输出样例】

略。

【算法分析】

提示：(a+b)%c=(a%c+b%c)%c，(a*b)%c=(a%c*b%c)%c。

与变量相对应，在程序运行中，值不能被改变的数值称为常量，常量使用关键字const标识，例如：

const int num=34;　　//定义num为整型常量，值为34

const float pi=3.1415;//定义pi为单精度浮点型常量，值为3.1415

程序运行时，常量的值不能被更改。下面是一个试图改变常量a的错误程序，程序编译会出现“assignment of read-only variable ’a’ ”的错误，即只读变量'a'的赋值错误。

```
 1 //试图改变常量a的错误程序
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   const int a=34;                    //定义了一个整型常量a，a的值为34
 8   a=a*5;                             //试图将a乘以5后再将其值赋给a
 9   cout<<a<<endl;
10   return 0;
11 }
```

也可以用预处理指令#define定义常量的值，例如下面的程序中用#define定义PRICE代表数值34，此后凡是在该程序中出现的PRICE都代表34。常量名一般用大写字母表示，可以和变量一样运算。

```
 1 //预处理指令定义常量
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4 #define PRICE 34     //定义PRICE为一个常量，值为34，注意末尾无分号
 5
 6 int main()
 7 {
 8   int num=10;
```

```
 9    int total=num*PRICE;                          //PRICE会替换为34来运算
10    cout<<"total="<<total<<endl;                  //输出结果为total=340
11    return 0;
12 }
```



定义常量的好处是含义清晰，而且在需要改变常量值时能做到“一改全改”。例如，将#define PRICE 34改为#define PRICE 100，则程序中所有用到PRICE的地方就全部由34变成100而无须一个一个修改。

1.3 字符和字符串



扫码看视频

字符变量用于存放单个字符，用单引号标注，例如 ’a’、’x’、’?’、 ’$’等。

字符串变量用于存放任意多个字符，用双引号标注，例如"Hello,world!"、"a" 、"$34567"等。

每一个字符串的结尾有一个“字符串结束标志”，以便系统判断字符串是否结束。C++语言规定以字符'\0'作为字符串结束的标志，'\0'占1字节。例如，如图1.17所示，字符串"Hello"的实际存储空间不是5字节，而是6字节，最后一个字节为'\0'。但'\0'是隐含的，输出时不会输出'\0'，赋值时也不必手动加'\0'，这些系统会自动进行后台处理。



图1.17

字符变量的使用很简单，参考例程如下所示。

```
 1 //字符变量的使用
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   char a='A',b='B',c,d;    //定义字符变量a、b、c、d，a和b已赋值
为'A'和'B'
 8   cin>>c>>d;                                    //从键盘输入c和d的值
 9   cout<<a<<' '<<b<<' '<<c<<' '<<d<<'\n';//输出,字符间以一个空格间隔
10   return 0;
11 }
```



不要将字符变量和字符串变量混淆，例如，char c="a"这种把字符串"a"赋给只能容纳一个字符的字符变量c的语句是错误的，因为字符串"a"实际包含两个字符'a'和'\0'。

1.3.1　字符金字塔（Pyramid）

【题目描述】

试编程输入一个字符，用该字符构造一个高度为3的字符金字塔。例如，输入字符为“*”，则输出的字符金字塔如下。

```
  *
 * * *
* * * * *
```

【输入格式】

用键盘输入一个字符。

【输出格式】

输出用该字符构造的一个高度为3的字符金字塔，第1行行首有4个空格，第2行行首有2个空格，第3行行首无空格，非空格的字符间以一个空格间隔，行末无多余空格。

【输入样例】

*

【输出样例】

*

* * *

* * * * *

英文字母、数字以及一些常用的符号（例如*、#、@等）在计算机中是使用二进制数来表示的。具体用哪些二进制数表示哪个符号，每个人都可以约定自己的一套编码规则，但如果想互相通信而不造成混乱，就必须使用相同的编码规则。美国国家标准协会（American National Standards Institute,ANSI）出台了美国信息交换标准代码（American Standard Code for Information Interchange, ASCII），统一规定了上述常用符号用哪些二进制数来表示。

ASCII对照表请参见本书的附录部分。表1.4中列出了几个常用的码表值。

表1.4

ASCII值（十进制）	字符	ASCII值（十进制）	字符
48	0	57	9

ASCII值（十进制）	字符	ASCII值（十进制）	字符
65	A	90	Z
97	a	122	z

字符在内存中以该字符相对应的ASCII值存储，它的存储形式与整数的存储形式类似，这样使得字符型数据和整型数据可以通用。一个字符既可以以字符形式输出，也可以以整数形式输出，还可以对它们进行算术运算。

下面的程序演示了字符变量与整数之间的通用性。

```
 1 //字符变量与整数的通用性演示
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   char c1,c2;
 8   c1=65;                          //此处用数字而不是字符赋值
 9   c2=66;                          //此处用数字而不是字符赋值
10   cout<<c1<<"  "<<c2<<endl;    //因为c1、c2为字符类型，所以默认输出字符
11   cout<<int(c1)<<"  "<<int(c2)<<'\n';           //int()表示强制转换为整数
12   return 0;
13 }
```

输出结果如图1.18所示。

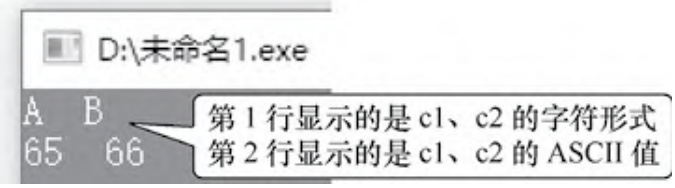


图1.18

程序中的第11行使用int(c1)将字符c1强制转换为整数。C++可以利用强制类型转换运算符将表达式转换成所需的类型，例如：

```
cout<<int(3.84); //将浮点数3.84强制转换为整数，故舍去小数部分，输出整数3
cout<<fixed<<double(10);//将整数10强制转换为浮点数10.0000输出
```

C++有两种转换，一种是系统自动进行的类型转换，例如3+3.4的结果是浮点数；另一种是强制类型转换。当系统自动进行的类型转换不能实现目的时，可以用强制类型转换。例如，取模运算符"%"要求其两侧均为整数，若x为浮点数，则x%3不合法，必须用int(x)%3进行转换。由于强制类型转换运算符优先级高于"%"运算符，因此先强制转换浮点数x为整数，再进行取模运算。

1.3.2 ASCII值转字符（ASCII1）

【题目描述】

输入一个整数，然后输出对应的字符。

【输入格式】

输入一个整数（保证整数存在对应的可见字符）。

【输出格式】

输出对应的字符。

【输入样例】

65

【输出样例】

A

1.3.3　小写字母转大写字母

【题目描述】

下面的程序实现了小写字母转对应的大写字母。

```
1 //小写字母转大写字母
2 #include <bits/stdc++.h>
3 using namespace std;
```

```
 4
 5 int main()
 6 {
 7   char c1,c2;
 8   c1='a';                    //c1赋值为小写字母'a'
 9   c2='b';                    //c2赋值为小写字母'b'
10   c1=c1-32;                  //小写字母比对应的大写字母的ASCII值大32
11   c2=c2-32;
12   cout<<c1<<"  "<<c2<<'\n';//输出大写字母：A  B
13   return 0;
14 }
```



从附录的ASCII对照表可以看出，每一个小写字母比它对应的大写字母的ASCII值大32。C++语言允许字符数据与整数直接进行算术运算，这种处理方法增大了程序的自由度，对字符做各种转换比较方便。

1.3.4　大写字母转小写字母（ASCII2）

【题目描述】

试编程使用cin语句输入3个大写字母，并将之转换为对应的小写字母输出。

【输入格式】

输入3个大写字母，字母间以一个空格间隔。

【输出格式】

输出3个对应的小写字母，字母间以一个空格间隔，行末无空格，有换行。

【输入样例】

A B C

【输出样例】

a b c

1.3.5　恺撒加密术

【题目描述】

“恺撒加密术”是一种替代密码的加密技术。如图1.19所示，对于信件中的每个字母，会用它后面的第t个字母代替。例如当t=4时，“China”加密的规则是用原来字母后面的第4个字母代替原来的字母，即字母“A”后面的第4个字母是“E”，用“E”代替“A”。因此，“China”应加密为“Glmre”。请编写程序将任意5个字符加密（暂不考虑替代的字母跳过“界”的问题）。

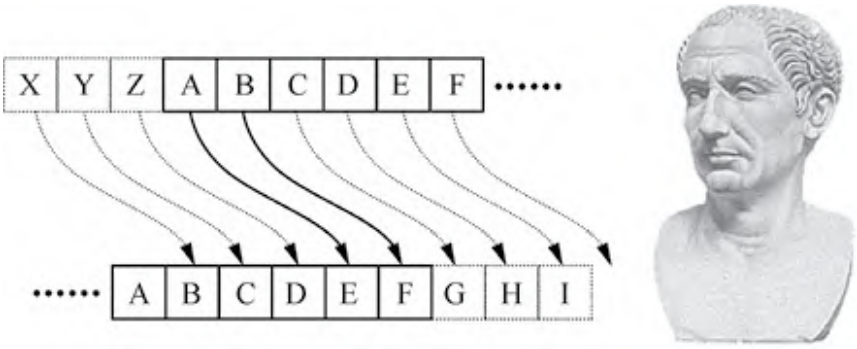


图1.19

```
 1 //恺撒加密术
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   char c1='C',c2='h',c3='i',c4='n',c5='a';//此处也可换成其他英文字母
 8   c1+=4;                                  //c1+=4即c1=c1+4，这种写法更简
洁
 9   c2+=4;
10   c3+=4;
11   c4+=4;
12   c5+=4;
13   cout<<c1<<c2<<c3<<c4<<c5<<endl;
14   return 0;
15 }
```

在运算符“=”之前加上其他运算符，称为复合运算符，例如：

```
a+=3;//等价于a=a+3
x%=3;//等价于x=x%3
```

如果是包含若干项的表达式，则相当于它有圆括号，例如x%=y+4相当于x%=(y+4)，即x=x%(y+4)而不是x=x%y+4。

1.3.6　改进的加密术（encryption）

【题目描述】

请编写一个程序，使用cin语句输入任意5个英文字符和任意一个值t（-8＜t＜8），则原先的5个英文字符将分别用其后面的第t个英文字符代替并输出（暂不考虑替代的字符跳过“界”的问题）。

【输入格式】

第1行输入5个英文字符，字符间以空格间隔。

第2行输入1个整数t。

【输出格式】

输出加密后的英文字符，字符间无须空格间隔。

【输入样例】

C h i n a

4

【输出样例】

Glmre

参考伪代码如下所示，请根据伪代码的思路编写C++程序完成本题。

```
1 定义5个字符变量
2 定义t
3 输入5个字符变量的值
4 输入t的值
5 5个字符依次以其后面的第t个字符代替
6 输出5个加密后的字符
```

C++还允许一种特殊形式的字符常量，就是以“\”开头的字符序列。例如前面已经用过的'\n'，它代表换行符。这是一种“控制字符”，是不能在显示器上显示的，在程序中也无法用一般形式的字符表示，只能采用特殊形式的字符表示。

以“\”开头的常用特殊字符（转义字符）如表1.5所示。

表1.5

字符形式	含义	ASCII值
\n	换行符，从当前位置移到下一行	10
\	反斜线字符	92

字符形式	含义	ASCII值
\’	单引号字符	39
\″	双引号字符	34

通过转义字符，就可以输出诸如“\”“’”“″”之类的字符了。

例如要输出“\”，代码应该为cout<<“\\”。

1.3.7　输出特殊字符（special）

【题目描述】

请输出图1.20所示的字符串，注意所有的字符均为英文半角形式。

```
He said:"The symbol is '\'."
```

图1.20

【输入格式】

无输入。

【输出格式】

见输出样例，注意除单词之间以一个空格间隔外，无任何多余空格，以换行结束。

【输入样例】

无。

【输出样例】

He said:″The symbol is '\'.″

为了更好地处理字符串，C++提供了string类，它提供了添加、删除、插入和查找等丰富的操作方法。

string类的部分操作方法如下所示。

```
 1 //string类的部分操作方法
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   string s="Hi,..morn";                          //定义了名为s的string类
并初始化
 8   s=s+"ing";                                     //尾部添加字符串"ing"
 9   cout<<"添加字符串:"<<s<<endl;                  //输出"Hi,..morning"
10   s.erase(3,2);                                  //删除第3个字符后的2个字
符
11   cout<<"删除字符串:"<<s<<endl;                  //输出"Hi,morning"
12   int f=s.find("Hi,");                            //查找"Hi,"在s中的位
置，-1为无法找到
13   s.insert(f+3,2,'G');                           //在第3个字符后插入单个字
符'G'2次
14   cout<<"插入两字符:"<<s<<endl;                  //输出"Hi,GGmorning"
15   s.insert(5,",MM,");                            //在第5个字符后插入字符
```

```
串",MM,"
16    cout<<"插入字符串:"<<s<<endl;                    //输出”
Hi,GG,MM,morning”
17    string v=s.substr(4,3);                          //取s中第4个字符后的3个
字符给v
18    cout<<"字符串子串:"<<v<<endl;                    //输出"G,M"
19    cout<<"string长度:"<<v.length();                 //输出3,即v的长度
20    return 0;
21 }
```

输出结果如图1.21所示。

```
添加字符串:Hi,..morning
删除字符串:Hi,morning
插入两字符:Hi,GGmorning
插入字符串:Hi,GG,MM,morning
字符串子串:G,M
string长度:3
```

图1.21

1.3.8 费解的对话（chat）

【题目描述】

宠物机器人的单词库被病毒感染了，第一种感染方式是圆括号内的内容没有被破坏，但是圆括号外面加入了一些奇怪的数字，例如“（Mary）”变成了“253(Mary)5”；第二种感染方式是在单词中加了一个“...”子串，例如“hello”变成了“he…llo”。请修复它。



使用“内容市场”App扫描看视频

【输入格式】

输入两行字符串（字符串中无空格）。第一行是被第一种感染方式感染的字符串，第二行是被第二种感染方式感染的字符串。

【输出格式】

输出一行对应的正确字符串，两个字符串间以逗号间隔。

【输入样例】

23453(Mary)24565

mor…ning

【输出样例】

Mary,morning

1.4　输入/输出及格式控制



扫码看视频

getchar()函数（字符输入函数）的作用是从终端（键盘）获取输入的字符。

putchar()函数（字符输出函数）的作用是向终端（显示器）输出字符。例如下面的程序在运行时，如果用键盘输入字符后按Enter键，显示器上就会输出该字符。

```
 1 //getchar()函数和putchar()函数的使用演示
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   char c;
 8   c=getchar();                                    //输入字符赋值给c
 9   putchar(c);                                     //输出字符变量c
10   return 0;
11 }
```

cin读取一行字符串时，遇到空格就会停止。如果读取一行包含空格符的字符串，可以使用getline()函数。例如将“I am here”赋值

给string类的例程如下所示。

```
 1 //使用getline()函数读取string类字符串
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   string str;                    //string类可用getline()函数读取
 8   getline(cin,str);              //读取一行包含空格符的字符串赋值给str
 9   cout<<str;
10   return 0;
11 }
```

某些情况下，程序需要以八进制数或十六进制数的形式输出结果。C++提供了简单的实现方法：只要在输出流中输出dec（对应十进制）、oct（对应八进制）或hex（对应十六进制）即可。下面的程序演示了不同进制数之间的相互转换。

```
 1 //进制转换
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int n=314;
 8   cout<<"十进制："<<dec<<n<<endl;  //n本身为十进制数，所以可省略dec
 9   cout<<"八进制："<<oct<<n<<endl;
10   cout<<"十六进制:"<<hex<<n<<endl;
11   cin>>oct>>n;                     //输入八进制形式的n值
12   cout<<dec<<n;                    //转换为十进制形式的n值输出
13   return 0;
14 }
```

C++提供的setprecision()函数可以控制浮点数的输出精度，其中的参数代表输出的位数。试分析以下程序的执行结果。

```
 1 //setprecision()函数的使用
 2 #include <bits/stdc++.h>
```

```
 3 using namespace std;
 4
 5 int main()
 6 {
 7   double a=234.1234567890;
 8   cout<<a<<endl;
 9   cout<<setprecision(4)<<a<<endl;
10   cout<<setprecision(5)<<a<<endl;
11   cout<<setprecision(6)<<a<<endl;
12   cout<<setprecision(7)<<a<<endl;
13   cout<<setprecision(8)<<a<<endl;
14   cout<<setprecision(9)<<a<<endl;
15   cout<<setprecision(15)<<a<<endl;
16   return 0;
17 }
```

输出结果如图1.22所示。

```
234.123
234.1
234.12
234.123
234.1235
234.12346
234.123457
234.123456789
```

图1.22

可见，单纯使用setprecision()函数，控制输出的位数是整数部分与小数部分相加的位数，如果想真正精确地控制小数点后的位数，需添加fixed()函数，见以下程序。

```
 1 //fixed()函数的使用
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   float a=111.2345678;
 8   cout<<setprecision(5)<<fixed<<a<<endl;    //输出111.23457，有误差
 9   return 0;
10 }
```

1.4.1　精确到小数点（fixed）

【题目描述】

输入一个双精度浮点数，保留其小数点后5位输出。例如输入1.23，输出1.230 00。

【输入格式】

输入一个双精度浮点数。

【输出格式】

将输入的双精度浮点数保留其小数点后5位输出。

【输入样例】

1.0

【输出样例】

1.00000

1.5　一些运算规则

扫码看视频

整数和浮点数可以混合运算，并且由于字符可以与整数通用，因此，整数、浮点数、字符可以混合运算，例如134+’a’+3.4-45.8*’c’是合法的。

在运算时，系统会自动将不同类型的数据转换成同一类型的数据后再进行运算。转换的规则如图1.23所示。

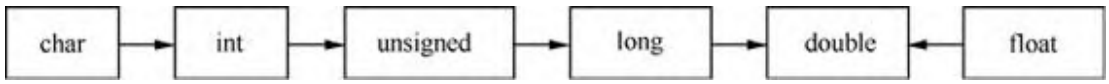


图1.23

运算顺序为由左到右。例如10+’a’+int*float-double/long的运算顺序如下。

（1）进行10+’a’的运算，先将'a'转换成整数，结果为107。

（2）由于“*”比“+”优先，先进行int*float的运算，将两数都转换为双精度浮点数。

（3）107与int*float的积相加，先将整数转换为双精度浮点数，结果为双精度浮点数。

（4）将变量long转换为双精度浮点数，double/long的结果为双精度浮点数。

（5）将10+'a'+int*float的结果与double/long的商相减，结果为双精度浮点数。

C++语言还规定了运算符的优先级和结合性，在表达式求值时，先按运算符的优先级高低顺序执行，例如先乘除后加减。如果一个运算对象两侧的运算符的优先级相同，例如a-b+c，则按规定的“结合方向”执行。

大多数算术运算符的结合方向为“先左后右”，即左结合性，例如a-b+c中，b两侧的“-”和“＋”两种算术运算符的优先级相同，按先左后右的结合方向，b先与减号结合，执行a-b的运算，再执行加c的运算。有些运算符（例如赋值运算符“=”）的结合方向为“自右向左”，即右结合性，例如语句int a,b=1,c=2;a=b=c中，a=b=c相当于a=(b=c)，即a=2。

C++语言的自增自减运算符“++”和“--”的作用是使变量的值加1或减1，例如：

++i，--i（在使用i之前，使i的值加/减1）；

i++，i--（在使用i之后，使i的值加/减1)。

“++”和“--”不能用于常量或表达式，例如5++或(a+b)++都是不合法的。

“++”和“--”的结合方向是自右向左。

粗略地看，++i和i++的作用相当于i=i+1，但它们的不同之处在于++i是执行i=i+1后再使用i的值，而i++是使用i的值后再执行i=i+1。例如，假设i的原值等于3，分别执行下面的赋值语句的区别如下。

（1）j=++i;。此赋值语句执行时，i的值先变成4，再赋给j，j的值为4。

（2）j=i++;。此赋值语句执行时，先将i的值3赋给j，j的值为3，然后i的值变为4。

例如当i=3时，cout<<++i的结果为4。若改为cout<<i++则输出结果为3。

例如当i=1时，i=(++i)+(++i);，执行cout<<i的结果为6，因为最后i的值为3，i＝(i)+(i)即i=3+3=6。

试运行下面的程序并分析结果。

```
 1 //i++与++i
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int i=1;
 8   cout<<++i<<endl;
 9   cout<<i++<<endl;
10   cout<<i<<endl;
11   cout<<(++i)+(++i)<<endl;
12   cout<<(i++)+(i++);  //编译器不同，值可能为10或11，所以要避免这样写
13   return 0;
14 }
```

C++语言提供一种特殊的运算符——逗号运算符，用它将两个表达式连接起来，例如：

3+5，6+8

逗号表达式的求解过程是：先求解表达式1，再求解表达式2。整个逗号表达式的值是表达式2的值，例如cout<<(3+5,6+8)输出的值为14。



之前定义变量时已经使用过逗号表达式了，例如：

```
int a,b=0,c,d=34;        //这相当于一句代码
double x,y,z;            //这相当于一句代码
```

第02章　基本结构

任何简单或者复杂的代码都是由顺序结构、选择结构和循环结构这3种基本结构组合而成的，所以这3种结构被称为程序设计的3种基本结构。

2.1　顺序结构



扫码看视频

如图2.1所示，顺序结构如同自然语言中的文章一样，按照事件的发展顺序，依次自上而下书写程序语句，并以语句出现的顺序来执行。

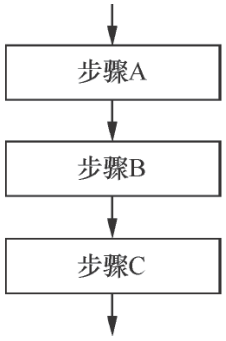


图2.1

2.1.1　三位数分解（check）

【题目描述】

输入一个三位数num，试将该数拆为百位数、十位数和个位数后逐行输出。

【输入格式】

输入一个三位数num。

【输出格式】

将输入的三位数num拆为百位数、十位数和个位数后逐行输出。

【输入样例】

123

【输出样例】

Hundreds is 1

Tens is 2

Ones is 3



本题需要找到通用的方法分解一个三位数，例如num=398,分解num这个三位数，一种可行的方法如下。

（1）取其个位：num%10=8。

（2）取其十位：num/10%10=9。

（3）取其百位：num/100=3（整数除整数的结果还是整数）。

你还能想到其他的方法吗？

```
1 //三位数分解
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
6 {
7   int num;
8   cin>>num;                    //注意：是输入 1 个三位数而不是 3 个一位数
9   cout<<" Hundreds is "<<num/100<<endl;
10  cout<<" Tens is "<<num/10%10<<endl;
11  cout<<" Ones is "<<num%10<<endl;
12  return 0;
13 }
```

2.1.2　反向输出四位数（reverse）

【题目描述】

用键盘输入一个四位数后反向输出，例如输入1234，输出4321。

【输入格式】

输入一个四位数。

【输出格式】

将输入的四位数反向输出，如果倒序的数有前导0，则不应输出前导0（参见样例）。

【输入样例】

1230

【输出样例】

321

2.1.3　小数的四舍五入（rounding）

【题目描述】

输入一个浮点数，实现小数点后第3位四舍五入。

【输入格式】

输入一个浮点数。

【输出格式】

对该数的小数点后第3位四舍五入后输出，保留小数点后两位，即使为0。

【输入样例】

1.235

【输出样例】

1.24



设输入的浮点数为x，假设要操作的数据x=2.3567，如果希望保留两位小数实现第3位四舍五入，那么执行x=(int)(x*1000+5)/10)可得x=236，再执行x/=100.0即可得x=2.36。

2.1.4　求等差数列末项（series）

【题目描述】

等差数列是指从第二项起，每一项与它的前一项的差等于同一个常数的一种数列，这个常数叫作等差数列的公差d。例如等差数列1,3,5,7,9,…中，a_1为第一项即1，公差d=2，第n项$a_n=a_1+d\times(n-1)$。现输入一个等差数列的前两项a_1、a_2和n，试求第n项即a_n的值。

【输入格式】

输入3个整数，即等差数列前两项的值和n。

【输出格式】

输出等差数列第n项的值。

【输入样例】

1 2 5

【输出样例】

5

2.1.5 徽章（badge）

【题目描述】

已知一个徽章的价格是2元1角，试求*a*元*b*角最多能买多少个徽章。

例如用键盘输入10 3，表示10元3角，则最多能买4个徽章。



使用“内容市场”
App扫描看视频

【输入格式】

用键盘输入两个整数*a*和*b*。

【输出格式】

输出一个整数，即最多能买到的徽章的个数。

【输入样例】

10 3

【输出样例】

4

2.1.6 计算梯形的面积（trapezoid）

【题目描述】

如图2.2所示，输入梯形的上底up和下底down的长度，以及梯形中阴影部分的面积area，试计算梯形的面积。

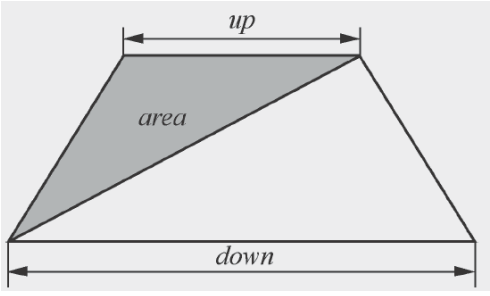


图2.2

【输入格式】

输入3个整数up、down及area，即梯形的上底的长度、下底的长度及阴影部分的面积。

【输出格式】

考虑到浮点数的误差问题，输出梯形的面积为整数。

【输入样例】

10　20　100

【输出样例】

300



梯形的面积公式为(上底+下底)×高/2，题中已知梯形的上底和下底的值，唯独缺少梯形的高，所以要想办法算出梯形的高。

可以发现，上三角形（阴影部分）的高即梯形的高，则由三角形的面积公式(底×高)/2可以算出梯形的高。

2.1.7　二进制存储（bin）

【题目描述】

计算机的最小存储单位是位，一位可以保存1或者0，这样可以表示1或0两个二进制数，两位可以表示00、01、10、11这4个二进制数，那么*n*位可以表示多少个二进制数呢？



使用“内容市场”
App扫描看视频

【输入格式】

输入一个整数n（$n \leqslant 32$）。

【输出格式】

输出一个整数，即n位可以表示多少个二进制数。

【输入样例】

2

【输出样例】

4



加法原理：做一件事情，完成它有n类方式，第一类方式有M_1种方法，第二类方式有M_2种方法，……，第n类方式有M_n种方法，那么完成这件事情共有$M_1+M_2+\cdots+M_n$种方法。如图2.3所示，C地到D地有2条公路、2条铁路，则C地到D地共有4条路。

乘法原理：做一件事情，完成它需要分成n个步骤，完成第一步有m_1种不同的方法，完成第二步有m_2种不同的方法，……，完成第n步有m_n种不同的方法，那么完成这件事情共有 $m_1 \times m_2 \times \cdots \times m_n$种不同的方

法。如图2.3所示，A地到B地有2条路，B地到C地有3条路，C地到D地有4条路，则A地到D地共有2×3×4＝24（种）方案。

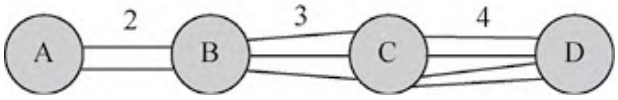


图2.3

所以由乘法原理可知：每一位可以放0或1两种方案，则两位可以表示4个数，即2的2次方，三位可以表示8个数，即2的3次方……

pow(a,b)可以计算出a的b次方的值，例如：

```
cout<<"2的8次方为"<<pow(2,8)<<endl;
```

```
cout<<"2的16次方为"<<pow(2,16)<<endl;
```

但它计算的结果是双精度浮点数，实际操作时可能需要将其转换为整数输出。

2.1.8　校庆日（day）

【题目描述】

7月20日是某学院的校庆日，已知7月1日是星期六，问7月20日是星期几？



7月1日～7月20日是20-1=19（天）。

19÷7=2（星期）……5（天）

将星期一～星期六以数字1—6表示，星期天以数字0表示，则7月20日为(6+5)÷7=1（星期）……4（天），所以7月20日是星期四。

程序中的求余运算符以“%”表示，例如计算19÷7的余数可以写成19%7。

注意计算余数时，除数与被除数均应为整数。

参考程序如下所示。

```
 1 //校庆日
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int day=20-1;
 8   cout<<"共经过了："<<day/7<<"星期，多"<< day%7<<"天"<<"\n";
 9   cout<<"校庆日是星期"<< (6+ day%7)%7 <<"\n";
10   return 0;
11 }
```

2.1.9 计算星期几（week）

【题目描述】

假设今天是星期天，用键盘输入a和b（$0<a<10$，$0<b<10$），试计算a^b天之后是星期几（以数字1～7表示星期一～星期天）。

【输入格式】

输入两个整数a和b。

【输出格式】

输出a^b天之后是星期几（以数字1~7表示星期一～星期天）。

【输入样例】

2 2

【输出样例】

4

2.1.10　求余数（remainder）

【题目描述】

计算两个双精度浮点数a和b相除的余数，a和b都是正数。这里余数“r”的定义是$a=k\times b+r$，其中k是整数，$0\leqslant r<b$。

【输入格式】

输入两个双精度浮点数a和b。

【输出格式】

输出两个双精度浮点数a和b相除的余数（四舍五入，保留小数点后两位）。

【输入样例】

73.263 0.9973

【输出样例】

0.46

2.1.11 求二次方根（sqrt）

【题目描述】

二次方根又称为平方根，数学符号为“$\sqrt{}$”，它是平方的逆运算，例如$4^2=16$，则 $\sqrt{16}=4$。C++语言里使用sqrt来表示平方根，例如int x=sqrt(25)，则x的值为5。

现输入一个整数，求它的平方根，输出结果向下取整。

【输入格式】

输入一个整数x（$1 \leqslant x \leqslant 100\,000$）。

【输出格式】

输出x的平方根，结果向下取整（取比本身小的最大整数）。

【输入样例1】

49

【输出样例1】

7

【输入样例2】

5

【输出样例2】

2

参考程序如下所示。

```
 1 //求二次方根
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a;
 8   cin>>a;
 9   cout<<int(sqrt(a))<<endl;
10   return 0;
11 }
```

2.1.12　勾股定理（triangle）

【题目描述】

如图2.4所示，设直角三角形的两条直角边的长度为a和b，斜边的长度为c，则有$a^2+b^2=c^2$。这就是有名的勾股定理（我国古代称直角三角形为勾股形，直角边中长度较小的边为勾，另一长直角边为股，斜边为弦）。

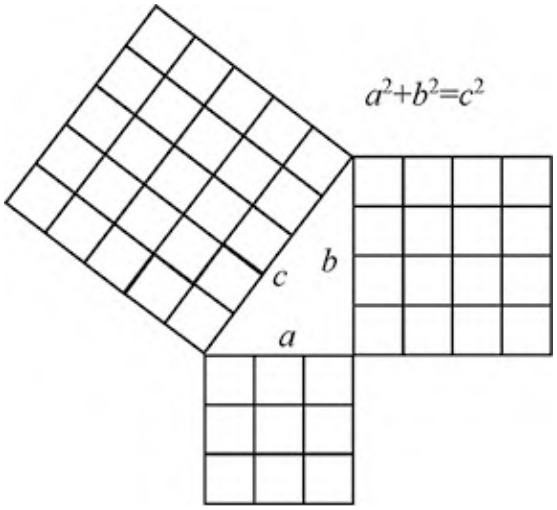


图2.4

输入直角三角形的两条直角边的长度，求斜边的长度。保证3条边的长度都为整数。

【输入格式】

输入两个不超过1 000的正整数，表示直角三角形的两条直角边的长度。

【输出格式】

输出一个整数，表示直角三角形的斜边的长度。

【输入样例】

3 4

【输出样例】

5

2.1.13　计算两点间的直线距离（distance）

【题目描述】

已知二维平面里两个点A、B的坐标分别为(x_1, y_1)和(x_2, y_2)，试计算A、B两点间的直线距离。已知A、B两点间的直线距离计算公式如下。

$$|AB| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

【输入格式】

输入4个数，分别为二维平面里两个点A、B的坐标值x_1、y_1、x_2、y_2。

【输出格式】

输出A、B两点间的直线距离（为防止出现浮点数误差，输出整数即可）。

【输入样例】

0 1 0 2

【输出样例】

1

2.1.14 海伦公式（helen）

【题目描述】

已知三角形3条边的长度a、b、c，求三角形的面积S，可以使用海伦公式。

设$p=\frac{a+b+c}{2}$，则$S\Delta ABC=\sqrt{p(p-a)(p-b)(p-c)}$。

试用海伦公式求三角形的面积（保证输入的a、b、c能构成三角形）。

【输入格式】

输入3个浮点数，表示三角形的3条边长。

【输出格式】

输出三角形的面积，小数点后保留两位小数。

【输入样例】

5 5 5

【输出样例】

area=10.83



计算p的值不能写成p=1/2*(a+b+c)，而应该写成p=1.0/2*(a+b+c)，或者p=(a+b+c)/2。为什么要用1.0而不是1呢，这是因为如果是1/2，系统将会视之为两个整数相除，得到的值也是一个整数，即1/2的结果舍弃小数部分后结果为0，而0乘以任何数都为0。

2.1.15　计算三角形的面积（helen2）

【题目描述】

三角形3个顶点的坐标分别为(x_1，y_1)、(x_2，y_2)、(x_3，y_3)，试计算三角形的面积。

【输入格式】

输入仅一行，包括6个浮点数，分别对应x_1、y_1、x_2、y_2、x_3、y_3。

【输出格式】

输出三角形的面积，精确到小数点后两位。

【输入样例】

0 0 4 0 0 4

【输出样例】

8.00

2.1.16 计算练习时间（training）

【题目描述】

琪儿努力练习编程，某天她从*a*时*b*分一直练习到当天的*c*时*d*分（24小时制，且*a*、*b*、*c*、*d*的值均用键盘输入），请你帮她计算一下，她一共练习了多少时间呢？



使用“内容市场”App扫描看视频

【输入格式】

输入4个整数，即a、b、c、d。

【输出格式】

输出两个整数，即练习时间的小时数和分钟数。

【输入样例】

12 50 19 10

【输出样例】

6 20

2.2 选择结构

选择结构通过判断某些特定条件是否满足来决定下一步的执行流程，例如，计算购物费用时，若总数sum＞1 000则打9折；a和b两个数比较时，若a＜b则输出a的值等。



扫码看视频

诸如sum＞1 000和a＜b这样的判断条件称为关系表达式，“＞”“＜”这样的符号称为关系运算符，因为它实现的不是算术运算而是关系运算。关系运算实际上是将两个数据进行比较，得到逻辑值“真”（true）或“假”（false），例如，3＜4的值为“假”，4＞0的值为“真”。C++语言以数字0代表“假”，以数字1或其他非0的数字代表“真”。

C++语言提供6种关系运算符，其含义和优先级如表2.1所示，关系运算符优先级的值越小，越优先运算。

表2.1

符号	含义	优先级	符号	含义	优先级

符号	含义	优先级	符号	含义	优先级
<	小于	6	>=	大于或等于	6
<=	小于或等于	6	==	等于	7
>	大于	6	!=	不等于	7

C++语言提供了3种形式的if语句来实现选择结构。

（1）if（表达式）　　//单分支语句

　　执行语句；

其流程如图2.5所示。

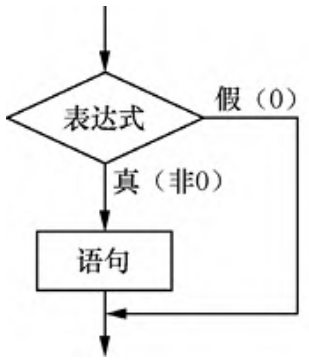


图2.5

类似的生活中的例子用伪代码描述如下。

```
if(天气好)
   我就去图书馆;
```

（2）if（表达式）　　//双分支语句

```
    执行语句1;
else
    执行语句2;
```

其流程如图2.6所示。

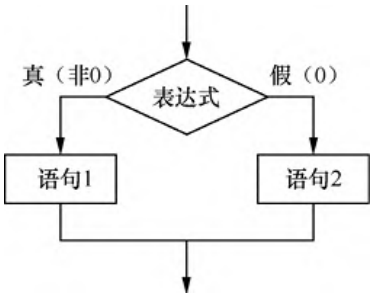


图2.6

类似的生活中的例子用伪代码描述如下。

```
if(天气好)
    我就去图书馆;
else
    我就待在家看书;
```

（3）if（表达式1）　　//多分支语句

```
    执行语句1;
  else if（表达式2）
    执行语句2;
  else if（表达式3）
    执行语句3;
  ……
  else if（表达式m）
    执行语句m;
```

```
else
    执行语句n;
```

其流程如图2.7所示。

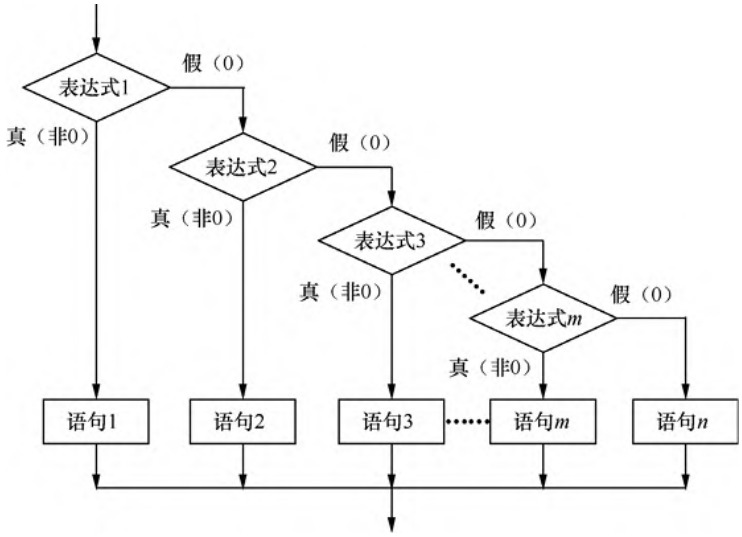


图2.7



类似的生活中的例子用伪代码描述如下。

```
if(我考了600分以上)
   我可以读国际顶级院校;
else if(我考了580分以上)
```

我可以读国内顶级院校;

else if(我考了550分以上)

我可以读国内一流院校;

else if(我考了500分以上)

我可以读国内二流院校;

else if(我考了450分以上)

我可以读国内三流院校;

else

人生豪迈，不过是从头再来，要不再读一年吧;

显然在这个例子中，最终有且只能有一个选择。此外，选择判断的先后逻辑顺序不能颠倒，如果写成下面这样，那就糟糕了。

```
if(我考了450分以上)                    //我考了800分,满足这个条件
  我可以读国内三流院校;                //于是上三流院校
else if(我考了500分以上)
  我可以读国内二流院校;
else if(我考了550分以上)
```

```
    我可以读国内一流院校;
else if(我考了580分以上)
    我可以读国内顶级院校;
else if(我考了600分以上)
    我可以读国际顶级院校;
else
    人生豪迈，不过是从头再来，要不再读一年吧;
```

2.2.1 求绝对值(abs)

【题目描述】

如图2.8所示，在数学中可以用一条直线上的点表示数，这条直线叫作数轴，它满足以下要求：（1）在直线上任取一个点表示0，这个点叫作原点；（2）通常规定直线上从原点向右（或上）为正方向，从原点向左（或下）为负方向；（3）选取适当的长度作为单位长度，直线上从原点向右，每隔一个单位长度取一个点，依次表示1 （向右1个单位长度）、2（向右2个单位长度）、3（向右3个单位长度）等；从

原点向左，用类似方法依次表示-1（向左1个单位长度）、-2（向左2个单位长度）等。

现输入数轴上的一个整数x，试求该整数到原点的距离。

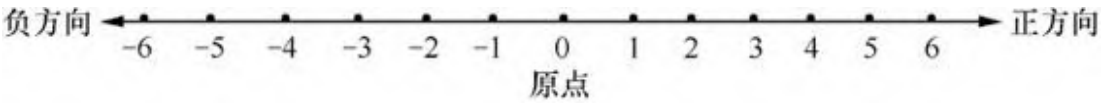


图2.8

【输入格式】

输入一个整数x。

【输出格式】

输出x的绝对值。

【输入样例1】

-1

【输出样例1】

1

【输入样例2】

2

【输出样例2】

2



可以使用单分支语句写核心代码：

```
if(x<0)
  x=-x;
cout<<x<<endl;
```

也可以使用双分支语句写核心代码：

```
if(x<0)
  cout<<-x<<endl;
else
  cout<<x<<endl;
```

还可以使用求绝对值的abs()函数，即cout<< abs(x) <<endl;。

此外，如果是对浮点数x取绝对值，则用fabs(x)函数。

2.2.2　奇偶数判断(odd)

【题目描述】

判断输入的整数是奇数还是偶数。

【输入格式】

输入一个整数。

【输出格式】

如果输入的整数是奇数，则输出“odd”，否则输出“even”。

【输入样例】

9

【输出样例】

odd

【笔试测验】请手动计算出程序运行的结果。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int n=9;
  if (n++<10)
    cout<<n<<endl;
  else
    cout<<n--<<endl;
  return 0;
}
```

【笔试测验】请手动计算出程序运行的结果。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  long a=7,b=18,c=23,d=320,sum;
  sum=a*55+b*14+c*12-d;
  sum+=497;
  sum%=593;
  if (sum%497==0)
    sum++;
  cout<<sum<<endl;
  return 0;
}
```

试找出下面程序的错误。

```
//输出星期几
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int n;
  cin>>n;
  if (n=1)  cout<<"星期一"<<endl;
  if (n=2)  cout<<"星期二"<<endl;
```

```
11    if (n=3)  cout<<"星期三"<<endl;
12    if (n=4)  cout<<"星期四"<<endl;
13    if (n=5)  cout<<"星期五"<<endl;
14    if (n=6)  cout<<"星期六"<<endl;
15    if (n=7)  cout<<"星期日"<<endl;
16    return 0;
17 }
```



对于n=1来说，它的意思是将1的值赋给n，所以这是一条赋值语句，执行完毕后的返回值为1，表示“真”输出“星期一”。其他的if语句同理，因此所有的输出语句均会被执行。改正方法是将所有“=”改为“= =”。

在if 和else后面只能含一条内嵌的执行语句，如果有多条内嵌的执行语句，就必须要用花括号标注，将多条内嵌的执行语句组合成为一个复合语句。例如：

```
if (x>y)
{
   number=0;         //语句1
   cout<<x<<endl;    //语句2
}
else
{
   number=1;         //语句1
   cout<<y<<endl;    //语句2
}
```



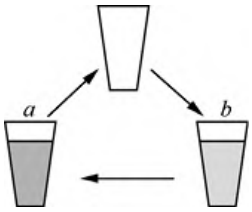
类似的生活中的例子用伪代码描述如下。

```
if(天气好)
{
   我就去买文具;
   我就去图书馆看书;
}
else
{
   我就在家写作业;
   我就在家看课外书;
}
```

2.2.3　两个实数排序(sort)

【题目描述】

实数包含有理数和无理数，其中无理数是无限不循环小数，有理数包含整数和分数。现输入两个实数*a*和*b*，通过交换将小的数放在*a*中、大的数放在*b*中后，按由小到大的顺序输出这两个实数。例如输入3 2，输出2 3；输入2 3，输出仍是2 3。



【输入格式】

输入2个无序的实数，以空格间隔。

【输出格式】

输出2个由小到大排好序的实数，以空格间隔。

【输入样例】

3 2

【输出样例】

2 3

根据题目要求，显然当$a>b$的时候，a和b的值要互换，但a和b的值不能直接互换，而是要通过一个中间变量，例如，用t来互换a和b的值。可以这样想象：假设有一瓶酱油和一瓶醋，如果要把酱油装入醋瓶，醋装入酱油瓶，就必须要有一个空瓶作为临时盛放酱油或醋的容器。中间变量t就相当于这个空瓶。

参考程序如下所示。

```
 1 //两个实数排序
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   double a,b,t;                              //假设a表示酱油瓶，b表示醋瓶，t表示空瓶
 8   cin>>a>>b;
 9   if (a>b)
10   {
11     t=a;                                     //将酱油瓶a中的酱油倒入空瓶t
12     a=b;                                     //将醋瓶b中的醋倒入酱油瓶a
13     b=t;                                     //将保存在空瓶t中的酱油倒入醋瓶b
14   }
15   cout<<a<<""<<b<<endl;
16   return 0;
17 }
```

2.2.4　3个实数排序(sort3)

【题目描述】

输入3个实数*a*、*b*、*c*，通过交换，将最小的数放在*a*中、次小的数放在*b*中、最大的数放在*c*中后，按由小到大的顺序输出这3个实数。

【输入格式】

输入3个无序的实数，以空格间隔。

【输出格式】

输出3个由小到大排好序的实数，以空格间隔。

【输入样例】

3 2 1

【输出样例】

1 2 3

伪代码如下所示，请根据伪代码的思路编写C++程序完成本题。

```
1 定义浮点数 a,b,c,t
2 输入a,b,c
3 如果(a>b)
4      a,b两值交换
5 如果(a>c)
6      a,c两值交换
7 如果(b>c)
8      b,c两值交换
9 输出a,b,c的值
```



注意代码中3个变量的比较顺序，即先比较a是否大于b，再比较a是否大于c，最后比较b是否大于c。如果调整比较顺序，结果是否仍然正确？

另外，C++有一个两数互换的函数，例如交换a和b的值，可以写成swap(a，b)。

2.2.5　判断两数的大小(compare)

【题目描述】

试编程判断用键盘输入的两个整数的大小。

【输入格式】

输入两个整数*a*和*b*。

【输出格式】

如果$a>b$，则输出$a>b$；如果$a=b$，则输出$a=b$；如果$a<b$，则输出$a<b$。

【输入样例】

3 4

【输出样例】

a＜b

使用多分支语句的参考程序如下所示。

```
1 //判断两数的大小
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
6 {
7   int a,b;
8   cin>>a>>b;
9   if (a>b)
10    cout<<"a>b"<<endl;
11  else if (a<b)
12    cout<<"a<b"<<endl;
13  else
14    cout<<"a=b"<<endl;
15  return 0;
16 }
```

2.2.6　等级制(grade)

【题目描述】

某学院的考试采用等级制，即将百分制分数转换成A、B、C、D、E这5个等级，设成绩为x，则$x \geqslant 90$为A等，$90>x \geqslant 80$为B等，$80>x \geqslant 70$为C等，$70>x \geqslant 60$为D等，否则为E等。试编写一个程序，将输入的分数转换成A、B、C、D、E这5个等级。

【输入格式】

输入一个浮点数。

【输出格式】

输出相应等级。

【输入样例】

100

【输出样例】

A

使用多分支语句的伪代码如下所示。

```
1 定义分数为浮点数x
2 输入x的值
3 如果(x>=90)
4       输出"A"
5 否则如果(x>=80)
```

```
 6       输出"B"
 7  否则如果(x>=70)
 8       输出"C"
 9  否则如果(x>=60)
10       输出"D"
11  否则
12       输出"E"
```

2.2.7　翻转硬币(coin)

【题目描述】

桌子上有4枚硬币，有些正面朝上，有些反面朝上。玩家每一次只能选其中的3枚硬币翻转，试求使之全部正面朝上或反面朝上的最少翻转次数。

【输入格式】

一行有4个数字，用0或者1表示每枚硬币的初始状态。

【输出格式】

输出一个数，表示最少翻转次数。

【输入样例】

1 0 1 1

【输出样例】

1

2.2.8　邮寄包裹(post)

【题目描述】

某邮局有规定：若包裹的质量超过30千克，则不予邮寄；对可以邮寄的包裹每件收取手续费0.2元，再收取根据表2.2按质量w计算出的费用。



使用“内容市场”App扫描看视频

表2.2

质量（千克）	收费标准（元/千克）
$w \leqslant 10$	0.80

质量（千克）	收费标准（元/千克）
$10<w \leqslant 20$	0.75
$20<w \leqslant 30$	0.70

请编写一个程序，输入包裹质量，输出所需费用，若无法邮寄，则输出“Fail”。

【输入格式】

输入一个数，表示包裹质量（千克）。

【输出格式】

输出所需总费用（保留小数点后两位），若无法邮寄，则输出“Fail”。

【输入样例】

18

【输出样例】

14.20



注意！请仔细观察样例，正确理解收费规定。

不要使用诸如if(10＜w＜=20)的语句来进行条件判断，因为它会先比较10＜w，得到的结果是0或者1，而0或1肯定小于等于20，所以最终判断的结果永远是“真”。

C++语言有一个可替代if…else语句的运算符，这个运算符被称为三目运算符，即“？：”，它是C++语言中唯一需要3个操作数的运算符。例如a＞b?c:d;，此句的含义是判断a是否大于b，如果a大于b，则取c的值，否则取d的值。

例如有语句Max=(a>b)?a:b；与下面的语句等价。

```
if (a>b)
      Max=a;
else
      Max=b;
```

在if 语句中包含一个或多个if语句称为if语句的嵌套，一般形式如下。

if ( )

```
    if ( ) 语句1
    else 语句2
else
    if ( ) 语句3
    else 语句4
```

应当注意if与else的配对关系，else总是与它上面的最近的未配对的if配对。就好像穿衣服一样，总是一层套一层。

可以通过代码缩进清晰地看出代码的逻辑关系。所谓代码缩进，是指在每一行的代码左端根据逻辑关系空出相应的位置。



类似的生活中的例子用伪代码描述如下。

```
if(天气好)
    if(是周末)
        我就逛街;
    else
```

我就看电影;

else

if(是周末)

我就看电视;

else

我就学习;

上例没有代码缩进的伪代码如下所示，很难看出代码内在的逻辑关系。

if(天气好)

if(是周末)

我就逛街;

else

我就看电影;

else

if(是周末)

我就看电视;

else

我就学习；

2.2.9 选班级代表(election)

【题目描述】

老师要从一班选择成绩大于90分的学生作为代表参加比赛，试输入学生的班级和成绩，

判断该学生是否符合要求，如果符合要求则输出“Yes”，否则输出“No”。

【输入格式】

输入两个数，分别表示学生的班级和成绩。

【输出格式】

判断该学生是否符合要求，如果符合要求则输出“Yes”，否则输出“No”。

【输入样例】

1 89

【输出样例】

No

参考程序如下所示，试找出程序的错误之处。

```
1 //选班级代表 —— 错误程序
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
6 {
7   int Class,Score;  //Class的首字母不能小写，因为Class是C++的关键字
8   cin>>Class>>Score;
9   if (Class==1)
10    if (Score>90)
11      cout<<"Yes\n";
12  else
13    cout<<"No\n";
14  return 0;
15 }
```



第9～13行代码的实际格式如下。

```
if(Class==1)
  if(Score＞90 )
    cout<<″Yes\n″;
```

```
    else
      cout<<"No\n";
```

因为else总是与它上面的最近的未配对的if配对，这样就只判断了1班学生的情况，而没有判断非1班学生的情况。

试判断下面修改后的核心代码是否正确?

```
if(Class==1)
   if(Score>90 )
     cout<<"Yes\n";
   else
     cout<<"No\n";
  else
    cout<<"No\n";
```

2.2.10　角色识别(role)

【题目描述】

输入一个人的年龄（整数）和性别（字符“M”或“F”，其中“M”代表男，“F”代表女），如果年龄未满18岁，则根据性别输出“Boy”或“Girl”，否则根据性别输出“Man”或“Woman”。

【输入格式】

输入一个人的年龄和性别。

【输出格式】

如果年龄未满18岁，则根据性别输出“Boy”或“Girl”，否则根据性别输出“Man”或“Woman”。

【输入样例】

15 M

【输出样例】

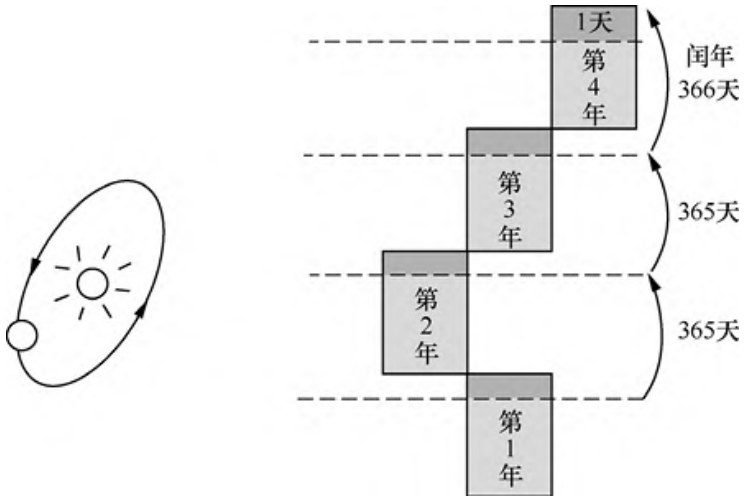
Boy

2.2.11　判断闰年(leap)

【题目描述】

判断为闰年的条件是符合下面二者之一：（1）能被4整除，但不能被100整除；（2）能被4整除，且能被400整除。因为地球绕太阳运行一周需365天5小时48分46秒，而公历的平年只有365天，所以每4年在2月末加1天，该年即闰年。但按照每4年一个闰年计算，平均每年就要多算出0.007 8天，400年就要多算出大约3天，故每过400年要减少3个闰年。公历规定：年份是整百数时，必须是400的倍数才是闰年；不是400的倍数的世纪年，即使其年份是4的倍数也不是闰年。即“四年一闰，百年不闰，四百年再闰”。

请编程判断输入的某数代表的年份是不是闰年。



【输入格式】

输入一个数，表示年份。

【输出格式】

如果是闰年，则输出“Y”，否则输出“N”。

【输入样例】

2001

【输出样例】

N

可能的核心伪代码如下所示（也可以自行设计程序结构）。

```
如果（年份能被400整除）
  输出"Y\n"
否则如果(年份不能被100整除)
{
  如果(年份能被4整除)
    输出"Y\n"
  否则
    输出"N\n"
```

}

否则

输出"N\n"

C++语言提供3种逻辑运算符，如表2.3所示。

表2.3

符号	含义
&&	逻辑与（相当于其他语言中的and，“并且”的意思）
||	逻辑或（相当于其他语言中的or，“或者”的意思）
!	逻辑非（相当于其他语言中的not，“非”的意思）

3种逻辑运算符对应的电路图如图2.9所示。其中a、b为开关，设开关闭合为“真”，开关断开为“假”；灯泡亮为“真”，灯泡灭为“假”。

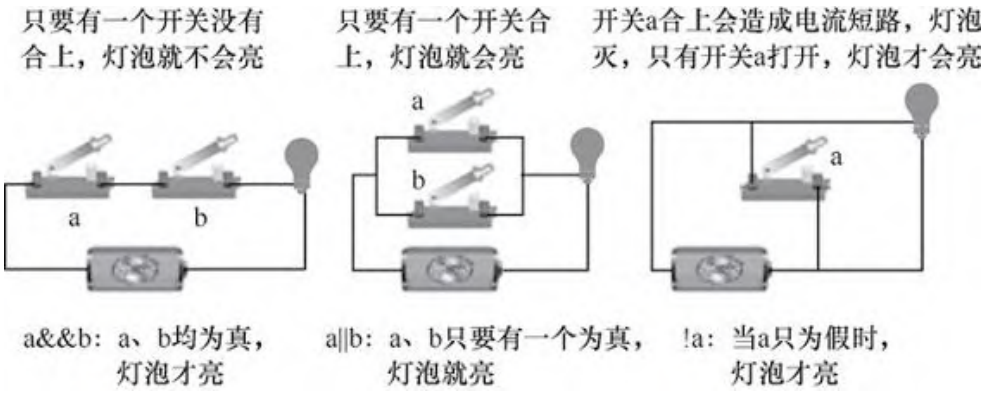


图2.9

逻辑运算举例如下。

a&&b：若a、b为真，则a&&b为真。

a‖b：　若a、b之一为真，则a‖b为真。

！a：若a为真，则！a为假。

一个逻辑表达式中如果包含多个逻辑运算符，例如！a&&b||x>y&&c，则运算优先顺序如下。

（1）！＞ && ＞‖。

（2）&&和‖的运算优先顺序低于关系运算符，！的运算优先顺序高于算术运算符。

例如：

(a>b)&&(x>y)可写成 a>b && x>y；

(a==b) || (x==y)可写成 a==b || x==y；

(!a) || (a>b)可写成 !a || a>b。

改写的判断闰年的代码如下所示。

```
 1 //判断闰年
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int year;
 8   cin>>year;
 9   if (year%400==0 || (year%4==0 && year%100!=0))
10     cout<<"Y\n";
11   else
12     cout<<"N\n";
13   return 0;
14 }
```

考虑到运行效率，在逻辑表达式的求解中，并不是所有的逻辑运算符都被执行，只有在必须执行下一个逻辑运算符才能求出逻辑表达式的解时，才执行该逻辑运算符。举例如下。

（1）执行语句a&&b&&c时，只有在a的值为真的情况下，才需要判断b的值；只有在a和b的值都为真的情况下，才需要判断c的值；只要a

的值为假，就不必判断b和c的值；如果a的值为真，b的值为假，则不必判断c的值。

（2）执行语句a‖b‖c时，只要a的值为真，就不必判断b和c的值；只有a的值为假，才判断b的值；只有a和b的值都为假，才判断c的值。

例如下例中输出的a、b、c的值分别为0、1、1。

```
 1 //逻辑表达式示例
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a=0,b=1,c=1;
 8   if (a && b++ && c++)       //因为a的值为假，所以b++和c++就不被执行
 9     b++;                     //此句也不会被执行
10   cout<<a<<b<<c<<endl;
11   return 0;
12 }
```

2.2.12　正方形内的点(point)

【题目描述】

有一个正方形，4个角的坐标分别是(1,-1)、(1,1)、（-1,-1）、(-1,1)。写一个程序，用键盘输入两个整数*x*、*y*，表示一个点的坐标，试判断该点（*x*，*y*）是否在这个正方形内（包括正方形

边界），如果在正方形内则输出“Yes”，否则输出“No”。

【输入格式】

输入两个整数x、y，表示一个点的坐标。

【输出格式】

如果该点在正方形内则输出“Yes”，否则输出“No”。

【输入样例】

0 0

【输出样例】

Yes

2.2.13　判断数的整除(div)

【题目描述】

输入一个整数，判断它能否被3、5、7整除，并输出以下信息。

（1）能同时被3、5、7整除，直接输出3 5 7，数与数之间以空格间隔。

（2）能同时被其中两个数整除，从小到大输出这两个数，两个数之间以空格间隔。

（3）只能被其中一个数整除，输出这个数。

（4）不能被任何数整除，输出“n”。

【输入格式】

输入一个整数。

【输出格式】

按题目要求，输出相应信息。

【输入样例】

5

【输出样例】

5

switch语句（开关语句）是多分支选择语句，用于实现多分支选择结构。例如，学生的成绩“A”等为85分以上，“B”等为70～84

分，“C”等为60～69分，“D”等为50～59分，其余的为“E”等，核心代码如下所示。

```
 1 char grade;
 2 cin>>grade;
 3 switch (grade)                          //比较grade值，grade值为一个字符
 4 {
 5   case 'A': cout<<"85~100分\n";      //若grade值为'A'
 6   case 'B': cout<<"70~84分\n";       //若grade值为'B'
 7   case 'C': cout<<"60~69分\n";       //若grade值为'C'
 8   case 'D': cout<<"50~59分\n";       //若grade值为'D'
 9   default:  cout<<"低于50分\n";      //否则为'E'
10 }
```

switch后面圆括号内的表达式可以为任何类型，例如上例中grade为字符型。

表达式的值必须与某一case后面的常量表达式的值完全匹配，才执行此case后面的语句，若所有的case后面的常量表达式的值都没有与表达式的值相匹配，就执行default后面的语句。

每一个case后面的常量表达式的值必须互不相同，否则会出现互相矛盾的现象。

每个case和default的出现次序不影响执行结果。



要注意的是：执行完一个case后面的语句后，流程控制转移到下一个case继续执行。“case常量表达式”只是起语句标号作用，并不是在该处进行条件判断。在执行switch语句时，根据switch后面表达式的值找到匹配的入口标号后，就从此标号开始执行下去，不再进行判断。例如上例中，若grade的值等于'A'，则将输出所有的字符串。

应该在执行一个case分支后，立即终止switch语句的执行，跳出switch结构，这可以通过break语句来实现。代码如下所示。

```
 1 char grade;
 2 cin>>grade;
 3 switch (grade)                          //比较grade值，grade值为一个字符
 4 {
 5   case 'A': cout<<"85~100分\n"; break;      //跳出switch结构
 6   case 'B': cout<<"70~84分\n";  break;      //跳出switch结构
 7   case 'C': cout<<"60~69分\n";  break;      //跳出switch结构
 8   case 'D': cout<<"50~59分\n";  break;      //跳出switch结构
 9   default:  cout<<"低于50分\n";             //最后一句可不加break语句
10 }
```



多个case可以共用一组执行语句，例如：

switch (grade)

```
{
    case 'A':                              //共用语句1
    case 'B':                              //共用语句1
    case 'C': cout<<"高于60分\n";break;     //语句1
    case 'D':                              //共用语句2
    default:cout<<"低于60分\n";   //语句2
}
```

即当grade的值为'A'、'B'、'C'时执行语句1，当grade的值非'A'、'B'、'C'时执行语句2。

【笔试测验】请手动算出下面程序执行的结果。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int n=3;
  switch (n)
  {
    case 3: n+=3;
    case 1: n++; break;
    case 5: n+=5;
    case 4: n+=4;
  }
  cout<<"n="<<n<<endl;
```

```
15    return 0;
16 }
```

2.2.14　五分制(five)

【题目描述】

学生的考试成绩实行五分制，即5分为最高分，现在输入一个学生的分数，如果是5分，则输出“A”；如果是4分，则输出“B”；如果是3分，则输出“C”；如果是2分，则输出“D”；如果是1分，则输出“E”。

【输入格式】

输入一个学生的五分制分数。

【输出格式】

根据题目要求输出结果。

【输入样例】

5

【输出样例】

A

参考代码如下所示。

```
 1 //五分制
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int grade;                          //注意此处是整数
 8   cin>>grade;
 9   switch (grade)                      //比较grade值，grade值为一个整
数
10   {
11     case 5:  cout<<"A\n"; break;
12     case 4:  cout<<"B\n"; break;
13     case 3:  cout<<"C\n"; break;
14     case 2:  cout<<"D\n"; break;
15     default: cout<<"E\n";             // 此处可以不加break语句
16   }
17   return 0;
18 }
```

2.2.15　等级划分(gradation)

【题目描述】

学生成绩分A、B、C、D、E共5个等级，现在输入一个学生的成绩等级，如果是A，则输出“Very good!”；如果是B，则输出“Good!”；如果是C，则输出“OK!”；如果是D，则输出“Bad!”；如果是E，则输出“Too bad!”。

【输入格式】

输入学生的成绩等级。

【输出格式】

输出对应的评价。

【输入样例】

A

【输出样例】

Very good!

参考代码如下所示。

```
 1 //等级划分
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   char grade;                          //注意此处是字符型
8   cin>>grade;
 9   switch(grade)                        //比较grade值，grade值为一个字
符
10   {
11     case 'A':  cout<<"Very good!\n";break;
12     case 'B':  cout<<"Good!\n";     break;
13     case 'C':  cout<<"OK!\n";       break;
14     case 'D':  cout<<"Bad!\n";      break;
15     default:   cout<<"Too bad!\n";  //此处可以不加break语句
16   }
17   return 0;
18 }
```

2.2.16　简单计算器(cal)

【题目描述】

设计一个支持“+”“-”“*”“/”4种运算的简单计算器。使用时用户需输入一个算式，算

式共有3个参数，其中第1和第3个参数为整数，第2个参数为运算符（“+”“-”“*”“/”）。

简单计算器将根据用户输入的算式输出结果。但是，

（1）如果出现除数为0的情况，则输出“Divided by zero!”；

（2）如果出现无效运算符（不为“+”　“-”　“*”　“/”之一），则输出“Invalid operator!”。

试用switch语句编程完成上述功能（禁止使用if语句）。

【输入格式】

输入一个算式，算式共有3个参数，参数之间以空格间隔。

【输出格式】

根据用户输入的算式输出结果，数据和运算结果不会超过整数的取值范围。

【输入样例】

1 + 1

【输出样例】

2

2.2.17 运费计算(freight)

【题目描述】

一批材料需要运输，已知运费计算公式为$f=p\times w\times s\times(1-d)$，其中$p$为运输物品价值，$w$为运输物品重量，$s$为运输物品距离，$d$为折扣。现用键盘输入$p$、$w$、$s$的值，而折扣$d$的计算遵守以下规则。

$s<250$	没有折扣
$250\leqslant s<500$	2%折扣
$500\leqslant s<1\,000$	5%折扣
$1\,000\leqslant s<2\,000$	8%折扣

$2\,000 \leqslant s<3\,000$　　10%折扣

$3\,000 \leqslant s$　　15%折扣

试计算运费f的值。注意，禁止使用if语句，只能用switch语句判断和计算折扣值。

【输入格式】

输入3个实数，即p、w、s。

【输出格式】

输出运费值，保留小数点后两位（无须四舍五入）。

【输入样例】

100 100 100

【输出样例】

1000000.00

【算法分析】

这个程序可以很容易地用if…else语句实现，但是如果改写为switch语句来实现会有点困难。因为case后面必须是常量表达式，所以写成case s＜250或者case(2000＜=s) &&(s＜3000)这样的形式是错误的。

但是也不可能写几千、几万个case，我们需要想办法将无限的数集映射为有限的数集。

观察折扣规则，可以发现折扣的变化点都是250的倍数，当s＜3000时，s/250的值为0～11。

2.2.18　某年某月有几天(day)

【题目描述】

输入年份和月份，试判断该月共有多少天。

【输入格式】

输入两个整数，表示年份和月份。

【输出格式】

输出一个整数，表示该月共有多少天。

【输入样例】

2004 1

【输出样例】

31

2.2.19 判断某日是一年的第几天(day)

【题目描述】

输入年份、月份、日期，判断这一天是这一年的第几天。

【输入格式】

输入3个整数，即年份、月份、日期，以空格间隔，保证输入的年份、日期是正确的。

【输出格式】

输出这一天是这一年的第几天，如果数据错误，则输出“data error!”。

【输入样例】

2016 1 1

【输出样例】

1

【算法分析】

一年有12个月，因此可以用case对应每个月。每个月的天数是一定的，因此可以按月将天数进行累加。例如1月5日，直接输出5即可。2月5日呢？输出31＋5的值即可……

此外还需要考虑到2月的天数计算，2月的天数可以先看成28，天数累加完毕后要判断这一年是闰年还是平年，如果是闰年并且输入月份大于2月，则输出结果需要加1。

2.3　循环结构



扫码看视频

使用循环结构的程序可以解决一些按一定规则重复执行的问题而无须重复书写代码，这是程序设计中最能发挥计算机特长的程序结构之一。

while语句用于实现“当型”循环结构，其一般形式如下。

```
while(表达式)
{
   循环语句;
}
```

即当表达式的值为真时，执行while语句中的内嵌循环语句。其流程如图2.10所示。

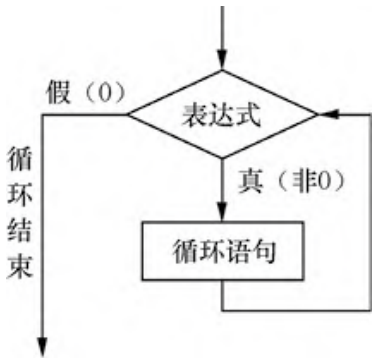


图2.10

例如计算1+2+3+…+100的值的程序如下。

```
1 //计算1+2+3+…+100的值
2 #include <bits/stdc++.h>
```

```
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int i=1,sum=0;
 8   while (i<=100)                    //当i<=100时，则执行循环体内的语句
 9   {
10     sum+=i;                         //累加1,2,3…到sum
11     i++;
12   }
13   cout<<sum<<endl;
14   return 0;
15 }
```

2.3.1　计算*n*的阶乘(factorial)

【题目描述】

自然数*n*的阶乘写作*n*!，例如15!＝1×2×3×4×…×15，试计算*n*的阶乘。

【输入格式】

输入一个整数*n*（1＜n＜21）。

【输出格式】

输出*n*!的结果。

【输入样例】

5

【输出样例】

120

【笔试测验】请手动算出程序的执行结果。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int p=1,d=5;
  while (d>1)
  {
    p=2*(p+1);
    d--;
  }
  cout<<p<<endl;
  return 0;
}
```

2.3.2　电文保密(message)

【题目描述】

电文保密的规律是将每个英文字母变成其后的第4个字母，例如A变成E、a变成e，最后的4个大写字母W、X、Y、Z分别变为A、B、C、D，小写字母w、x、y、z分别变为a、b、c、d，非字母字符不变。输入一行字符，要求输出相应的密文。

【输入格式】

输入一行字符。

【输出格式】

输出相应的密文。

【输入样例】

W

【输出样例】

A

参考程序如下所示。

```
 1 //电文保密
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   char c;
 8   while ((c=getchar())!='\n')          //getchar：从键盘获取一个字
符
 9   {
10     if ((c>='a' && c<='z') || (c>='A' && c<='Z')) //当字符为英文
字母时
11     {
12       c+=4;
13       if (c>'Z' && c<='Z'+4 || c>'z')//参照ASCII码对照表,考虑为何这
```

```
样写
14          c-=26;
15      }
16      cout<<c;
17    }
18    return 0;
19 }
```



第8行的语句并不是从终端输入一个字符马上就输出一个字符，而是一直输入直到按Enter键后将所有字符送入内存缓冲区，然后执行循环语句—每次从内存缓冲区读取一个字符加密后再输出。

2.3.3　求数的总和(power)

【题目描述】

用键盘输入*N*个数，试求全部数的总和。当用户输入0时，程序结束。

【输入格式】

输入*N*（*N*<100）个数，以空格间隔，最后以0结束。

【输出格式】

输出数的总和。

【输入样例】

1 2 3 0

【输出样例】

6

```
 1 //求数的总和
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int sum=0,n;
 8   while (true)            //表示永远为真，即永远循环，也可以用while(1)
 9   {
10     cout<<"输入一个整数（0表示结束）";//正式提交程序时不用写此句
11     cin>>n;
12     if (n==0)
13       break;             //跳出该层循环
14     sum+=n;
15   }
16   cout<<sum<<endl;
17   return 0;
18 }
```

2.3.4　整数猜想(guess)

【题目描述】

整数猜想是指对于任意给定的大于1的正整数，如果它是偶数则将其除以2，如果它是奇数则将其乘以3加1，其运算结果如果不是1，则重复上述操作。这样经过若干步操作后，总会得到结果1。

【输入格式】

输入一个大于1的正整数。

【输出格式】

输出该数运算过程中的数，数与数之间以空格间隔，最后一个数无空格，以换行结束。

【输入样例】

5

【输出样例】

5 16 8 4 2 1



由于事先无法知道要经过多少步操作才能得到结果1，因此用while语句是比较合适的。

接下来按题意模拟运算即可。

2.3.5 数字反转（rererse）

【题目描述】

给定一个整数n，请将该数各位上的数字反转得到一个新数。输出的新数不应有前导0。

【输入格式】

输入一个整数n（$-1\,000\,000\,000 \leqslant n \leqslant 1\,000\,000\,000$）。

【输出格式】

输出反转后的新数。

【输入样例】

-380

【输出样例】

-83

【算法分析】

因为不知道整数*n*的位数是多少，所以不知道需要多少步操作才可以把*n*的每一位数取出来，这种情况下用while语句是比较合适的。

当执行while语句时，每次取n最右边的一位数（n%10即可以实现）后删除这一位数（n/=10即可以实现），显然while语句结束的条件是n最后删除到只剩下0。

每次取出n的最右边的一位数后不要马上输出，这样无法解决前导0的问题，而是应该设一个整数变量例如T为最终要输出的反转后的新数，T初始时为0，每次取n的最右边的一位数，通过T＝T*10＋n%10的方式不断合并取出的数到T，最终T的值即答案。

2.3.6　级数求和（sum）

【题目描述】

已知：S_n=1+1/2+1/3+…+1/*n*。现用键盘输入一个整数*K*，要求输出一个最小的*n*，使得S_n＞*K*。

【输入格式】

输入一个整数K。

【输出格式】

输出一个最小的n，使得$S_n > K$。

【输入样例】

2

【输出样例】

4

初学者常常会因为没有注意到1除以整数的结果仍然是整数而导致结果错误。一个显然易见的调试方法是在循环语句中插入输出语句，观察程序运行时输出每次循环后的S_n的值即可迅速发现问题所在。

通过Dev-C++的调试功能可以方便地查看程序运行到某代码行时某变量值的变化。调试之前需先设置某代码行的断点，断点一般设置在要检查的程序段之前，如图2.11所示。

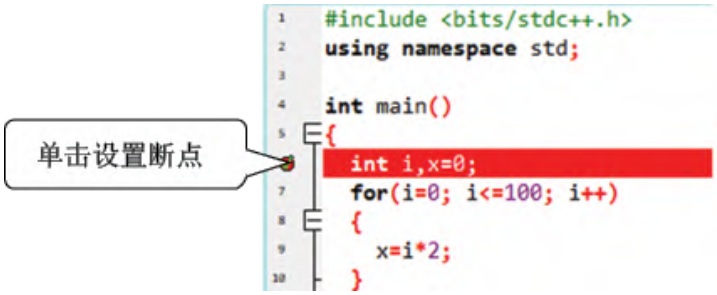


图2.11

单击功能区的“调试”选项卡，再单击“调试”按钮即进入调试模式（若进入调试模式成功，当前执行语句底色将变色），如图2.12所示。在此模式中，程序会运行到断点处停止，此时可单击“下一步”等按钮进行调试。



图2.12

若想动态地观察某个变量在程序运行中的变化，可单击“添加查看”按钮添加变量，如图2.13所示。

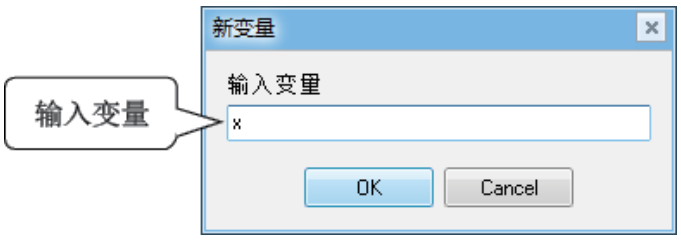


图2.13

在右侧的变量观察窗口，将显示该变量的值，并且其值随着程序的运行而发生相应变化，如图2.14所示。



图2.14

2.3.7　投资收益(investment)

【题目描述】

输入初始资金m、年利率r及期望收益W，试求要投资多少年，所有资金才会超过W?

【输入格式】

输入3个整数，即m（$100 \leqslant m \leqslant 1\,000\,000$）、$r$（$1 \leqslant r \leqslant 20$）及$W$。

【输出格式】

输出一个整数，即需要投资的年数(1～400)。

【输入样例】

5000 5 5788

【输出样例】

3



假设初始资金为5 000元，年利率为5%，则：

第1年后资金总额为1.05×5 000元＝5 250元；

第2年后资金总额为1.05×5 250元＝5 512.5元；

第3年后资金总额为1.05×5 512.5元＝5 788.125元；

……

2.3.8　找规律1（array）

【题目描述】

已知数列前6个数字为2、1、4、3、6、5，输入一个整数n，试输出数列前n项。

【输入格式】

输入一个整数n（$1 \leqslant n \leqslant 200$）。

【输出格式】

输出数列前n项（数与数之间有空格，行末无空格，有换行）。

【输入样例】

6

【输出样例】

2 1 4 3 6 5



因为n是用键盘输入的，所以必须控制while语句循环输出n项后就结束。可以在循环开始前设一个变量充当计数器，每循环一次，计数器自身加1，直到计数器等于n时，退出循环即可。

2.3.9　找规律2(array)

【题目描述】

已知数列前10个数字为1、1、1、2、3、4、6、9、13、19，现用键盘输入一个整数n，试找出规律输出该数列的前n项。

【输入格式】

输入一个整数n（$4 \leqslant n \leqslant 100$）。

【输出格式】

输出该数列的前n项（数与数之间有空格，行末无空格，有换行）。

【输入样例】

6

【输出样例】

1 1 1 2 3 4

2.3.10　求圆周率(pi)

【题目描述】

圆周率（π）为圆形的周长与直径之比。如图2.15所示，古代数学家祖冲之第一次将圆周率的值计算到小数点后7位。现用π/4=1-1/3+1/5-1/7+…

公式求π的近似值，直到某一项的绝对值小于10^{-6}为止。



图2.15

使用“内容市场”

App扫描看视频

【输入格式】

无输入。

【输出格式】

输出圆周率的值，保留小数点后5位（不用四舍五入）。

【输入样例】

无输入。

【输出样例】

略。



由公式π/4=1-1/3+1/5-1/7+…可推出π=(1-1/3+1/5-1/7+…)×4，那么1-1/3+1/5-1/7+…的值怎么计算呢？

可以设变量ans为输出的最终答案，初始值为0，使用while语句进行逐项累加直到某一项的绝对值小于10^{-6}为止。

但是累加每一项的时候，分母在不断地递增（每次多2），所以需要定义一个变量来保存分母的值。此外，每一项的正负号也在不断变换，这可能也需要一个变量来表示，例如s=-s即可完成正负号的转变。

伪代码如下所示。

```
 1 double n=1.0, s=1                  //n表示分母，初始值为1.0,s表示分子
 2 double t=1,ans=0                  //t为循环中要累加的每一项，第一项为1
 3 当(t的绝对值>1e－6)时          //1e－6即1乘以10的-6次方
 4 {
 5   ans=ans+t                       //累加
 6   更新下一项分母的值
 7   改变正负号
 8   更新t的值
 9 }
10 输出ans*4的值
```

2.3.11　求最大公约数和最小公倍数(euclid)

【题目描述】

输入两个正整数*a*和*b*，求其最大公约数和最小公倍数。

【输入格式】

输入两个正整数*a*和*b*。

【输出格式】

输出两个正整数，即*a*和*b*的最大公约数和最小公倍数，以空格间隔。

【输入样例】

153 123

【输出样例】

3 6273

【算法分析】

可以使用辗转相除法（欧几里得算法）来求最大公约数，即将大的数*a*除以小的数*b*，得余数*c*，设*a*=*b*、*b*=*c*，继续求余数，如此反复直到余数为0，此时的*b*为最大公约数。

例如a=24、b=10，则c=24%10=4；

由a=b、b=c得a=10、b=4，则c=10%4=2；

由a=b、b=c得a=4、b=2，则c=4%2=0，故最大公约数为b=2。

求最小公倍数的公式为初始数*a*×初始数*b*/最大公约数。

注意两数先乘会产生很大的数，可能会超过数据类型的取值范围，所以把计算顺序修改为初始数*a*/最大公约数×初始数*b*即可。

核心伪代码如下所示。

```
1 如果a<b
2    交换a和b的值                                    //考虑不交换是否可行？
3 当（b!=0）                                          //考虑为什么是b不等于0
4 {
5    c=a%b;
6    a=b;
7    b=c;
8 }
9 输出最大公约数和最小公倍数
```

2.3.12　求S_n的值(sn)

【题目描述】

有公式$S_n=a+aa+aaa+aaaa+\cdots$，其中a是一个数字。例如a=3、n=4时，S_n的值为3+33+333+3333。现用键盘输入两个整数a和n，试求S_n的值。

【输入格式】

输入两个整数a和n。

【输出格式】

输出S_n的值，保证输出结果不超过整数的取值范围。

【输入样例】

1 2

【输出样例】

12

2.3.13　找规律3(array)

【题目描述】

已知数列前7个数字为1、6、18、40、75、126、196，现用键盘输入一个整数n（$1 \leqslant n \leqslant 100$），试找出规律输出数列的前$n$项。

【输入格式】

输入一个整数n。

【输出格式】

输出前n项（数与数之间有空格，行末无空格，有换行）。

【输入样例】

4

【输出样例】

1 6 18 40

do…while语句的特点是先执行一次循环体内的内嵌循环语句，然后判断表达式的值是否为真，如果为真则继续执行循环，如果为假则循环结束。其一般形式如下。

```
do
{
    循环语句;
}while(表达式);              //注意行末有分号
```

其流程如图2.16所示。

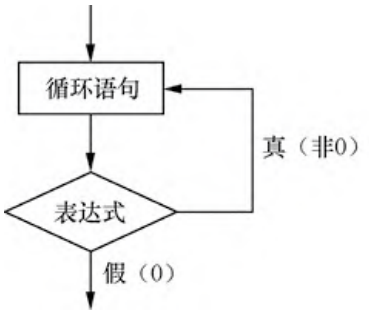


图2.16

例如计算1+2+3+…+100的值的程序如下。

```
 1 //计算1+2+3+…+100的值
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int i=1,sum=0;
 8   do
 9   {
10     sum+=i;
11     i++;
12   } while(i<=100);                                    //注意行末有分号
13   cout<<sum<<endl;
```

```
14    return 0;
15 }
```



while与do…while的区别如下。

while先判断表达式的值是否为真，如果为真则执行循环，如果为假则不执行循环。

do…while先执行一次循环体内的内嵌循环语句，之后就和while语句一样，判断表达式的值是否为真，以决定是否执行循环。

【笔试测验】请手动算出程序的执行结果。

```
 1 #include <bits/stdc++.h>
 2 using namespace std;
 3
 4 int main()
 5 {
 6   int m=9;
 7   do
 8   {
 9     cout<<(m-=2)<<" ";
10   } while(--m);
11   cout<<endl;
12   return 0;
13 }
```

2.3.14　计算皮球的弹跳高度(ball)

【题目描述】

输入一个整数h，表示皮球的初始高度，每次皮球落地后弹跳回原来高度的一半，编程计算皮球在第10次落地时，共经过多少米？第10次反弹有多高？

【输入格式】

输入一个整数h。

【输出格式】

输出两行数据，第一行数据为经过多少米，第二行数据为第10次反弹有多高，保留3位小数。

【输入样例】

10

【输出样例】

29.961

0.010

for语句是C++语言里使用最为灵活的语句之一，它可以代替while语句。其一般形式如下。

```
for（只执行一次的表达式1;循环条件的表达式2;循环过程的表达式3）
{
    循环语句;
}
```

它的执行过程如下。

（1）执行表达式1，该语句一般为循环变量赋初始值。

（2）执行表达式2，若其值为真即满足循环条件，则执行for循环体内的循环语句，然后执行第（3）步；若其值为假，则结束循环，转到第（5）步。

（3）执行表达式3，该语句一般为增加或减少循环变量，每次增加或减少的量也称为循环变量的步长。

（4）转到第（2）步继续执行。

（5）循环结束，执行for语句下面的语句。

例如计算1+2+3+…+100的值的程序如下。

```
 1 //计算1+2+3+…+100的值
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int sum=0,i;                  //在for循环体外定义循环变量i
 8   for (i=1; i<=100; i++)        //i赋初始值；i<=100为循环条件;i每次自
增1
```

```
 9      sum+=i;                          //sum=sum+i，即sum依次累加1,2,3…
10    cout<<sum<<endl;                  //若此时输出i值，则i值为101
11    return 0;
12 }
```

其流程如图2.17所示。

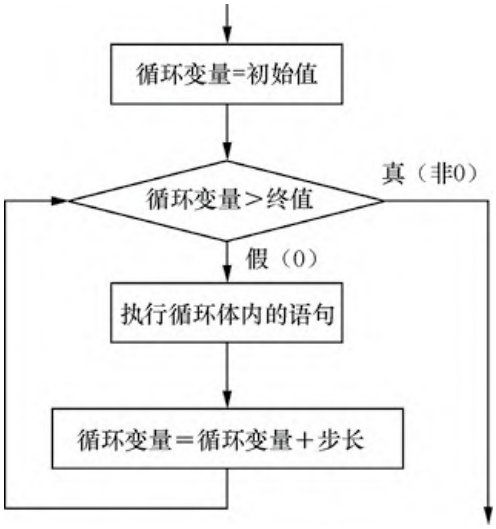


图2.17

for语句是很简单和方便的，使用该语句可以替代其他的循环语句。

表达式1、2、3可以根据实际情况省略（注意不可省略分号），但应另外设法保证循环能正常结束。以下几个程序和前文程序的结果是一样的。

程序1。

```
1 //计算1+2+3+…+100的值
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
```

```
 6 {
 7   int sum=0,i=1;                                    //此处循环变量i赋初始
值
 8   for (; i<=100; i++)
 9     sum+=i;
10   cout<<sum<<endl;
11   return 0;
12 }
```

程序2。

```
 1 //计算1+2+3+…+100的值
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int sum=0,i=1;
 8   for (; i<=100;)
 9   {
10     sum+=i;
11     i++;                                            //循环变量递增写在此处
12   }
13   cout<<sum<<endl;
14   return 0;
15 }
```

程序3。

```
 1 //计算1+2+3+…+100的值
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int sum=0,i=1;
 8   for(; ;)
 9   {
10     if (i>100)                                      //此处进行判断
11       break;
12     sum+=i;
13     i++;                                            //循环变量递增写在此处
14   }
15   cout<<sum<<endl;
```

```
16    return 0;
17 }
```

2.3.15　显示ASCII码值对应的字符(ASCII)

【题目描述】

目前计算机中广泛应用的字符集及其编码，是由美国国家标准协会制定的ASCII。请编写一个程序，显示ASCII码值为15～127的字符。

【输入格式】

无输入。

【输出格式】

显示的每个字符占用的空间为4个字符，右靠齐，每行输出10个字符后换行。

【输入样例】

无。

【输出样例】

略。

参考程序如下所示。

```
 1 //显示ASCII码值对应的字符
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int n=0;
 8   for (int i=15; i<128; i++) //循环体内定义i，则i只能在该循环体内使用
 9   {
10     cout<<setw(4)<<i<<":"<<char(i)<<" ";
11     if (!(++n%10))
12       cout<<endl;              //每显示10个字符换一行
13   }
14   cout<<endl;                  //for循环体外若想输出i的值会编译报错
15   return 0;
16 }
```

2.3.16 兔子“永生”（rabbit）

【题目描述】

如图2.18所示，兔子在出生两个月后就有繁殖能力。

第一个月小兔子没有繁殖能力，所以还是一对兔子；

两个月后，生下一对小兔子后总共有两对兔子；

3个月后，老兔子又生下一对小兔子，因为小兔子还没有繁殖能力，所以总共是3对兔子

……

假设所有兔子不死，那么40个月后可以繁殖多少对兔子？

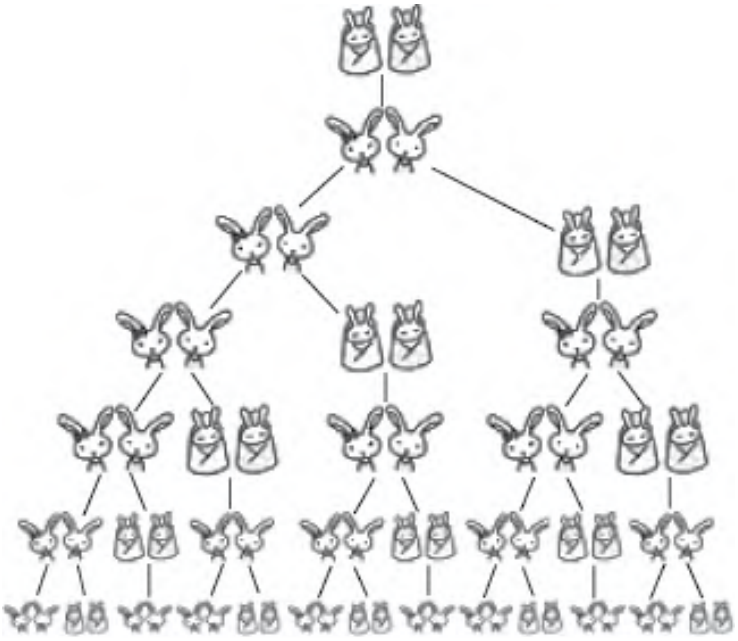


图2.18

【输入格式】

无输入。

【输出格式】

输出40个数，每个数后面以一个空格间隔(行末无空格)，每4个数占一行。

【输入样例】

无。

【输出样例】

1 1 2 3

5 8 13 21

……



这就是有名的斐波那契数列（Fibonacci Sequence），其特点为第1、2个数为1，从第3个数开始，该数是其前面两个数之和，即1,1,2,3,5,8,13,…。

参考程序如下所示。

```
1 //兔子"永生"
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
6 {
7   int f1=1,f2=1,i;
8   for (i=1; i<=20; i++)
9   {
10    cout<<f1<<' '<<f2<<' ';
11    if (i%2==0)   //当一行输出两次f1和f2（4个数）的值后另起一行
12      cout<<'\n';
```

```
13      f1=f1+f2;       //滚动向前计算
14      f2=f2+f1;
15    }
16    return 0;
17 }
```

2.3.17　计算平均分(average)

【题目描述】

输入一个整数*n*，表示有*n*个学生，随后输入这*n*个学生的成绩，试计算平均分。

【输入格式】

输入一个整数*n*和*n*个学生的成绩。

【输出格式】

输出结果即平均分，保留小数点后两位。

【输入样例】

2

100 100

【输出样例】

100.00

2.3.18　计算数列和(drug)

【题目描述】

试编程计算$1^2+2^2+3^2+\cdots+N^2$的和。

【输入格式】

输入一个数N，N不超过51。

【输出格式】

输出N个数的平方和。

【输入样例】

2

【输出样例】

5

2.3.19　奇数求和(odd)

【题目描述】

输入两个整数*n*和*m*，试计算*n*和*m*之间所有的奇数的和。例如*n*=3、*m*=8，则奇数和为3+5+7=15。

【输入格式】

输入两个整数*n*和*m*。

【输出格式】

输出一个整数，即*n*和*m*之间所有的奇数的和。

【输入样例】

3 8

【输出样例】

15

2.3.20　求水仙花数(flower)

【题目描述】

“水仙花数”是指一个三位数，其各位数字的立方和等于该数本身。

例如153＝1^3+5^3+3^3，153是一个水仙花数。请输出1 000以内所有的水仙花数。

【输入格式】

无输入。

【输出格式】

输出1 000以内所有的水仙花数，每个数占一行。

【输入样例】

无。

【输出样例】

略。

2.3.21 求最大跨度值(Max)

【题目描述】

输入一个整数*n*，表示有*n*个整数，随后输入*n*个整数的值，计算这*n*个整数的最大跨度值（最大跨度值＝最大值-最小值）。

【输入格式】

第一行为一个整数*n*，第二行为*n*个整数的值。

【输出格式】

输出这*n*个整数的最大跨度值。

【输入样例】

10

1 2 3 4 5 6 7 8 9 10

【输出样例】

9

2.3.22　求*e*的近似值(e)

【题目描述】

已知公式e=1+1/1!+1/2!+1/3!+…，试计算前100项相加的结果。

【输入格式】

无输入。

【输出格式】

输出e的近似值，保留小数点后3位（无须四舍五入）。

【输入样例】

无。

【输出样例】

2.7??

【样例说明】

一个“?”代表一位数，请将“?”转为具体的数值输出。

【算法分析】

计算阶乘可以利用前一项已算好的值递推出下一项的值，例如当计算出5!后，计算6!即6×5!。

假设某项值为t=1/n!，则下一项为1/n!/(n+1)，即t/(n+1)。

2.3.23　求分子序列和(sum)

【题目描述】

有分子序列2/1,3/2,5/3,8/5,13/8,21/13,…，试求该数列的前N项之和。

【输入格式】

输入一个整数N（N<100），表示求前N项之和。

【输出格式】

输出这个数列的前N项之和，保留小数点后两位（无须四舍五入）。

【输入样例】

2

【输出样例】

3.50

伪代码如下所示。

```
1 定义浮点数a＝2,b＝1,ans＝0                    //其中a为分子，b为分母
2 for(循环20次)
3 {
4   ans=ans+a/b
5   更新分子a和分母b的值
6 }
7 输出ans的值
```

2.3.24　计算圆面积(square)

【题目描述】

试依次计算半径r=1到r=10时的圆面积，直到面积$area$大于100为止。

【输入格式】

无输入。

【输出格式】

依次输出圆面积，格式见输出样例，保留小数点后两位（无须四舍五入）。

【输入样例】

无。

【输出样例】

```
1:area=3.14
2:area=12.56
……
```

参考程序如下所示。

```
 1 //计算圆面积
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   for (int r=1;r<=10;r++)
 8   {
 9     double area=3.14*r*r;
10     if (area>100)
11       break;                    //跳出当前for循环体，即彻底结束循环
12     cout<<r<<":area="<<setprecision(2)<<fixed<<area<<endl;
13   }
14   return 0;
15 }
```



break语句可以用来从最近的循环体内跳出，即提前结束当前循环。break语句不能用于循环语句和switch语句之外的任何其他语句中。

2.3.25　不能被3整除的数(sacred)

【题目描述】

试将100～200不能被3整除的数输出。

【输入格式】

无输入。

【输出格式】

输出符合题意的数，两数之间以一个空格间隔，最后一个数后面无空格，有换行。

【输入样例】

无。

【输出样例】

略。

参考程序如下所示。

```
 1 //将100～200不能被3整除的数输出
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 8   for (int n=100;n<=200;n++)
 9   {
10     if (n%3==0)
11       continue;            //强行结束本次循环，后面的循环还可能继续
12     cout<<n<<' ';          //请自行完善输出格式
13   }
14   return 0;
15 }
```



continue语句与break语句的区别是：continue语句只强行结束本次循环，接着执行下一次循环（如果循环条件仍然满足），而不是结束整个循环；break语句则是彻底结束整个循环。

2.3.26　火柴游戏

【题目描述】

有21根火柴，两个人轮流取，每人可取1～4根，不可多取，也不可不取，谁取最后一根谁输。假设小光先取，琪儿后取，试编程让琪

儿赢。



这是一道简单的博弈题，取胜策略：只需让后取者取的火柴数量与先取者取的火柴数量之和等于5即可。

伪代码如下所示。

```
设变量num=21，即火柴数
设变量man表示小光取的火柴数
while(游戏未结束时)          //想想游戏结束的条件是什么？应该怎么编写代码？
{
    输出现在的火柴数   //用于提示玩家的信息
    输入本次小光取的火柴数即变量man
    如果小光取的火柴数违反规定
      continue                //无效重取，即本次循环无效，继续下一轮循环
    否则输出琪儿取的火柴数（即5-man）和剩余的火柴数（即num-5）
    更新火柴数
}
输出"你输了!"
```

2.3.27　九九乘法表1

【题目描述】

九九乘法表又称九九乘法歌诀，在《荀子》等古书中就能找到“三九二十七”“六八四十八”“四八三十二”“六六三十六”等与

九九乘法表相关的句子。

试编程输出九九乘法表。

【输入格式】

无输入。

【输出格式】

见输出样例。

【输入样例】

无。

【输出样例】

1*1=1

1*2=2

1*3=3

1*4=4

1*5=5

1*6=6

1*7=7

1*8=8

1*9=9

2*1=2

2*2=4

……

9*9=81



这道题需要两层for循环语句嵌套完成，具体如下。

```
for(int i=1;i<=9;i++)
    for(int j=1;j<=9;j++)
        cout<<i<<'*'<<j<<"="<<i*j<<' ';
```

注意两层for循环语句的循环变量应分别定义为不同的变量，例如定义循环变量为i和j。其执行顺序是执行外循环到内循环时，要先把

内循环的所有循环执行完后再转换到外循环。

```
 1  //九九乘法表1
 2  #include <bits/stdc++.h>
 3  using namespace std;
 4
 5  int main()
 6  {
 7    for (int i=1;i<=9;i++)   //循环变量i和j可以在for循环体内直接定义
 8      for (int j=1;j<=9;j++) //作用范围仅在for循环体内，循环结束即"消
失"
 9        cout<<i<<'*'<<j<<"="<<i*j<<endl;
10   return 0;
11 }
```

2.3.28　九九乘法表2(99)

【题目描述】

试使用两层for循环语句输出如下的九九乘法表。

1*1=1

1*2=2　2*2=4

1*3=3　2*3=6　3*3=9

1*4=4　2*4=8　3*4=12　4*4=16

1*5=5　2*5=10　3*5=15　4*5=20　5*5=25

1*6=6　2*6=12　3*6=18　4*6=24　5*6=30　6*6=36

1*7=7　2*7=14　3*7=21　4*7=28　5*7=35　6*7=42　7*7=49

1*8=8　2*8=16　3*8=24　4*8=32　5*8=40　6*8=48　7*8=56　8*8=64

1*9=9　2*9=18　3*9=27　4*9=36　5*9=45　6*9=54　7*9=63　8*9=72　9*9=81

【输入格式】

无输入。

【输出格式】

输出格式见题目描述，注意为了保持格式整齐，乘积值输出宽度为4，左靠齐。

【输入样例】

无。

【输出样例】

略。



程序只能从上往下逐行输出，可以用外循环控制行数，内循环控制每行输出的式子数。

根据第1行输出1个式子，第2行输出2个式子，第3行输出3个式子……第9行输出9个式子，现在知道外循环和内循环相关语句应该怎么写了吧。

2.3.29　执行任务

【题目描述】

在A、B、C、D、E、F这6个人中尽可能多地挑选若干人去执行任务，限制条件如下：

（1）A和B两个人中至少去一人；

（2）A和D不能一起去；

（3）A、E及F这3人中要派两个人去；

（4）B和C都去或都不去；

（5）C和D两个人中去一人；

（6）若D不去，则E也不去。

问应当让哪几个人去？

【输入格式】

无输入。

【输出格式】

去的人以1表示，不去的人以0表示。例如a、b、c去，d、e、f不去，则输出格式如下。

```
a:1
b:1
c:1
d:0
e:0
f:0
```

【输入样例】

无。

【输出样例】

略。

用a、b、c、d、e、f这6个变量表示6个人是否去执行任务的状态，变量的值为1，表示此人去；变量的值为0，表示此人不去。由题意可写出如下表达式。

```
a+b>=1          //A和B两个人中至少去一人

a+d!=2          //A和D不能一起去

a+e+f==2          //A、E及F这3人中要派两个人去

b+c==0或b+c==2          //B和C都去或都不去

c+d==1          //C和D两个人中去一人

d+e==0或d==1          //若D不去，则E也不去(都不去，或D去E随便)
```

上述各表达式之间的关系为“与”关系。枚举每个人去或不去的各种可能情况，代入上述表达式中进行推理运算，使上述表达式均为“真”就是正确的结果。

参考程序如下所示。

```
 1 //执行任务
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   for (int a=1; a>=0; a--)                //枚举每个人是否去的所有情况
 8     for (int b=1; b>=0; b--)              //1表示去，0表示不去
9       for (int c=1; c>=0; c--)
10         for (int d=1; d>=0; d--)
11           for (int e=1; e>=0; e--)
12             for (int f=1; f>=0; f--)
13               if (a+b>=1 && a+d!=2 && a+e+f==2 &&
14                   (b+c==0||b+c==2) && c+d==1 && (d+e==0||d==1))
15               {
16                 cout<<"a:"<<a<<endl;
17                 cout<<"b:"<<b<<endl;
18                 cout<<"c:"<<c<<endl;
19                 cout<<"d:"<<d<<endl;
20                 cout<<"e:"<<e<<endl;
21                 cout<<"f:"<<f<<endl;
22               }
23   return 0;
24 }
```

2.3.30　谁是小偷(thief)

【题目描述】

A、B、C、D这4个人中有一个人是小偷，已知4个人中有一个人说了假话，请根据4个人的供词编程来判断谁是小偷。4个人的供词如下。

A：我不是小偷。

B：C是小偷。

C：D是小偷。

D：我不是小偷。



使用“内容市场”App扫描看视频

【输入格式】

无输入。

【输出格式】

输出谁是小偷，若A是小偷，则输出“A”。

【输入样例】

无。

【输出样例】

略。

2.3.31　绘制矩形（draw）

【题目描述】

输入两个整数*a*、*b*和一个字符*c*，*a*和*b*代表矩形的宽度和高度，*c*代表绘制矩形的符号，输出一个由字符*c*围起来的空心矩形。

【输入格式】

输入两个整数*a*、*b*和一个字符*c*。

【输出格式】

输出一个由字符*c*围起来的空心矩形。

【输入样例】

5 4 *

【输出样例】

```
     *****
*    *
*    *
*****
```

2.3.32　不定方程的解(equation)

【题目描述】

输入正整数a、b、c，满足不定方程$ax+by=c$，试求关于未知数x和y的所有非负整数解有多少个。

【输入格式】

输入正整数a、b、c。

【输出格式】

输出关于未知数x和y的所有非负整数解有多少个。

【输入样例】

2 3 18

【输出样例】

4

2.3.33　1的次数(sum1)

【题目描述】

输入一个正整数n，试计算1～n的所有整数中，出现1的次数为多少？例如当n=12时，有1、2、3、4、5、6、7、8、9、10、11、12，一共出现了5个1。

【输入格式】

输入一个正整数n（$1 \leqslant n \leqslant 10000$）。

【输出格式】

输出一个整数，即出现1的次数。

【输入样例】

12

【输出样例】

5

2.3.34　算式成立1（formula）

【题目描述】

试编程求出a、b、c、d的值各为多少时，下面的算式成立。

$$\begin{array}{r} a\ b\ c\ d \\ \times \qquad 9 \\ \hline d\ c\ b\ a \end{array}$$

【输入格式】

无输入。

【输出格式】

输出一行4个数，即a、b、c、d的值，数与数之间以空格间隔，末尾无空格，有换行。

【输入样例】

无。

【输出样例】

略。

2.3.35　算式成立2（formula2）

【题目描述】

下面的乘法竖式中，每个汉字代表一个数字，不同的汉字代表不同的数字，请编程计算出每

个汉字代表的数字是多少。

学习再学习

×　　学

优优优优优优

【输入格式】

无输入。

【输出格式】

输出4行，输出格式参见输出样例。

【输入样例】

无。

【输出样例】

学:?

习:?

再:?

优:?

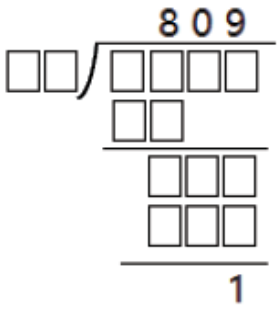
【样例说明】

“?”代表一个数字，请将“?”换成具体的数字输出。

2.3.36　算式成立3（formula3）

【题目描述】

试编程算出下面算式中的被除数和除数。



【输入格式】

无输入。

【输出格式】

输出两个整数，即被除数和除数。

【输入样例】

无。

【输出样例】

略。

设x是被除数、y是除数，则x的取值范围为1 000～9 999、y的取值范围为10～99。

枚举x和y，如果x/y=809并且x%y=1，则x和y就有可能是我们想要的答案。但是程序实际运行时会输出很多无效的答案，所以还需要再仔细观察算式，找出更多的限制条件，例如9*y＞=100……

2.3.37　同构数(num)

【题目描述】

正整数*n*若是它的平方数的尾部，则称*n*为同构数。例如5是25右边的数，25是625右边的数，5和25都是同构数。请找出2～10 000的全部同构数。

【输入格式】

无输入。

【输出格式】

输出2～10 000的全部同构数，从小到大依次排列，每行一个数。

【输入样例】

略。

【输出样例】

5

6

……

伪代码如下所示。

```
 1  枚举2～10000的每一个数i
 2  {
 3    s=i*i                          //i的平方数赋值给s
 4    j=i                            //i用于执行循环，不能修改，所以赋值给j
 5    当(s与j的末位数的值相等)   //循环,从个位开始，依次比较两数对应位的值
 6    {
 7       j=j/10                      //消去最末位的数
 8       s=s/10                      //消去最末位的数
 9    }
10    如果(j==0)
11       则i是同构数，输出
12  }
```

2.3.38　判断质数（prime）

【题目描述】

质数又称素数，是指在一个大于1的自然数中，除了1和此整数自身外，无法被其他自然数整除的数。比1大但不是质数的数称为合数。1和0既非质数也非合数。合数是由若干个质数相乘而得到的。所以，质数是合数的基础，没有质数就没有合数。

用键盘输入一个整数，试判断该数是不是质数。

100以内的质数

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	12	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

【输入格式】

输入一个整数。

【输出格式】

若该整数为质数则输出“Yes”，否则输出“No”。

【输入样例】

2

【输出样例】

Yes

```
 1 //判断质数
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int number,i;
 8   cin>>number;
 9   int k=sqrt(number);        //k为输入的数的平方根,想一下为什么
10   for (i=2; i<=k; i++)       //枚举数2～k
11     if (number%i==0)
12       break;                 //只要能被整除，就不是质数，跳出for循环
13   cout<<(i>k?"Yes\n":"No\n");
14   return 0;
15 }
```

数学家欧几里得在《几何原本》中证明了质数有无穷多个。

假设只有有限个质数P_1，P_2，P_3，…，P_n，令$N=P_1\times P_2\times P_3\times\cdots\times P_n$，那么，$N+1$是质数或者是合数。

如果$N+1$是质数，则$N+1$要大于P_1，P_2，P_3，…，P_n，所以它不在假设的质数集合中。

如果$N+1$是合数，因为任何一个合数都可以分解为几个质数的积，而N和$N+1$的最大公约数是1，$N+1$不可能被P_1，P_2，P_3，…，P_n整除，所以该合数分解得到的质因数肯定不在假设的质数集合中。

因此无论该数是质数还是合数，都意味着在假设的有限个质数之外还存在着其他质数。

所以原先的假设不成立。也就是说，质数有无穷多个。

2.3.39 质因数分解(factor)

【题目描述】

已知输入的正整数n是两个不同质数的乘积，试求出较大的质数。

【输入格式】

输入一个正整数n（$6 \leqslant n \leqslant 200\,000\,000$）。

【输出格式】

输出一个数，即较大的质数。

【输入样例】

21

【输出样例】

7

2.3.40　完美数(perfect)

【题目描述】

一个数如果恰好等于它的因子之和，则被称为“完美数”。例如6的因子为1、2、3，而6=1+2+3，因此6是完美数。目前发现的完美数都以6或28结尾。试编程找出1 000以内的所有完美数。

【输入格式】

无输入。

【输出格式】

每个完美数占一行，后面为冒号，由小到大输出它的因子，两数间以空格间隔，末尾无空格。

【输入样例】

无。

【输出样例】

6:1 2 3

……

2.3.41 防护罩(safe)

【题目描述】

如图2.19所示，所有街区整齐地排列，均为边长为1千米的正方形，半径为r千米的圆形防护罩以中心4个街区的交点为圆心，试计算圆形防护罩所能保护的完整街区数N。

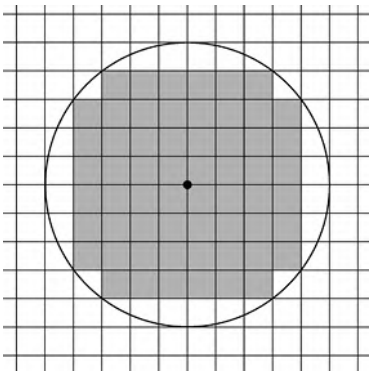


图2.19

【输入格式】

输入一个单精度浮点数r（$\sqrt{2} \leqslant r \leqslant 1\,000$）。

【输出格式】

输出M的值。

【输入样例】

5

【输出样例】

60

【算法分析】

由于圆的对称性，因此只需要计算1/4圆中所包含的完整街区数*N*。将其中所包含的完整街区数以纵列划分为组，设*K*表示1/4圆中共有*K*组完整街区，显然*K*为不超过半径*r*的最大整数，如图2.20所示。

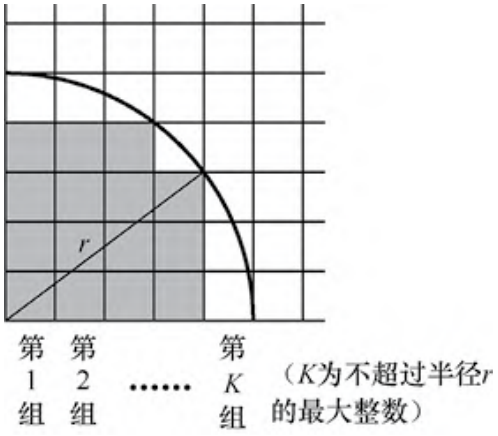


图2.20

则*N*=4×（第1组内的完整街区数＋第2组内的完整街区数＋…＋第*K*组内的完整街区数），但每组内的完整街区数如何计算呢？

以第4组为例，作一直角三角形如图2.21所示，可知斜边为*r*，底边为组数。根据勾股定理，当组数为*a*时，则第*a*组内的完整街区数为不大于$\sqrt{r^2-a^2}$的整数。

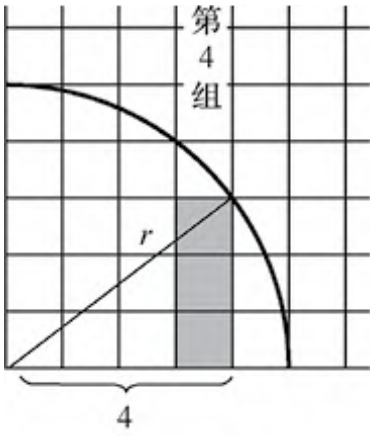


图2.21



题目中需要用到对实数取整的操作，除用int()强制取整的方法外，C++语言还有如下几个取整函数：

（1）floor()为向下取整函数，会取不大于自变量的最大整数，例如floor(3.1)=floor(3.9)=3，floor(-5.1)=floor(-5.9)=-6；

（2）ceil()为向上取整函数，会取不小于自变量的最大整数，例如ceil(3.1)=ceil(3.9)=4，
ceil(-2.1)=ceil(-2.9)=-2；

（3）round()为四舍五入函数，它会返回与自变量最接近的整数，例如round(10.5)=11，round(10.4)=10。

2.3.42　换零钱1（money）

【题目描述】

把100元换成10元、5元、1元的零钱，求每种零钱都至少各有一张的情况下，共有多少种兑换方案？

设100元可以换10元x张、5元y张、1元z张，则有：

100=10x+5y+z（$x \geqslant 1$，$y \geqslant 1$，$z \geqslant 1$）

初步分析上面的等式可以发现：$1 \leqslant x \leqslant 9$，$1 \leqslant y \leqslant 19$，$1 \leqslant z \leqslant 99$。

进一步优化可以发现：因为$x \geqslant 1$、$z \geqslant 1$，所以5y的取值范围变成$5 \leqslant 5y \leqslant 100-10\times1-1\times1$，解得$1 \leqslant y \leqslant 17$ ；同理，z的取值范围为$1 \leqslant z \leqslant 100-10\times1-5\times1$，解得$1 \leqslant z \leqslant 85$。这样使得循环体执行的次数由原来的9×19×99=16 929次缩小到9×17×85=13 005次，循环次数减少了3 924次。

```
 1 //换零钱 —— 基本算法
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int Count=0;
 8   for (int x=1; x<=9; x++)
 9     for (int y=1; y<=17; y++)
10       for (int z=1; z<=85; z++)
11         if ((10*x+5*y+z)==100)
12           Count++;
13   cout<<Count<<endl;
14   return 0;
15 }
```



还可以利用不等式减少循环次数。

在已经确定10元有*x*张、5元有*y*张的情况下，其实无须知道1元的张数，只要剩下的钱数大于0就可以组成一个方案，这样只需二重循环，即共循环9×17=153次，效率比之前提高了80多倍。

```
 1 //换零钱 —— 利用不等式优化算法
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int Count=0;
 8   for (int x=1; x<=9; x++)
 9     for (int y=1; y<=17; y++)
10       if (10*x+5*y<100)
11         Count++;
12   cout<<Count<<endl;
13   return 0;
14 }
```



取1张10元的，则剩下的90元可以用1~17张5元和若干张1元相加得到，故共有17种方案；

取2张10元的，则剩下的80元可以用1~15张5元和若干张1元相加得到，故共有15种方案；

……

取8张10元的，则剩下的20元可以用1~3张5元和若干张1元相加得到，故共有3种方案；

取9张10元的，则剩下的10元可以用1张5元和5张1元相加得到，故共有1种方案。

综上可得，总方案数为1+3+5+…+17=81（种），则问题就变成了求9个数的和，用一重循环运算9次即可，效率比用不等式运算提高了17倍。

```
 1 //换零钱 —— 利用排列组合优化
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int i,Count=0;
 8   for (i=1; i<=9; i++)
 9     Count+=2*i-1;
10   cout<<Count<<endl;
11   return 0;
12 }
```



观察数字序列1, 3, 5, 7, 9, 11, …，可以发现相邻两个数的差恒为2，这显然是等差数列。记第一项为a_1，相邻两个数之间的差为公差，

记为d，记第n项为a_n，容易推导出$n=(a_n-a_1)\div d+1$和$a_n=a_1+(n-1)\times d$。

求前n项和的公式如下。

$$S_n=((a_1+a_n)\times n)/2 \quad (2\text{-}1)$$

$$S_n=na_1+(n\times(n-1))/2\times d \quad (2\text{-}2)$$

$$S_n=na_n-(n\times(n-1))/2\times d \quad (2\text{-}3)$$

公式（2-1）用于求知道首项、末项及项数的等差数列的前n项和。

公式（2-2）用于求知道首项、公差及项数的等差数列的前n项和。

公式（2-3）用于求知道末项、公差及项数的等差数列的前n项和。

所以程序无须循环，直接用任何一个公式就可求出和为81。

```
1 //换零钱 —— 利用等差数列求和公式
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
6 {
7   cout<<(1+17)*9/2<<endl;
8   return 0;
9 }
```

2.3.43　换零钱2(money)

【题目描述】

把N元换成10元、5元、1元的零钱，求每种零钱都至少各有一张的情况下，共有多少种兑换方案？

【输入格式】

输入一个整数N（$16 \leqslant N \leqslant 10^6$）。

【输出格式】

输出方案数。

【输入样例】

100

【输出样例】

81

【算法分析】

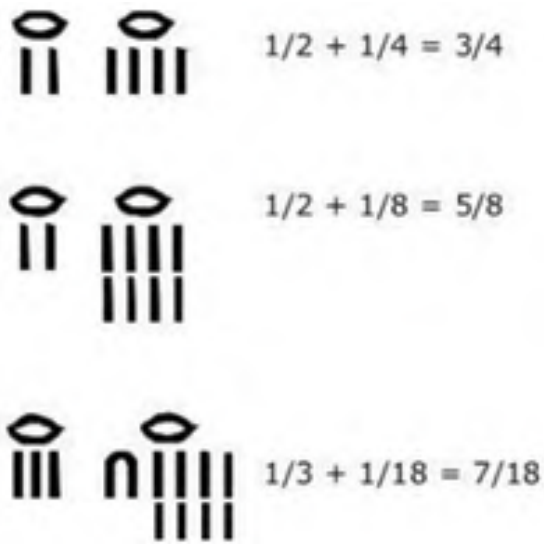
数据规模过大，请考虑使用等差数列求和公式计算。

2.3.44　埃及分数(Egypt)

【题目描述】

一个老人将11匹马分给3个儿子，老大分1/2，老二分1/4，老三分1/6。在3个儿子无奈之际，邻居把自己家的马牵来，老大分1/2，牵走了6匹马；老二分1/4，牵走了3匹马；老三分1/6，牵走了2匹马。分完后，邻居把自己的马牵了回去。即11/12=1/2+1/4+1/6。这种分子是1的分数叫作埃及分数，因为古埃及人只使用分子是1的分数进行分数运算。

现输入一个真分数，请将该分数分解为埃及分数。



【输入格式】

输入两个整数，表示分子和分母。

【输出格式】

输出分解后的埃及分数。

【输入样例】

8 11

【输出样例】

1/2+1/5+1/55+1/110



若真分数a/b中的分子a能整除分母b，则真分数经过化简直接就可以得到埃及分数；若真分数a/b中的分子a不能整除分母b，则可以从原来的分数中分解出一个分母为$c=b/a+1$的埃及分数。

分解后剩下的部分形成一个新的分数a_1/b_1、即$a_1=a\times c-b$、$b_1=b\times c$。继续用这种方法将剩余部分反复分解，最后可得到结果。

以样例来说，8/11分解出来的分母$c=b/a+1=2$，所以第一项为1/2，剩下的部分形成一个新的分数a_1/b_1,即分子$a_1=a\times c-b=5$、分母

$b_1=b\times c=22$，5/22继续分解为1/5……

2.3.45　除式还原(div1)

【题目描述】

石碑上有一个除式如图2.22所示，试解开此除式。

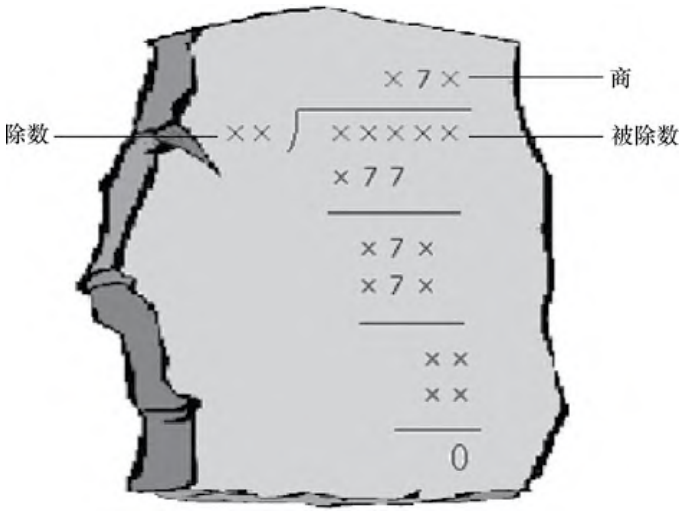


图2.22

【输入格式】

无输入。

【输出格式】

输出被除数、除数及商，输出格式见输出样例，如果有多组答案，每组答案占一行。

【输入样例】

无。

【输出样例】

12/6=2

【样例说明】

输出样例仅为举例，并非正确答案。



由除式本身尽可能多地推出已知条件：

（1）被除数和除数的取值范围分别为10 000～99 999和10～99，且可以整除；

（2）商的取值范围为100～999，且十位数为7；

（3）商的第1位与除数的积为三位数，且后两位为77；

（4）被除数的第3位一定为4；

（5）7乘以除数的积为三位数，且第2位为7；

（6）商的最后一位不能为0，且与除数的积为两位数。

由已知条件就可以采用枚举的方法找出结果。

第03章　竞赛模拟

学完了C++语言的3种基本结构后，就应该尝试做一些简单的竞赛题了，因为真实的比赛环境和规则与平常练习的情况略有不同。

3.1　文件读写



扫码看视频

正式比赛中，一道题会用多组测试数据来评测，只有全部测试数据都通过评测，该题才能得满分。试想一下，如果在评测时，每道题都是评卷老师人工输入测试数据再人工检查结果是否正确，那一定是非常麻烦的事情。所以比赛要求选手编写的代码必须能自行读取指定文件中的测试数据进行处理，并将输出答案写入指定的文件中，之后评卷老师会对所有选手提交的源代码用专业评测软件例如Arbiter、Cena等自动评测计分。

现以比赛环境下的一道简单模拟题来说明。

3.1.1　求中间数（mid）

【题目描述】

从任意输入的3个整数中，找出按大小顺序排序处于中间位置的数。

【输入格式】

输入文件为mid.in，共一行，即3个整数，以空格间隔。

【输出格式】

输出文件为mid.out，共一行，即中间数。

【输入样例】

1 2 3

【输出样例】

2

【时间限制】

1秒



按照NOIP系列比赛的规则，对于这道题，C++选手需提交名为mid.cpp的源代码，不能提交编译好的EXE文件。选手提交的代码要实现从名为mid.in的文件中读取到输入数据，并将输出答案写入名为mid.out的文件里的功能。

每组测试数据必须在1秒内运行出解。

读写文件数据有多种方法，此处只介绍利用freopen()函数将输入、输出重定向到文件中的方法。参考程序如下所示。可以看到，只需在原有代码中加入第7和第8行语句，即可很方便地实现读写文件数据的功能。

```
 1 //求中间数——此处使用数学方法判断，更简单的其他方法请读者自行发挥
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   freopen("mid.in","r",stdin);  //从文件mid.in里读取数据
 8   freopen("mid.out","w",stdout);//程序运行会创建文件mid.out并将结果写入其中
 9   int a,b,c;
10   cin>>a>>b>>c;
11   if ((a-b)*(b-c)>=0)
12     cout<<b<<endl;
```

```
13    else if ((b-a)*(a-c)>=0)
14      cout<<a<<endl;
15    else
16      cout<<c<<endl;
17    return 0;
18  }
```

要成功运行该程序，必须在程序的同一文件目录下用记事本等纯文本编辑软件建立一个名为“mid.in”的文件，并且根据题意，该文件里应该有3个以空格间隔的整数，否则程序运行时会因无法找到输入文件和数据而报错。

如图3.1所示，当计算机操作系统默认设置为隐藏文件扩展名时，完全文件名为mid.txt的文件在显示器上仅显示该文件名为mid，这往往会导致在修改文件名时，看似将文件名修改成了mid.in，但实则修改成了mid.in.txt。



图3.1

上例已知输入文件中的数据个数，那么在不知道文件中有多少个数据的情况下，如何正确读取全部数据并输出呢？请看以下程序。

```
1  //读取文件中所有数据到文件末尾
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  int main()
6  {
7    freopen("sum.in","r",stdin);   //必须有名为sum.in的文件在同一文件目
```

```
录下
 8    freopen("sum.out","w",stdout);//重定向语句要放在main()函数的最前面
 9    int x,i,n;
10    for (i=0; cin>>x; i++) //读取文件中的所有数据，当无数据可读取时退出循
环
11      cout<<x<<" ";
12    cout<<"共有"<<i<<"个数据\n";
13    return 0;
14 }
```



这种读取文件中所有数据到文件末尾的代码，必须要有读取文件数据的语句才可以正确运行。换句话说，如果删除了代码中的第7和第8行，按之前用键盘输入的方式读取数据的话，程序将无法正常结束，因为程序并不知道用键盘输入的数据什么时候会结束，这样就永远无法触发读取文件中所有数据到文件末尾的结束条件。

3.1.2 修改作文（word）

【题目描述】

小光写的作文总是在单词与单词之间插入多余的空格，试编程删除多余的空格。

【输入格式】

输入一个长度不超过300的字符串组成的句子，句首与句尾不含空格。

【输出格式】

输出删除多余空格的字符串。

【输入样例】

I am back.

【输出样例】

I am back.

伪代码如下所示。

```
1 重定向输入文件word.in
2 重定向输出到文件word.out
3 定义字符串t和v
4 当文件word.in里还有下一个单词时，读取下一个单词赋值给字符串v
5     字符串v后面加一个空格后连接到字符串t后面
6 删除字符串t后的最后一个空格
7 输出字符串t
```

3.2 制作测试数据

扫码看视频

并不是只要通过题目的测试数据就能保证代码是正确的，一般要自制多组不同的测试数据自测。自制的测试数据文件应与可执行文件保存在同一目录下，如图3.2所示。



图3.2

例如某道题为求两数之和，文件名为sum，则可自制多组测试数据sum1.in,sum2.in,sum3.in,…，再自制与之相对应的答案文件sum1.ans,sum2.ans,sum3.ans,…。必须保证答案文件是正确的，例如sum1.in和sum1.ans的内容可能与如图3.3所示的内容相似。



图3.3

写好的sum.cpp的代码内容可能如下所示。

```
//求两数之和
#include <bits/stdc++.h>
using namespace std;

int main()
{
  freopen("sum.in","r",stdin);              //注意输入文件名为sum.in
  freopen("sum.out","w",stdout);            //注意输出文件名为sum.out
  int a,b;
  cin>>a>>b;
  cout<<a+b<<endl;
  return 0;
}
```



程序中输入/输出语句的格式必须严格匹配题目要求的输入/输出文件的格式。同理，自制的若干组测试数据文件和答案文件，其数据格式也必须严格匹配题目要求的格式，否则程序无法正确评测。

在平时的训练中，有现成的各类测试软件如Arbiter、Cena等，可以批量测试程序是否正确。但是在正式比赛环境中，不可能给选手提供测试软件和正式测试数据，因此需要选手现场编写测试程序来自测所有的自制测试数据。例如有名为sum.cpp的源文件，编译生成名为sum.exe的可执行文件。自制了5组测试数据，输入数据文件分别为

sum1.in,sum2.in,…,sum5.in，输出答案文件分别为sum1.ans,sum2.ans,…,sum5.ans，则可写测试程序如下所示。

```
 1 //测试程序
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   for (int i=1; i<=5; i++)       //5组测试数据，所以循环5次
 8   {
 9     string F="sum";
10     F+=char(i+48);                    //整数i转为字符i,"0"的ASCII值为48
11     //复制命令，例如i=1时,为"copy sum1.in sum.in"
12     string s="copy "+ F+".in sum.in"; //注意"copy"后有一个空格
13     system(s.c_str());                //执行复制命令
14     int time1=clock();         //获取程序当前运行的时钟数
15     system("sum.exe");        //运行由源代码事先编译好的sum.exe文件
16     int time2=clock();        //获取程序运行结束后的时钟数
17     //比较文件命令,例如i=1时,为"fc sum.out sum1.ans";
18     string Cmd="fc sum.out "+F+".ans";
19     if (!system(Cmd.c_str()))//执行比较文件命令
20        cout<<" 测试点"<<i<<"通过，用时"<<time2-time1<<"毫秒\n\n";
21   }
22   return 0;
23 }
```

第12行将外部指令保存在string中，它会将输入数据文件依次复制为sum.in文件。例如当循环变量i为1时，将文件sum1.in复制为sum.in，即“copy sum1.in sum.in”。

第13行的system()函数用于调用Windows操作系统上的指令，例如system("cls")可以实现清屏操作；system("pause")可以冻结窗口，便于观察程序的执行结果等（调试程序时，经常使用该命令暂停程序运行，以观察程序的执行结果）。因为该函数的形参为字符数组，所

以需要使用c_str()函数将字符串（string）类型数据转换为字符数组后再使用。

第14行的clock()函数用于获取程序从开始运行到该函数出现时所运行的时钟数，单位为毫秒（1 000毫秒＝1秒）。

第15行通过system()函数运行之前已编译好的sum.exe文件，sum.exe文件运行时，将读取sum.in文件的数据进行处理，最后将运行结果输出到sum.out文件里。

第18行字符串Cmd中的字符串是用来比较两个类似文件的不同之处的外部指令。具体来说，就是当i＝1时，比较sum.out和sum1.ans两个文件的异同；当i＝2时，比较sum.out和sum2.ans两个文件的异同……

第19和第20行表示若两个文件的内容无差异，则输出测试结果和运行时间。

写好后编译该源程序，例如设文件名为Check.cpp，则编译好的可执行文件为Check.exe。注意所有文件都应保存在同一目录下，如图3.4所示。



图3.4

双击运行Check.exe，程序将运行并测试比较每一组测试数据，如图3.5所示。

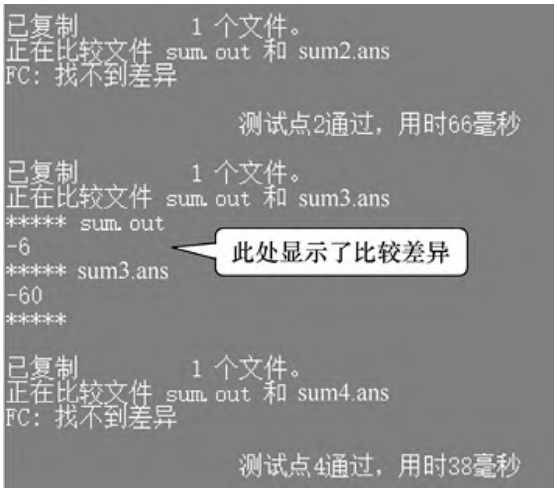


图3.5

正式比赛中，只允许选手提交源文件，不允许提交自制的测试数据和可执行文件，且每个源文件需分别保存在一个单独文件夹中。例如有选手考号为zj_123，所提交的3道题的程序名分别为proble1、proble2、proble3，则一般提交的格式如图3.6所示。

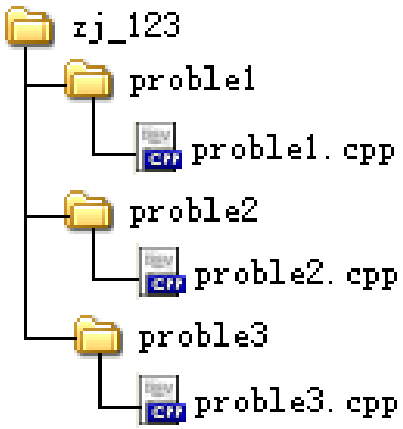


图3.6

3.3　随机数据与对拍



扫码看视频

部分特殊测试数据和极限测试数据必须手动生成，但除此之外的多数测试数据显然可以随机生成，这就需要随机数函数帮忙了。

例如趣味摇奖机的游戏规则如下：计算机在0～9这10个数中随机取出1个数，由学生去猜，猜中的获特等奖，相差1的获一等奖，相差2的获二等奖，相差3的获三等奖，其余的没有奖。

参考程序如下所示。

```
 1 //趣味摇奖机
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   srand(time(0));     //srand()取当前时间作为随机数种子
 8   int t=rand()%10;    //rand()产生0～32767的随机数，随后对10取余
 9   cout<<"      *** 趣味摇奖机 ***   \n\n";
10   cout<<"请任选一个数字(0～9)：  ";
11   int j;
12   cin>>j;
13   if (j<0 || j>9)     //如果输入的数字不符合游戏规则
14     return 0;
```

```
15   if (j==t)
16     cout<<"\n哇,特等奖!你真厉害!";
17   else if (abs(j-t)<=1)
18     cout<<"\n一等奖!很不错呀!";
19   else if (abs(j-t)<=2)
20     cout<<"\n二等奖!也可以啦...";
21   else if (abs(j-t)<=3)
22     cout<<"\n三等奖!还要努力哦...";
23   else
24     cout<<"\n真可惜!什么都没有...";
25   return 0;
26 }
```

第7行代码中的time(0)获取格林尼治时间1970年01月01日00时00分00秒起至现在的总秒数，srand()以此作为随机数种子，通过rand()产生0～32 767的随机数。如果没有这一句，每次运行程序时，rand()得到的随机数是相同的。

产生0～1的随机数的参考程序如下所示。

```
 1 //产生0～1的随机数
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   srand(time(0));                //srand()取当前时间作为随机数种子
 8   for (int i=1; i<=5000; i++)
 9     cout<<rand()%(2)<< "  ";
10   return 0;
11 }
```

产生low和hight范围内的随机数的参考程序如下所示。

```
 1 //产生指定范围内的随机数
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int low,hight;
```

```
 8    srand(time(0));                    //srand()取当前时间作为随机数种子
 9    cin>>low>>hight;                      //输入low、hight值，大小不得颠
倒
10    for (int i=1; i<=5000; i++)
11      cout<<rand()%(hight-low+1)+low<<"  ";
12    return 0;
13  }
```

产生随机字符串的参考程序如下所示。

```
1  //产生指定长度的随机字符串
2  #include <bits/stdc++.h>
3  using namespace std;
4
5  int main()
6  {
7    int i,n,m;                                  //输出n行m个字符的随机字符串
8    string str;
9    cin>>n>>m;
10    srand(time(0));
11    for (i=1; i<=n; i++)
12    {
13      str="";                                   //清空字符串
14      for (int j=1; j<=m; j++)
15      {
16        int temp=rand()%2;                      //随机决定输出大写字母或小写字
母
17        if (temp)                               //相当于temp==1
18          str+=(char)(rand()%(26)+1+64);//随机生成大写字母加到str末尾
19        else
20          str+=(char)(rand()%(26)+1+96);//随机生成小写字母加到str末尾
21      }
22      cout<<str<<endl;
23    }
24    return 0;
25  }
```

3.3.1　模拟骰子

【题目描述】

游戏中的骰子类似一个正立方体，上面分别有1～6个孔（或数字），其相对两面的孔数（或数字）之和必为7。试编程模拟使用骰子产生的随机数。



在比赛中，有时对于一道题写好了相关代码却不知道正确与否怎么办？

例如，对于输入a、b、c的值，输出（a*b）%c的值这一道题，为了防止计算过程中数据过大而导致“溢出”，可以使用（a%c*b%c）%c这个公式。但是如何验证用这个公式写的代码是否正确呢？具体操作步骤如下所示。

首先根据题目要求写一个名为CreatRand.cpp的随机生成测试数据的程序。

```
 1 //随机生成测试数据的程序
 2 #include<bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   freopen("test.in","r",stdin);    //若使用Dev-C++智能开发平台则此行代码必写
 8   freopen("test.in","w",stdout);   //生成的随机数据写入test.in文件
 9   srand(time(0));
10   int a=rand();                    //设为整数（int）类型，在小范围内验证即可
11   int b=rand();
12   int c=rand();
13   cout<<a<<" "<<b<<" "<<c<<endl;   //输出的值将写入test.in文件
14   return 0;
15 }
```



如果用Dev-C++智能开发平台编写代码，那么在CreatRand.cpp中，若不添加第7行代码（因为实际上并没有读取任何文件的数据，所以读取文件的名称可以任意写），自检程序就无法自动循环运行下去。使用其他编译器则无须添加此行代码。

再写两个不同算法的程序来“对拍”检验。先写一个名为program1.cpp的普通程序。

```
 1 //program1.cpp —— 普通写法
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   freopen("test.in","r",stdin);              //从test.in中读取数据
 8   freopen("test1.out","w",stdout);           //注意输出文件为
test1.out
 9   long long a,b,c;
10   cin>>a>>b>>c;
11   cout<<a*b%c<<endl;
12   return 0;
13 }
```

第2个程序是使用公式改进后准备最终上交的程序，名为program2.cpp，我们将验证该程序是否正确。

```
 1 //program2.cpp —— 公式法
 2 #include <bits/stdc++.h>
 3 using namespace std;
```

```
 4
 5 int main()
 6 {
 7   freopen("test.in","r",stdin);      //注意读取的文件也是test.in
 8   freopen("test2.out","w",stdout);   //注意输出文件不同，为test2.out
 9   long long a,b,c;
10   cin>>a>>b>>c;
11   cout<<(a%c*b%c)%c<<endl;
12   return 0;
13 }
```

将这3个源程序均编译成可执行文件，并放在同一文件夹下。在该文件夹下编写一个名为Check.cpp 的对拍程序，并将其编译成名为Check.exe的可执行文件。

```
 1 //对拍程序
 2 #include<bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   freopen("test.in","r",stdin); //若使用Dev-C++智能开发平台则需加此行代码以保持程序运行
 8   do
 9   {
10     system("CreatRand.exe");     //运行CreatRand.exe产生随机测试数据
11     system("program1.exe");      //运行program1.exe生成test1.out
12     system("program2.exe");      //运行program2.exe生成test2.out
13   } while(system("fc test1.out test2.out")==0);//比较两个答案文件的异同
14   return 0;
15 }
```

如图3.7所示，双击运行Check.exe即可自动评测。一旦发现文件差异，程序即结束循环并退出运行，此时就可以打开test.in、test1.out及test2.out文件来查错。

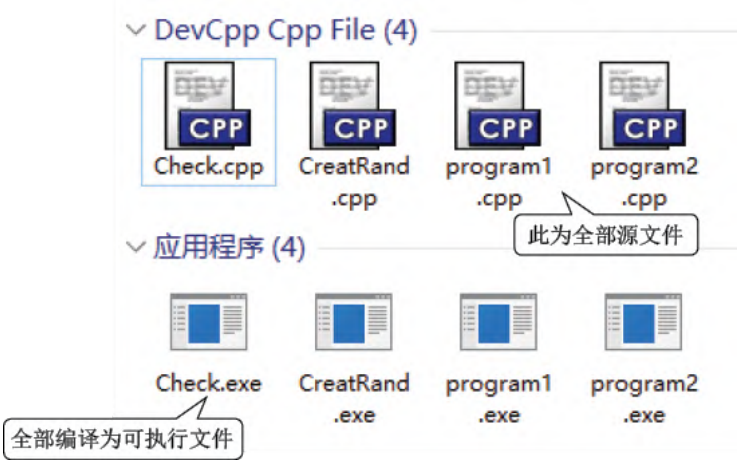


图3.7

3.4 效率分析及简单优化



扫码看视频

每个程序的运行次数称为该程序的“时间复杂度”。

例如，有一个程序段如下所示，其时间复杂度为$O(n)$。

```
1 for (int i=0; i<n; i++)            //循环了n次
2     x=x+2;
```

例如，有一个程序段如下所示，其时间复杂度为$O(n^3)$。

```
1 for (int i=0; i<n; i++)            //本层循环了n次
2   for (int j=i; j<n; j++)          //本层循环了n次
```

```
3      for (int k=j; j<n; k++)      //本层循环了n次，共循环了n×n×n次
4        x+=2;
```

例如，有一个程序段如下所示，其时间复杂度为$O(\log_2 n)$。这是因为i的值每次循环均增大到原先的一倍，所以每次循环后，循环次数降低为原先的一半。设循环次数为x，有$2^x=n$，则$x=\log_2 n$，故该算法的时间复杂度为$O(\log_2 n)$，通常记为$O(\log n)$。如果一个for循环（n次）里套一个此类算法，那么时间复杂度为$O(n\log n)$。

```
1 int i = 1, n = 1024;
2 while (i<n)
3   i=i*2;
```

例如，有一个程序段如下所示，其代码的运行次数为n^2+n+1。但当n足够大，例如$n=10^6$时，可以看到代码的运行时间主要取决于多项式的第一项，后面的基本可以忽略不计，所以其时间复杂度应为$O(n^2)$。

```
1 for (int i=0; i<n; i++)
2   for (int j=0; j<n; j++)
3     k++;
4 for (int i=0; i<n; i++)
5   t++;
6 p++;
```

试判断下列式子的时间复杂度：

（1）$3n^4+8n^2+n+2$；

（2）$2^{n+1}+n^{100}+5$；

（3）$1^2+\cdots+n^2$；

（4）1k＋…＋n^k；

（5）$\sum_{i=1}^{n}\sum_{j=1}^{n}(j-i+1)$（Σ为数学求和符号，英语名称为sigma，例如$\sum_{i=1}^{T} i$，即求1+2+…+$T$的值）。



答案如下：

（1）$O(n^4)$；

（2）$O(2^{n+1})$，因为2^{n+1}是指数级别，增长速度远大于多项式级别的n^{100}；

（3）$O(n^3)$，因为$1^2+\cdots+n^2=n(n+1)(2n+1)/6$；

（4）$O(n^{k+1})$，类似（3）；

（5）$O(n^2)$，两层循环，外循环从1到n变化，内循环也从1到n变化。

设某算法的时间复杂度函数的递推方程是$T(n)=T(n-1)+n$（n为正整数）和$T(0)=1$，则该算法的时间复杂度为（　　）。

A. $O(\log n)$　　B. $O(n\log n)$　　C. $O(n)$　　D. $O(n^2)$

【解析】$T(n)=T(n-1)+n=T(n-2)+n-1+n=T(n-3)+n-2+n-1+n=\cdots$

$$=T(0)+(1+2+3+\cdots+n)$$

$$=\frac{n\times(n+1)}{2}+1$$

故答案选D。

“空间复杂度”是程序运行所需要额外消耗的存储空间，也用$O()$来表示，它是判断算法优劣的重要度量指标。

程序的复杂程度称为“程序复杂度”。显然，程序写得越简单、复杂度越低越好。

编程竞赛必须要考虑到程序的运行效率，对输入/输出进行优化是提高程序运行效率最普遍的方法之一。

C++语言的标准输入/输出流cin和cout使用非常方便，但在涉及大量数据输入/输出时，因为需要花费时间自动判断输入/输出数据的类型并转换为相应的格式输入/输出，所以运行效率并不是最高。因此需要大量读写数据时，一般使用C++语言的scanf()函数读取数据和printf()函数输出数据效率会更高。

printf()函数可以向终端输出任意类型的多个数据。它的一般格式如下。

```
printf(格式控制，输出表列)
```

例如printf(″%d,%c\n ″,i,c);表示将变量*i*以整数形式输出，变量*c*以字符形式输出，两个变量间以逗号间隔。

格式控制是指用双引号标注的字符串，它包括两种信息：一种为普通字符，即需要原样输出的字符；一种为格式说明，由“%”和格式字符组成，例如“%d”“%f”等。它的作用是将输出的数据转换为指定的格式输出。如表3.1所示为printf()函数可用的格式字符。

表3.1

格式字符	说明
%d	以带符号十进制数形式输出整数
%lld	以超长整数形式输出
%c	以字符形式输出，只输出一个字符
%s	以字符串形式输出
%f	以小数形式输出单精度浮点数，隐含输出6位小数
%lf	以小数形式输出双精度浮点数，隐含输出6位小数
%o	以八进制数的形式输出

格式字符	说明
%x	以十六进制数的形式输出

请务必注意数据类型应与上述格式字符说明匹配，否则可能会发生错误。部分使用示例如下所示。

```
 1 //printf()函数的使用示例
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   //输出整数,即输出a=123
 8   printf("a=%d\n",123);
 9   //输出整数宽度为5,默认右对齐,即输出a=  123
10   printf("a=%5d\n",123);
11   //输出整数宽度为5,负号表示左对齐,即输出1    ,2
12   printf("%-5d,%d\n",1,2);
13   long long x=1234567890000;
14   //输出超长整数,即输出1234567890000
15   printf("%lld\n",x);
16
17   float y=123.456;
18   //输出单精度浮点数,即输出y=123.456001。注意此处有误差
19   printf("y=%f\n",y);
20   //输出数据宽度为12,默认右对齐,即输出y=  123.456001
21   printf("y=%12f\n",y);
22   //输出数据宽度为12,负号表示左对齐,即输出y=123.456001  ,OK
23   printf("y=%-12f,OK\n",y);
24   //输出数据总宽度为10,小数部分取2位,即输出y=    123.46。注意此处有误差
25   printf("y=%10.2f\n",y);
26   //输出数据总宽度为1(过小无效),小数部分取2位,即输出123.46。注意此处有误
差
27   printf("%1.2f\n",y);
28
29   double z=45.67;
30   //输出双精度浮点数,即输出45.670000
31   printf("%lf\n",z);
32   //输出数据宽度为12,默认右对齐,即输出z=   45.670000
33   printf("z=%12lf\n",z);
```

```
34    //输出数据宽度为12,负号表示左对齐,即输出z=45.670000    ,OK
35    printf("z=%-12lf,OK\n",z);
36    //输出数据宽度为10,小数部分取2位,即输出z=      45.67
37    printf("z=%10.2lf\n",z);
38    //输出数据宽度为0(过小无效),小数部分取2位,即输出z=45.67
39    printf("z=%0.2lf\n",z);
40
41    char c='x';
42    //输出单个字符
43    printf("%c\n",c);
44    //输出字符串,即输出s=abc
45    printf("s=%s\n", "abc");
46    //输出字符串宽度为8,右对齐,即输出s=      abc
47    printf("s=%8s\n","abc");
48    //输出字符串宽度为8,左对齐,即输出a       ,b
49    printf("%-8s,%s\n","a","b");
50    //输出字符串长度大于2,宽度限制无效,即输出s=abc
51    printf("s=%2s\n","abc");
52    //宽度为10,取左端4个字符,默认右对齐,即输出       abcd
53    printf("%10.4s\n","abcdef");
54    //宽度为8,取左端3个字符,负号表示左对齐,即输出abc     ,OK
55    printf("%-8.3s,OK\n","abcde");
56    return 0;
57 }
```

scanf()函数的作用是输入指定形式的数据。该函数的一般格式如下。

```
scanf(格式控制，地址表列)
```

格式控制的含义同printf()函数；地址表列是指由若干个地址组成的表列，可以是变量的地址，也可以是字符串的首地址。scanf()函数格式字符的使用方法与printf()函数格式字符的使用方法类似。

下面是用scanf()函数、printf()函数输入/输出数据的实例。

```
1 //使用scanf()函数、printf()函数输入/输出数据
2 #include <bits/stdc++.h>
3 using namespace std;
4
```

```
5 int main()
6 {
7   int a,x,y;
8   float b;
9   long long c;
10   scanf("%d %f %lld",&a,&b,&c);   //输入整数、单精度浮点数、超长整数
11   printf("%d,%f,%lld\n",a,b,c);
12
13   scanf("%o %x",&x,&y);           //以八进制数、十六进制数形式输入x、y
14   printf("%d %d\n",x,y);          //以十进制数形式输出x、y
15
16   char ch;
17   getchar();                      //用getchar()函数消去之前输入的换
行符
18   scanf("%c",&ch);                //否则字符变量ch就会接收到换行符
19   printf("%c\n",ch);
20   return 0;
21 }
```

例程第10行中的地址表列的表示方法为&a,&b,&c，&是C++语言中的取地址符，意思是取a这个变量的地址，取b这个变量的地址，取c这个变量的地址。通俗的解释即找到a、b、c所在的内存地址，将用键盘输入的值放入其中，即a、b、c被赋值。



scanf()函数需要加入“&”这个取地址符，printf()函数不需要加入“&”这个取地址符，注意不要混淆。

getchar()函数表示从键盘获得一个字符，因为第13行输入完数据后需要按Enter键确定，此换行符要用getchar()函数消去，否则第18

行的输入字符ch将无法正确赋值。

如果在格式控制中除了格式说明以外还有其他字符，则在输入数据时在对应位置应输入与这些字符相同的字符。

例如：

```
scanf("%d,%d",&a,&b);
```

输入时应用如下形式。

```
3,4
```

注意3后面是逗号，因为要与该scanf()函数中格式控制中的逗号相对应。如果输入时不严格一一对应而用空格或其他字符，则会发生错误。

例如：

```
scanf("%d %d",&a,&b);                    //中间有2个空格
```

程序运行输入2个数据时应有2个或更多的空格。

例如：

```
scanf("%d:%d:%d",&h,&m,&s);
```

程序运行输入数据时应该用类似于12:23:34的格式。

scanf()函数输入有一个不可忽视的优势——过滤不想读取的字符。

例如：

```
scanf("%d+%d+%d",&a,&b,&c);          //输入1＋2＋3
```

这样就把“+”忽略了，直接把1、2、3分别赋值给a、b、c。

例如：

```
scanf("%d %*d %d",&m,&n);            //输入113 118 69
```

113、69分别被赋值给m、n，因为“*”表示跳过它相应的数据，所以118不赋值给任何变量。



事实上，可以通过两条语句使cin/cout的执行效率达到甚至超过scanf()函数/printf()函数，其语句分别如下。

ios::sync_with_stdio(false)；,取消cin与stdin的同步，使cin与cout拥有与scanf()函数和printf()函数相近的执行效率。注意使用该语句后，cin和cout不可与scanf()函数和printf()函数等混合使用！

cin.tie(0);，解除cin与cout的绑定，进一步加快执行效率。

简单代码示意如下。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    for(int i=0,x; i<1e7; i++)//1e7后的执行效率超过scanf()函数/printf()函数
    {
      cin>>x;
      cout<<x<<'\n';           //用endl会降速，应替换为'\n'
    }
    return 0;
}
```

3.4.1 排行榜（list）

【题目描述】

试将成绩排行榜中成绩超过10 000 000（十进制数）的学员的信息输出。

【输入格式】

输入文件为list.in，第一行是整数*n*，表示有*n*个学员，随后*n*行为每个学员的代号（1个英文字符，代号各不相同）、测试时间、成绩

（为八进制数形式，保证至少有一个值超过十进制数10 000 000）及评价分数（双精度浮点数）。数据与数据之间的分隔符见输入样例。

【输出格式】

输出文件为list.out，输出满足条件的学员信息：编号左对齐，输出宽度为3个字符；学员代号右对齐，输出宽度为2个字符；成绩以十进制数形式输出，右对齐，输出宽度为12个字符；评价分数右对齐，输出宽度为6个字符，保留小数点后1位（四舍五入）；评价分数与测试时间以两个空格间隔，测试时间的年、月、日以“-”间隔，行末无多余空格。

【输入样例】

```
3

Z 1990/8/2,46113200,80.2

L 2017/7/2, 55615400, 81.50123

W 2123/6/4, 61456500, 83.89123
```

【输出样例】

```
1    L         12000000    81.5    2017-7-2
```

2	W	13000000	83.9	2123-6-4

第04章　数组

数组可以把相同类型的若干变量按照有序的形式组织起来。合理地使用数组，会使程序的结构比较整齐，而且可以把较为复杂的运算转化成简单的数组来表示。

4.1　一维数组



扫码看视频

如果有无数个相同类型的数据需要存储和运算，难道需要逐个定义变量名吗？当然不用，使用C++语言中的数组可以很方便地把相同类型的若干变量按照有序的形式组织起来。

数组和变量一样，必须先定义再使用。例如定义一维数组int a[10]。

它表示数组名为a的数组共有10个int类型的整数（数组元素），分别为a[0],a[1],a[2],…,

a[9]。可以看出，数组元素的标识方法为数组名[下标]，且下标从0开始，如图4.1所示。

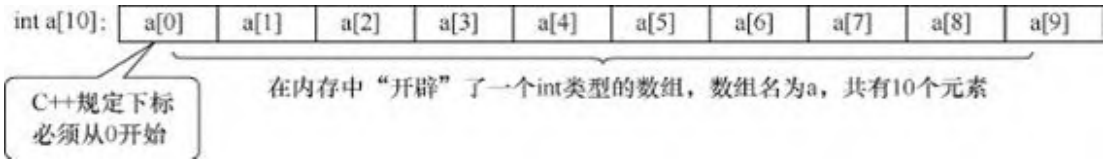


图4.1

又如定义一维数组double b[5]。这表示b数组有5个double类型的数组元素，分别为b[0]，b[1]，b[2]，b[3]，b[4]。同理，定义一维数组bool x[n]，表示x数组包含n个数组元素，分别为x[0]，x[1]，…，x[n－1]，且均为布尔类型。

定义数组时可以同时对数组元素赋值，有以下方法可以实现。

（1）在定义数组时对数组元素赋初始值。例如int a[10]={0,1,2,3,4,5,6,7,8,9}。

（2）可以只给一部分数组元素赋初始值。例如int a[10]={0,1,2,3,4}只给前5个数组元素赋了初始值，后5个数组元素自动初始化为0。

（3）在对全部数组元素赋初始值时，可以不指定数组元素的数量，系统会根据赋值的数组元素个数自动定义数组元素的数量。例如int a[]={1,2,3,4,5}相当于int a[5]={1,2,3,4,5}。



定义数组元素数量时要注意内存空间的限制，例如竞赛规定仅允许使用不超过256 MB的内存空间，那么程序大约可以定义数组元素数量为60 000 000的整型数组（60 000 000×4 B / 1 024 / 1 024≈228 MB）。超出内存空间限制会导致程序运行失败。

下面是逆序输出数组元素值的程序，可以看出，只能逐个引用数组元素而不能一次引用整个数组。例如cin>>a[0]>>a[1]>>a[2]是正确的，而cin>>a、cin>>a[n]或cout<<a这种试图一次性输入或输出全部数组元素的操作是错误的。

```
 1 //逆序输出数组元素值
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a[10];                          //此时数组里的各数组元素值未知,不是0
 8   for (int i=0; i<=9; i++)            //数组元素必须逐个赋值
 9     cin>>a[i];                        //输入a[i]的值
10   for (int i=9; i>=0; i--)            //逆序输出
11     cout<<a[i]<<" ";                  //数组元素必须逐个输出
12   return 0;
13 }
```

4.1.1 上楼梯（stairs）

【题目描述】

小光要上楼梯，他每次能向上走1阶、2阶或3阶楼梯，问n阶楼梯有几种不同的走法？

【输入格式】

输入文件为stairs.in，有多组数据，每组一个整数n（$n \leqslant 73$），表示楼梯阶数。

【输出格式】

输出文件为stairs.out，每组输出一行，每行一个整数，即有几种不同的走法。

【输入样例】

1

4

【输出样例】

1

7

【算法分析】

如果楼梯只有1阶，有1种走法；

如果楼梯有2阶，有2种走法；

如果楼梯有3阶，有4种走法；

如果楼梯有4阶，有7种走法；

如果楼梯有5阶，有13种走法……

其规律是：从第4个数开始，走到当前阶梯的走法数是由走到前3个阶梯的走法数相加而来的。

设F[i]表示走到当前阶梯的走法数，则有：

F[1]=1

F[2]=2

F[3]=4

$F[i]=F[i-3]+F[i-2]+F[i-1]\ (i \geqslant 4)$

参考程序如下所示。

```
1 //上楼梯
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
6 {
7   freopen("stairs.in","r",stdin);
8   freopen("stairs.out","w",stdout);
```

```
 9    unsigned long long F[75]={0,1,2,4};//注意n=73的结果会非常大
10    for (int i=4; i<=73; i++)
11      F[i]=F[i-3]+F[i-2]+F[i-1];          //预处理出所有结果
12    for (int n; scanf("%d",&n)!=EOF;)    //C++中的EOF表示文件结束
13      printf("%llu\n",F[n]);             //unsigned long long的格式
控制
14    return 0;
15  }
```

4.1.2　逆序输出数组元素（reverse）

【题目描述】

有一个由整数组成的数组，数组元素为n个，其中a[0]＝2,a[1]＝4,…,a[n]＝2×a[n-1]，试编程逆序输出各数组元素。

【输入格式】

输入一个整数n（$2 \leqslant n \leqslant 50$）。

【输出格式】

逆序输出各数组元素，每个数组元素占一行。

【输入样例】

3

【输出样例】

8

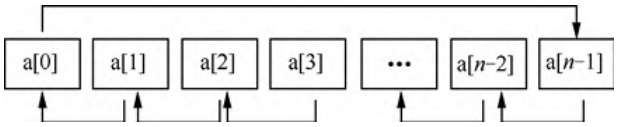
4

2

4.1.3　数组元素前移（move）

【题目描述】

有一个由整数组成的数组，数组元素为*n*个，用键盘输入，试将数组中第一个数组元素移到数组末尾，其余数组元素依次前移一个位置后顺序输出。



【输入格式】

第一行为整数*n*（*n*<100），表示有*n*个数组元素，第二行为*n*个数组元素。

【输出格式】

依次输出前移位置后的数组元素，每行一个数组元素。

【输入样例】

3

1 2 3

【输出样例】

2

3

1

4.1.4　分糖（candy）

【题目描述】

100个人围成一圈分糖，初始时每人手中的糖的个数分别为3、5、7、9……由第一个人开始，每人将一半的糖分给下一个人，多余的自己吃掉（例如某人手中有5块糖，分给下一个人5/2＝2块糖后，多余的1块自己吃掉）。问经过一轮操作后，每人手中还有多少块糖？

【输入格式】

无输入。

【输出格式】

从第一个人开始依次输出每人手中的糖的个数，每人一行数据。

【输入样例】

略。

【输出样例】

略。

4.1.5　开关灯1（light1）

【题目描述】

有100个排成一行的灯，编号为1～100。初始时所有灯是关闭的，第一个人对所有编号为2的倍数的灯改变状态（所谓改变状态就是将原先亮着的灯关闭，将原先未亮的灯点亮），第二个人对所有编号为3的倍数的灯改变状态，两人操作完毕后，依次输出各灯的状态（以1表示打开，0表示关闭）。

参考程序如下所示。

```
1 //开关灯1
2 #include <bits/stdc++.h>
3 using namespace std;
```

```
 4
 5 int main()
 6 {
 7   int f[101];                //没赋初始值的情况下，数组f[]内的各元素值未
知
 8   memset(f,0,sizeof(f));   //将数组f[]内的数组元素全部赋值为0
 9   for (int i=1; i<=100; i++)
10     if (i%2==0)
11       f[i]=!f[i];           //状态取反
12   for (int i=1; i<=100; i++)
13     if (i%3==0)
14       f[i]=!f[i];
15   for (int i=1; i<=100; i++)
16     printf("%d",f[i]);
17   return 0;
18 }
```



memset(f,0,sizeof(f))的作用是将数组f[]内的数组元素全部赋值为0，memset(f,-1,sizeof(f))的作用是将数组f[]内的数组元素全部赋值为-1，那么memset(f,1,sizeof(f)) 的作用是将数组f[]内的数组元素全部赋值为1吗？

memset()函数的作用是将数字以单个字节逐个复制的方式放到指定的内存中，int类型占用4字节，对每个字节赋值1就变成了二进制数00000001 00000001 00000001 00000001，转化为十进制数为16 843 009。

memset(f,127,sizeof(f))在内存中存放的内容是01111111 01111111 01111111 01111111，即十进制数2 139 062 143，这就实现

了将数组中所有数组元素初始化为一个很大的数的目的。

memset(f,128,sizeof(f))在内存中存放的内容是10000000 10000000 10000000 10000000，即十进制数-2 139 062 144，这就实现了将数组中所有数组元素初始化为一个很小的数的目的。

4.1.6 开关灯2（light2）

【题目描述】

将n个灯编号为1,2,3,…,n，开始时灯全不亮。

现有n个人去拉开关，第1个人把编号为1的倍数的灯的开关都拉一下，第2个人把编号为2的倍数的灯的开关都拉一下，第3个人把编号为3的倍数的灯的开关都拉一下……直到第n个人把编号为n的倍数的灯的开关都拉一下。请问这n个人全拉完灯的开关后，有多少个灯是亮的?

【输入格式】

输入一个整数n（$1 < n \leqslant 1\,000\,000\,000$）。

【输出格式】

输出一个整数，表示有多少个灯是亮的。

【输入样例】

100

【输出样例】

10

【算法分析】

因为数据规模大，一般的模拟算法可能会超时。

4.1.7　放花炮（fire）

【题目描述】

n个小伙伴一起放花炮，他们先同时放响了第一个花炮，随后n个人分别以$A_1, A_2, A_3, \cdots, A_n$秒的间隔继续放花炮，到最后每个人都放了$b$个花炮（包括第一个）。请问总共可听到多少声花炮响？



【输入格式】

用键盘输入3行数据，第1行仅一个整数n（$n \leqslant 10$），第2行是$A_1, A_2, A_3, \cdots, A_n$共$n$个整数（每个整数$\leqslant 100$，各整数间以空格相隔），第3行只有一个整数$b$（$b \leqslant 100$）。

【输出格式】

输出仅一行，即一个整数（听到的花炮响声数）。

【输入样例】

3

1 2 3

4

【输出样例】

7

【算法分析】

因为最多不超过10个人，所以可定义数组A[10]存放每个人放花炮的时间间隔数。

因为每个人最多可能放100个花炮，时间间隔最多可能为100秒，所以全部人放完花炮的时间最多不超过10 000秒。为此定义bool数组a[10005]用于标记每1秒是否有放花炮。例如，初始时同时放了一个花炮，则a[1]=1；第3秒某个人放了一个花炮，则a[3]=1。枚举模拟每个人放花炮的过程，最后统计数组a里1的个数即可。

4.1.8 冒泡排序法（sort）

【题目描述】

对无序的N个整数按从小到大的顺序输出。

【输入格式】

输入N+1个数，第1行输入整数N（$N \leqslant 10\,000$），第2行输入N个整数，表示待排序数。

【输出格式】

输出从小到大排好序的整数，以空格间隔，最后一个数末尾无空格，有换行。

【输入样例】

3 2 3 1 5

【输出样例】

1 2 3 3 5

【算法分析】

对一组数排序可以使用冒泡排序法，其思路是：将相邻两个数比较，将小的数交换到前面，好像轻的气泡上浮、重的石头沉到水底一样。冒泡排序需要两层循环，设外层循环为大循环、内层循环为小循环，5个数9、8、5、4、2的排序方法如图4.2所示。

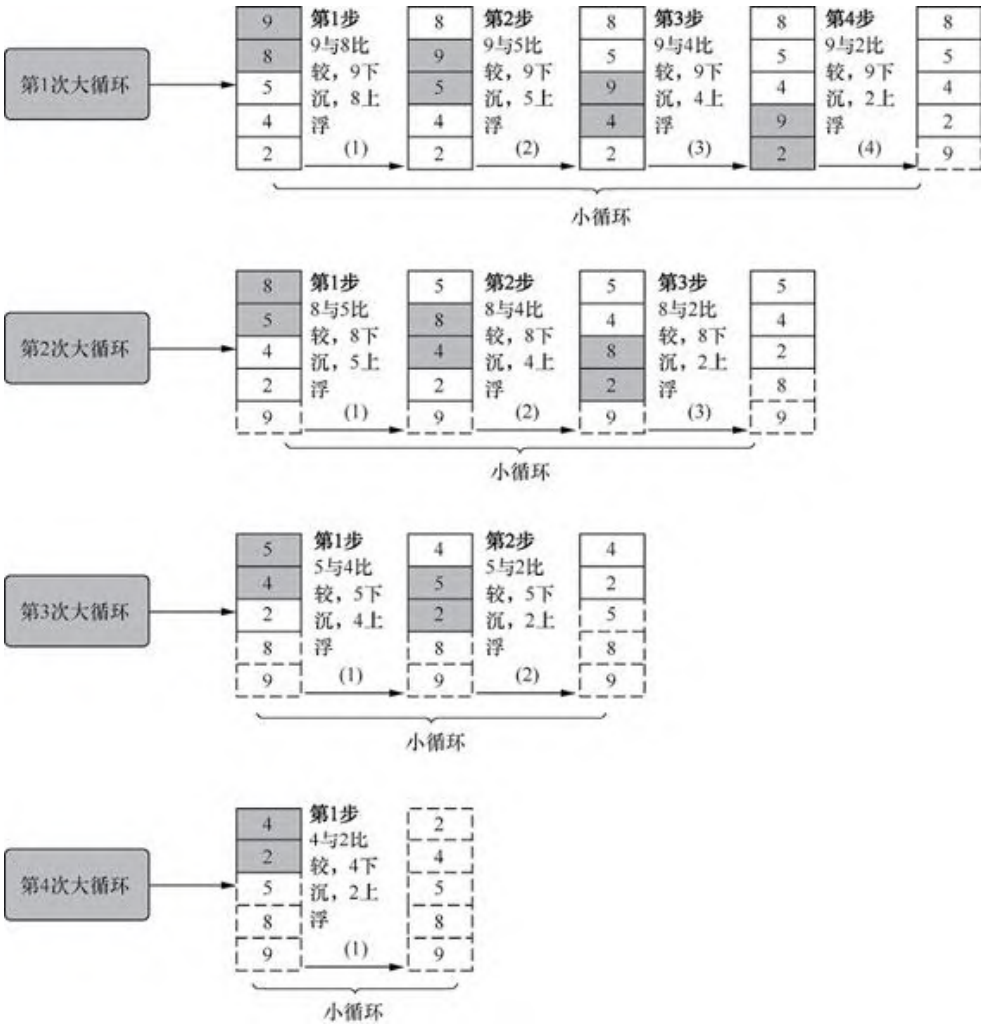


图4.2

假设有n个元素，则大循环的循环次数为n-1，而每个大循环内的小循环的循环次数依次减1，初始小循环的循环次数为n-1。请将下面对10个数由小到大排序的例程改写为对N个数由小到大排序的程序。

```
 1 //用冒泡排序法对10个数排序（由小到大）的例程，请自行修改为对N个数排序的程序
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int i,j;                  //在main()函数之上定义，为全局变量，初始值自动为0
 6 int a[11];                //在main()函数之上定义，为全局数组，所有元素值均为
0
 7
```

```
 8 int main()
 9 {
10   for (i=1; i<11; i++)              //从a[1]开始，a[0]不参与运算
11     scanf("%d",&a[i]);
12   for (j=1; j<=10-1; j++)           //大循环的循环次数为9次
13     for (i=1; i<=10-j; i++)         //每个小循环的循环次数逐次递减
14       if (a[i]>a[i+1])              //比较两个数，例如前面的数大于后面的数
15         swap(a[i],a[i+1]);          //则交换，即小数上浮,大数下沉
16   for (i=1; i<11; i++)
17     printf("%d ",a[i]);             //最后输出已排好序的数组
18   return 0;
19 }
```



冒泡排序法可以做进一步的改进，学有余力的读者可自行思考。

4.1.9 车厢重组（train）

【题目描述】

老火车站旁有一座桥，桥的长度最多能容纳两节车厢，其桥面可以绕河中心的桥墩水平旋转。如果将桥旋转180度，则可以交换相邻两节车厢的位置，用这种方法可以重新排列车厢的顺序。问输入初始的车厢顺序，最少旋转多少次能将车厢从小到大排序。

【输入格式】

第一行是车厢总数N（$N \leqslant 50\,000$），第二行是N个不同的数，表示初始车厢顺序。

【输出格式】

输出一个数，即最少的旋转次数。

【输入样例】

4

4 3 2 1

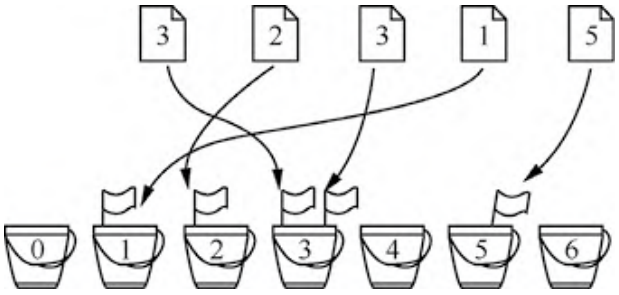
【输出样例】

6

4.1.10 统计各数据的个数（number）

【题目描述】

有0～20的整数N个，统计N的个数和每个数的个数。



【输入格式】

输入N（$N \leqslant 100\,000$）个整数，以空格间隔。

【输出格式】

输出第一行为一个整数N，即数的个数，第二行为每个数的个数。

【输入样例】

3 2 3 1 5

【输出样例】

5

0 1 1 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

最容易想到的方法是把这*N*个数存到数组里，然后一遍一遍地扫描统计。但其实可以只定义一个包含21个元素的数组例如b[21]，初始值均为0。当读取5时，b[5]++；当读取8时，b[8]++。

最后依次输出b[0]～b[20]的值即答案。这种方法称为“桶排序”。

```
 1 //统计不同数据的个数的优化算法，利用下标
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int b[20+1];      //定义在main()函数之上的数组为全局数组，元素值均为0
 6 int i,temp,sum;   //定义在main()函数之上的变量为全局变量，初始值均为0
 7
 8 int main()
 9 {
10   while (scanf("%d",&temp)==1)           //scanf()函数读取数据成功会返回1
11   {
12     b[temp]++;
13     ++sum;
14   }
15   cout<<sum<<endl;
16   for (i=0; i<=20; i++)
17     printf("%d ",b[i]);
18   return 0;
19 }
```



第10行的while(scanf("%d",&temp)==1)执行时，若scanf()函数读取temp的值成功则会返回1。例如while(scanf("%d %d",&a,&b)==2)

的语句：如果a和b都被成功读取，则返回值为2;如果只有a被成功读取，则返回值为1;如果a和b都未被成功读取，则返回值为-1。

4.1.11　狼抓兔子（mystery）

【题目描述】

一只兔子躲进了10个环形分布的山洞中的一个。狼在第1个山洞中没有找到兔子，就隔一个山洞，到第3个山洞去找；也没有找到，就隔2个山洞，到第6个山洞去找；以后每次多隔一个山洞去找兔子……这样下去找了1 000次也没找到兔子，请问兔子可能躲在哪个山洞中？



使用“内容市场”
APP扫描看视频

【输入格式】

无输入。

【输出格式】

由小到大输出兔子可能躲的山洞的编号，每个数占一行。

【输入样例】

无。

【输出样例】

2

……



可定义一个数组cave[10]，cave[0]～cave[9]依次对应第1～第10个山洞，初始值均为0，表示狼还没有进入过山洞。然后循环1 000次模拟狼进山洞的过程，凡是狼进过的山洞均标记为1。当狼走到最后一个山洞的位置时，下一个位置应该是第1个山洞，这可以使用if语句判断，也可以通过取10的余数（值为0～9）的方法实现。

最后输出标记仍为0的山洞的位置即可。

伪代码如下所示。

```
定义数组cave[10]表示10个山洞,初始值均为0，表示狼还没进入过这些山洞
定义pos表示狼的位置，初始值为0，表示狼先要进第1个山洞
尝试1000次
{
   cave[pos]=1,表示狼所在位置的山洞标记为1，即该洞狼已进入过
   计算出狼在下一个山洞的位置（保证永远为0～9），即更新pos的值
}
输出所有值为0的山洞的位置
```

4.1.12　求分数精确值1（exact）

【题目描述】

使用数组精确计算M/N（$0<M<N\leqslant 100$）的值，保留小数点后100位。

【输入格式】

输入两个整数，即分子和分母，中间用斜杠分隔。

【输出格式】

输出M/N的值，保留小数点后100位。

【输入样例1】

27/29

【输出样例1】

27/29=0.93103448275862068965517241379310344827586206896551724137931034482758620689655172413793103448275862068965517241379310344827586206

【输入样例2】

1/25

【输出样例2】

1/25=0.04000000000

由于计算机字长的限制，常规的浮点运算都有精度限制，为了得到高精度的计算结果，必须自行设计实现方法。

为实现高精度的计算，可将商存放在一维数组中，数组的每个元素存放一位十进制数，即商的第一位存放在第一个元素中，商的第二位存放在第二个元素中……依次类推。这样就可以使用数组表示高精度的计算结果。例如1/3＝0.3333…333，保存到数组中如图4.3所示。

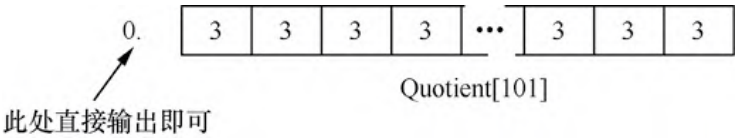


图4.3

进行除法运算时可以模拟人的手动操作，即每次求出商的第一位后，将余数乘以10，再计算商的下一位，重复以上过程。例如1/3的计算过程为：将余数即分子1×10/3＝3余1，将3存入Quotient数组后，余数1继续1×10/3＝3余1的操作即可。

核心伪代码如下所示。

```
1 int Quotient[101];            //Quotient[]依次存放商的每一位
2 循环100次
3 {
4     余数（分子）扩大10倍
5     将分子/分母得到的商依次存到Quotient[]中
6     分子/分母的余数为新的分子
7 }
```

4.1.13　求分数精确值2（exact）

【题目描述】

使用数组精确计算M/N（$0<M<N \leqslant 100$）的值。如果M/N是无限循环小数，则计算并输出它的第一循环节，同时要求输出循环节的起止位置（小数位的序号）。

【输入格式】

输入两个整数，即分子和分母。

【输出格式】

输出结果，如果是无限循环小数，输出循环节的起始位置和结束位置。

【输入样例】

77 78

【输出样例】

0.9871794

from 2 to 7

【算法分析】

因为*M*和*N*不超过100，所以*M*/*N*要么一定会被除尽，要么在小数点后100位内一定有循环节。分以下两种情况讨论。

（1）如果某次计算后的余数为0，则表示*M*/*N*为有限不循环小数。

（2）如果是无限循环小数，如何判断循环节呢？显然是当某次计算后的余数与前面的某个余数相同，则*M*/*N*为无限循环小数，且从该余数第一次出现到目前循环所求得的每一位商就是小数的循环节。所以可以用数组Remainder[101]存放第几次循环出现了某余数，例如第3次循环出现了余数17，则Remainder[17]＝3，这样当第i次循环再次出现余数17时，循环节的起止位置即from 3 to i。

参考伪代码如下所示。

```
 1 定义Remainder[101]，Remainder[x]=y表示在第y次循环时出现了余数x
 2 循环100次
 3 {
 4   余数存放在Remainder[]的相应位置上
 5   余数扩大10倍
 6   输出当前循环算出的商
 7   更新余数
 8   if(余数为0)
 9       退出循环
10   if(该余数对应的位在前面已经出现过)
11   {
12     输出循环节的起止位置
13     退出循环
14   }
15 }
```

4.1.14　复杂除式还原（div2）

【题目描述】

如图4.4所示，除式中仅有一个数字7可见，其他画“×”的位置的数字全部被破坏了，请还原该除式。

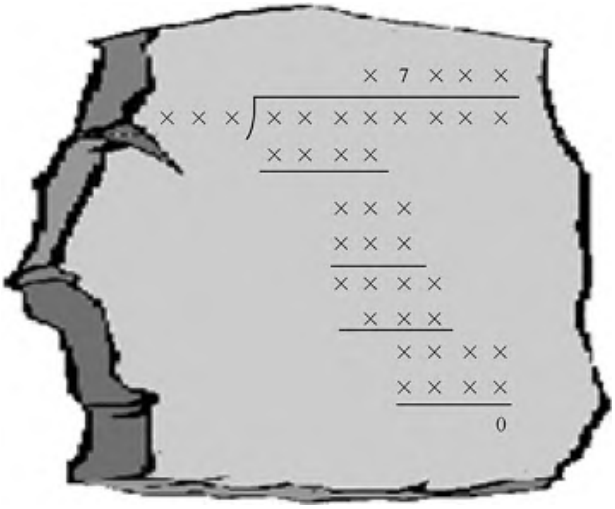


图4.4

【输入格式】

无输入。

【输出格式】

标准输出，输出算式，即被除数/除数=商。

【输入样例】

无。

【输出样例】

24/2=12

【样例说明】

输出样例仅为示例，并非正确答案。



这道题不能用单纯的枚举法求解，一是因为计算时间太长，二是因为难以求出除式中各部分的值，所以需要对除式进行分析，尽可能多地发现限制条件。

分析除式，预处理如图4.5所示。

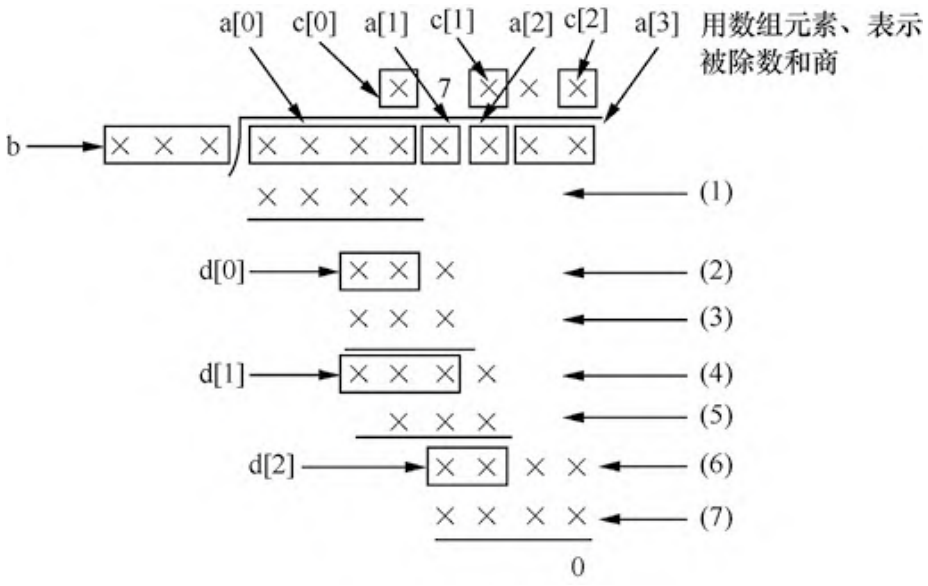


图4.5

由（3）可以看出，商的第2位7乘除数得一个三位数，所以除数小于等于142。

由除数乘商的第1位为四位数可知，商的第1位只能为8或9，且除数大于等于112。同时商的第5位也为8或9，前4位一定小于等于142×9＋99且大于等于1 000＋10。这是因为前4位最大可能是142×9，但是又考虑到（2）中余下来一个两位数，所以还可以加上99。又因为前4位一定是四位数，其大于等于1 000，但同样考虑到（2），所以加上10。

由（4）～（6）可以看出，（4)的前2位一定为10，（5）的第1位一定为9，（6）的前2位一定为10～99，商的第4位一定为0。

由（5）的第1位一定为9和$112 \leqslant$除数$\leqslant 142$可知：商的第3位可能为7或8。

由（1）可知：除数×商的第1位应当为四位数。

由（5）可知：除数×商的第3位应当为三位数。

编程时为了方便，将被除数分解：前4位用a[0]表示，第5位用a[1]表示，第6位用a[2]表示，第7、8位用a[3]表示。除数用变量*b*表示。将商分解：第1位用c[0]表示，第3位用c[1]表示，第4位是0，第5位用c[2]表示。其他部分的商分别表示为：（2）的前2位用d[0]表示，（4）的前3位用d[1]表示，（6）的前2位用d[2]表示。将上述分析用数学的方法综合起来可以表示如下。

被除数：$1\,010 \leqslant a[0] \leqslant 1\,377\;0 \leqslant a[1] \leqslant 9$

$0 \leqslant a[2] \leqslant 9 \qquad 0 \leqslant a[3] \leqslant 99$

除数：$112 \leqslant b \leqslant 142$

商：$8 \leqslant c[0] \leqslant 9 \quad 7 \leqslant c[1] \leqslant 8 \quad 8 \leqslant c[2] \leqslant 9$

（2）的前2位：$10 \leqslant d[0] \leqslant 99$

（4）的前3位：$100 \leqslant d[1] < b$

（6）的前2位：$10 \leqslant d[2] \leqslant 99$

（1）式需满足：$b \times c[0] > 1\ 000$

（5）式需满足：$100 < b \times c[1] < 1\ 000$

4.1.15　求质数（prime）

【题目描述】

质数是组成一切自然数的基本元素，例如7是由1个2和1个5组成的。试编程求出
100 000以内的质数。

【输入格式】

无输入。

【输出格式】

输出100 000以内的质数，数与数之间以一个空格间隔，行末无空格，有换行。

【输入样例】

无。

【输出样例】

2 3 5 7 ……

【算法分析】

对于较大范围的质数的判断，可使用时间复杂度为$O(n\log n)$的埃氏筛法。

方法是先把N个自然数按从小到大的顺序排列存入数组（实际编程以数组下标表示N个自然数）。1不是质数也不是合数，要划去，如图4.6所示。



图4.6

第二个数2是质数要留下来，把2后面所有能被2整除的数都划去，如图4.7所示。

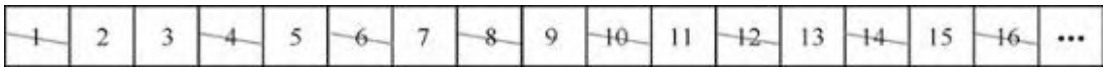


图4.7

把2后面第一个没划去的数3留下来，从9（因为6已经被划去了）开始，把所有能被3整除的数都划去，如图4.8所示。

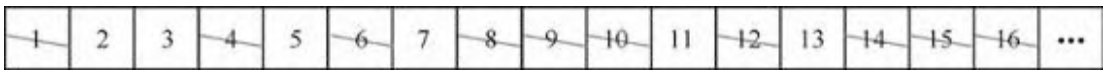


图4.8

把3后面第一个没划去的数5留下来，从25开始，把所有能被5整除的数都划去……这样一直下去，就会把不超过N的全部合数都划掉，留

下的就是不超过N的全部质数。

参考程序如下所示。

```
 1 //求质数—埃氏筛法
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4 const int MAX=100001;
 5
 6 bool a[MAX+10];                          //全局数组，初始值均为false即0
 7 int prime[MAX/3],Count=0;                //prime[]用于存质数，Count用于统
计质数个数
 8
 9 int main()
10 {
11   for (int i=2,limit=sqrt(MAX); i<MAX; i++)
12   {
13     if (a[i]==0)                          //a[i]=0表示a[i]是质数
14       prime[Count++]=i;                   //质数依次存入prime[]
15     if (limit>i)                          //限制，否则i*i可能会过大而溢出
16       for (int j=i*i; j<MAX; j+=i)        //从i*i开始，因为i*(2～i-1)已删
除
17         a[j]=1;                           //标记a[j]不是质数
18   }
19   for (int i=0; i<Count; i++)
20     cout<<prime[i]<<' ';
21   return 0;
22 }
```

可以发现某些数被筛选了多次，例如12被2和3各筛选了1次。继续优化的方法是使用时间复杂度为$O(n)$的欧拉线性筛算法。

欧拉线性筛算法基于“整数唯一分解定理”，即任何一个大于1的自然数N可以分解成唯一的有限个质数的乘积，例如12＝2×2×3、18＝2×3×3。并可以得出：任一合数＝最小质因数×最大因数（非自身）。欧拉线性筛算法由此在埃氏筛法的基础上，让每个合数只被它最小的质因数筛选一次，以达到不重复筛选的目的。

参考程序如下所示。

```
 1 //求质数—欧拉线性筛算法
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4 const int MAXN = 100001;
 5
 6 bool a[MAXN];                 //a[i]=0表示a[i]是质数
 7 int prime[MAXN/3], Count;     //prime[]用于存质数，Count用于统计质数
个数
 8
 9 int main()
10 {
11   for (int i=2; i<MAXN; i++)
12   {
13     if (a[i]==0)               //如果a[i]是质数
14       prime[Count++] = i;     //将此质数按顺序存到prime[]
15     for (int j=0; j<Count && i*prime[j]<MAXN; j++)//枚举筛出的质
数
16     {
17       a[i*prime[j]]=1;        //标记为合数
18       if (i%prime[j]==0)       //此处是优化的关键
19         break;                //保证每个合数被它最小的质因数筛选一次
20     }
21   }
22   for (int i=0; i<Count; i++)
23     cout<<prime[i]<<' ';
24   return 0;
25 }
```

该程序每次用已筛出来的质数乘i去删除合数，并筛出更大的质数。例如：

当i=2时，筛出质数2，删除合数4，即质数2×2；

当i=3时，筛出质数3，删除合数6和9，即质数2×3和质数3×3；

当i=4时，只删除合数8后即退出，因为4能被2整除，然后向下执行i=5……

为什么不继续执行质数3×4，并删除合数12呢？这是因为4能被2整除，说明后面凡是4乘以某个因数得到的合数的最小质因数都是2，如果用后面更大的质数例如3去筛选，就不符合让每个合数只被它最小的质因数筛选一次，以达到不重复筛选的目的了。

那么12在什么时候被删除呢？答案是当i=6的时候，即质数2×6，因为2是12的最小质因数。显然这种算法不会重复筛选，也不会漏选。

4.2　二维数组



扫码看视频

二维数组可以分解为多个一维数组。如图4.9所示，定义一个二维数组a[3][4]，该数组名为a，是一个3行4列的数组，可以将它看作由3个一维数组a[0]、a[1]、a[2]组成，每个一维数组又包含4个数组元素。

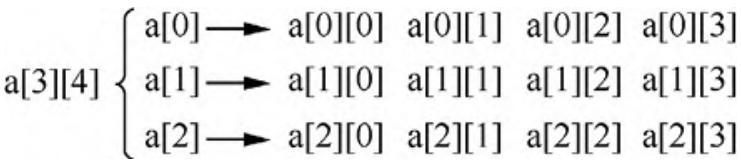


图4.9

注意：a[0]、a[1]、a[2]在二维数组里是一维数组名，而在之前学到的一维数组的相关内容中，a[0]、a[1]、a[2]是单个的数组元素。

C++语言中，二维数组中数组元素排列的顺序是按行存储的，即在内存中先顺序存储第一行的数组元素，再存储第二行的数组元素……如图4.10所示。

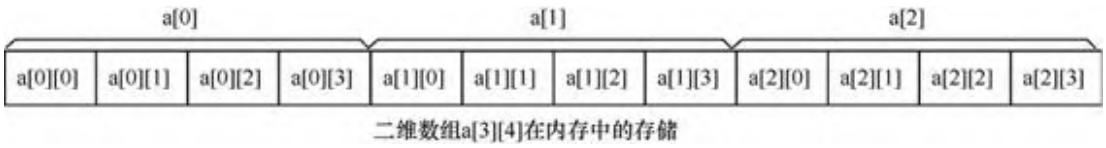


图4.10

C++语言中还可以定义多维数组，例如定义三维数组 a[2][3][4]，多维数组的数组元素在内存中的排列顺序为：第一维的下标变化最慢，最右边的下标变化最快。例如，a[2][3][4]的数组元素排列顺序如图4.11所示。

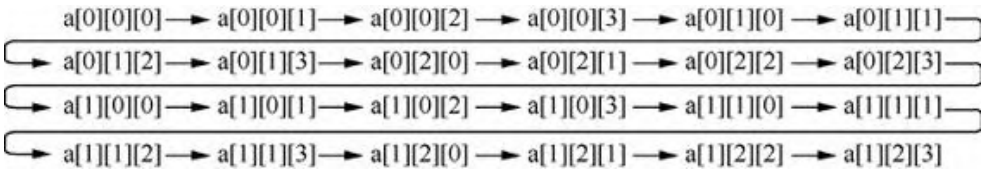


图4.11



不管是一维数组还是多维数组，下标都是从0开始的。当我们定义int a[3][4]后，它可用的行下标值最大应为2，列下标值最大应为3，即最后一个数组元素为a[2][3]。如果有类似a[3][4]=3这样的语句就是错误的，因为a[3][4]已经超过了数组的范围。

很多C++编译器对数组下标的检查不是非常严格，即使数组元素下标值“越界”，仍然可以编译通过，这是C++语言自由度较大的一个弊端。

可以对二维数组以如下方式赋值。

```
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};

int b[ ][4]={1,2,3,4,5,6,7,8,9,10,11,12}; //第一维的下标值可省略，第二维的下标值不可省略

int c[ ][4]={{0,0,3},{ },{0,10}};      //未赋值的数组元素均自动初始化为0
```

4.2.1　最强魔法师（powerful）

【题目描述】

M×*N*个魔法师依次站在一个*M*行*N*列的矩阵中，已知他们的魔法力，试编程输出最强魔法师的魔法力和他所在的行号和列号。

【输入格式】

第一行两个数*M*和*N*（1＜*M*，*N*＜100），即行号和列号。第一行以下为*M*行，每行有*N*个数，表示每个魔法师的魔法力，保证没有相同的数。

【输出格式】

输出最大值、最大值所在的行、最大值所在的列。

【输入样例】

2 2

1 2

3 4

【输出样例】

max=4 row=2 column=2

参考程序如下所示，请根据题目要求自行更改。

```
 1 //最强魔法师
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a[3][4]= {{1,2,3,4},{5,6,7,8},{-10,-3,-4,4}};
 8   int i,j,Row=0,Column=0,MAX=a[0][0];      //假设a[0][0]为最大值
 9   for (i=0; i<=2; i++)                    //必须两层循环，一层循环枚
举行
10     for (j=0; j<=3; j++)                  //一层循环枚举列
11       if (a[i][j]>MAX)
12       {
13         MAX=a[i][j];                      //更新为更大值
14         Row=i;                            //更新为最大值所在的行
15         Column=j;                          //更新为最大值所在的列
16       }
17   printf("max=%d row=%d column=%d\n",MAX,Row+1,Column+1);
18   return 0;
19 }
```

4.2.2 矩阵加法（matrix1）

【题目描述】

由$m\times n$个数 a_{ij}（$i=1,2,\cdots,m$; $j=1,2,\cdots,n$）排成的m行n列的数表，

例如$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$称为$m\times n$的矩阵，记作

$\boldsymbol{A}=\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$。

例如：$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$是2行3列的矩阵，或称为2×3矩阵。

两个矩阵的行数和列数分别相等时，可以相加、相减。两个矩阵相加、相减时，结果还是矩阵，它的各个元素等于两个矩阵对应元素相加、相减所得的值。例如：

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}+\begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 1 & 2 & 4 \end{bmatrix}=\begin{bmatrix} 2 & 4 & 6 \\ 7 & 7 & 7 \\ 8 & 10 & 13 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}-\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}=\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

现输入两个m行n列的矩阵A和B，输出它们的和$A+B$。

【输入格式】

第一行包含两个整数m和n（$1\leqslant m$，$n\leqslant 100$），分别表示矩阵的行数和列数。

接下来m行，每行n个整数，表示矩阵A的元素。

接下来m行，每行n个整数，表示矩阵B的元素。

【输出格式】

输出m行n列的矩阵，表示$A+B$的结果。

【输入样例】

```
2 3
1 2 3
4 5 6
1 2 3
4 5 6
```

【输出样例】

```
2 4 6
8 10 12
```

4.2.3 杨辉三角（YangHui）

【题目描述】

图4.12所示的杨辉三角是二项式系数在三角形中的一种几何排列。你可以找出其排列规律并编程输出*N*行数字吗？

【输入格式】

输入一个整数*N*（1＜*N*＜15）。

【输出格式】

输出*N*行的杨辉三角，两数间以空格间隔。

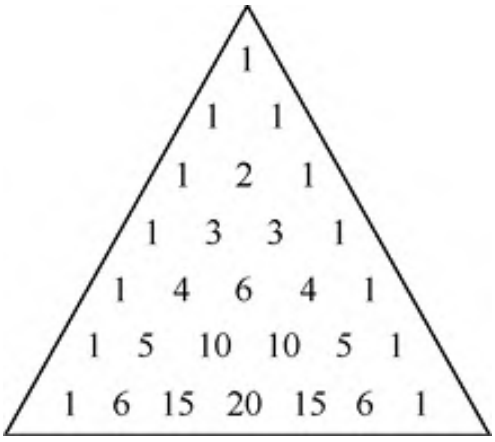


图4.12

【输入样例】

3

【输出样例】

1

1 1

1 2 1

4.2.4 矩阵转置（matrix3）

【题目描述】

将矩阵A的行和列进行调换，形成一个新的矩阵，记作转置矩阵$A^{\top}$。例如

$\boldsymbol{A}=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}$，则 $\boldsymbol{A}^{\top}=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$。

试编程输入一个m行n列的矩阵A，输出它的转置矩阵。

【输入格式】

第一行为两个整数m和n（$1 \leqslant m$，$n \leqslant 100$），表示矩阵A的行数和列数。

接下来为m行n列的矩阵A。

【输出格式】

输出矩阵*A*的转置矩阵。

【输入样例】

2 3

1 2 3

4 5 6

【输出样例】

1 4

2 5

3 6

4.2.5 扫雷游戏（game）

【题目描述】

在*m*行*n*列的雷区中有一些格子含有地雷（称为地雷格），其他格子不含地雷（称为非地雷格）。玩家翻开一个非地雷格时，该格将会出现一个数字提示周围格子中有多少个是地雷格。游戏的目标是在不翻出任何地雷格的条件下，找出所有的非地雷格。

已知雷区的地雷分布，要求计算出每个非地雷格周围的地雷格数，一个格子的周围格子包括其上、下、左、右、左上、右上、左下、右下8个方向上与之直接相邻的格子。

【输入格式】

第一行输入两个整数m和n（$1 \leqslant m$，$n \leqslant 100$），分别表示雷区的行数和列数。

接下来为m行，每行n个字符，描述了雷区中的地雷分布情况。字符“*”表示相应格子是地雷格，字符“?”表示相应格子是非地雷格。相邻字符之间无分隔符。

【输出格式】

输出m行，每行n个字符，用以描述整个雷区。用字符“*”表示地雷格，用周围的地雷个数表示非地雷格。相邻字符之间无分隔符。



【输入样例】

3 3

*??

???

?*?

【输出样例】

*10

221

1*1

4.2.6 矩阵乘法（matrix2）

【题目描述】

当第1个矩阵*A*的列数等于第2个矩阵*B*的行数时，这两个矩阵可以相乘，*n*×*m*的矩阵与

m×*k*的矩阵相乘变成*n*×*k*的矩阵，其乘积矩阵*A*×*B*的第*i*行第*j*列的元素为矩阵*A*第*i*行上的*m*个数与矩阵*B*第*j*列上的*m*个数对应相乘后所得的*n*个乘积数之和。如图4.13所示，C[1][1]＝A[1][0]×B[0][1]＋A[1][1]×B[1][1]＋A[1][2]×B[2][1]。

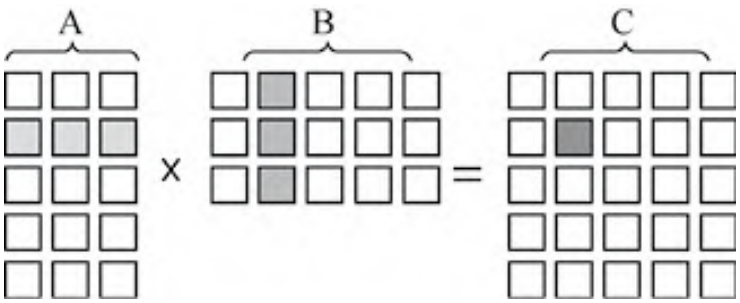


图4.13

例如已知 $\boldsymbol{A}=\begin{bmatrix}1&4\\2&5\\3&6\end{bmatrix}$，$\boldsymbol{B}=\begin{bmatrix}1&2&3\\4&5&6\end{bmatrix}$，则乘积矩阵$A\times B$的计算过程如下。

$$\boldsymbol{A}\times\boldsymbol{B}=\begin{bmatrix}1\times1+4\times4&1\times2+4\times5&1\times3+4\times6\\2\times1+5\times4&2\times2+5\times5&2\times3+5\times6\\3\times1+6\times4&3\times2+6\times5&3\times3+6\times6\end{bmatrix}=\begin{bmatrix}17&22&27\\22&29&36\\27&36&45\end{bmatrix}$$

现输入一个n行m列的矩阵A和一个m行k列的矩阵B，输出$A\times B$的矩阵。

【输入格式】

第一行输入3个整数，分别为n、m、k。

接下来输入A矩阵和B矩阵，矩阵中每个元素值的绝对值不超过1000。

【输出格式】

输出$A\times B$的矩阵，两数之间以空格间隔。

【输入样例】

```
3 2 3
1 4
2 5
3 6
1 2 3
4 5 6
```

【输出样例】

```
17 22 27
22 29 36
27 36 45
```

4.2.7　神奇矩阵

【题目描述】

如图4.14所示，所谓神奇矩阵即将1～9这 9 个数排成 3 行 3 列，使其行、列、对角线上3个数之和均相同。试编程输出所有神奇矩阵的排列方式。

2	9	4
7	5	3
6	1	8

图4.14



这道题最容易想到的算法是枚举法，但其时间复杂度高达$O(n^3)$。

```
for(i=159;i<=951;i++)                    //枚举第1行数
  for(j=159;j<=951;j++)                  //枚举第2行数
    for(k=159;k<=951;k++)                //枚举第3行数
      {
      将i、j、k这3个三位数拆成9个一位数
      判断3行上的3个数之和是不是15
```

判断3列上的3个数之和是不是15

判断两个对角线上的3个数之和是不是15

判断9个数是否重复　　//思考代码如何实现

}

进一步的优化是将每一行看作一个三位数，首先找出所有三位上的数之和为15，且三位上的数各不相同的三位数依次存入数组ary[]。

再通过枚举的方法从ary[]中任取3个数，为方便起见，将放在第1行的数编号为1，放在第2行的数编号为2，放在第3行的数编号为3，显然这3行数共有6种排列方式，分别为(1,2,3)、(1,3,2)、(2,1,3)、(2,3,1)、(3,1,2)及(3,2,1)。

因为从3行数中取1行数放在第1行有3种方法，再取1行数放在第2行有2种方法，最后取1行数放在第3行有1种方法，由乘法原理得3×2×1＝6（种）方法。更进一步，记A_n^m表示从n个不同的元素中任取m（$m \leqslant n$）个元素，计算公式为$A_n^m = n(n-1)(n-2)\cdots(n-m+1)$。

参考程序如下所示。

```
1 //神奇矩阵
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
```

```
 6 {
 7   int a,b,c,n=0,ary[100];
 8   for (int i=159; i<=951; i++)//找出所有三位上的数之和为15且各不相同
的三位数
 9   {
10     a=i/100;
//百位
11     b=i/10%10;
//十位
12     c=i%10;
//个位
13     if ((a+b+c==15) && (a!=c) && (a!=b) && (b!=c) &&
(a*b*c!=0))
14       ary[n++]=i;                          //依次存入ary[]数组
15   }
16   int a1,b1,c1,a2,b2,c2,a3,b3,c3;
17   for (int i=0; i<n; i++)                   //枚举，i的取值范围为0～
n-1
18     for (int j=i+1; j<n; j++)               //j从i+1开始，保证j不等于
i
19       for (int k=j+1; k<n; k++)             //k从j+1开始，保证k不等于
i、j
20       {
21         a1=ary[i]/100;b1=ary[i]/10%10;c1=ary[i]%10;
22         a2=ary[j]/100;b2=ary[j]/10%10;c2=ary[j]%10;
23         a3=ary[k]/100;b3=ary[k]/10%10;c3=ary[k]%10;
24         if ((a1+a2+a3==15) && (b1+b2+b3==15) && (c1+c2+c3==15)
&&
25             a1*b1*c1*a2*b2*c2*a3*b3*c3==362880)
26         {                                          //测试所有的行排列组
合
27           if ((a1+b2+c3)==15 && (c1+b2+a3==15))
//123
28             printf("%d\n%d\n%d\n\n",ary[i],ary[j],ary[k]);
29           if ((a1+b3+c2)==15 && (c1+b3+a2==15))
//132
30             printf("%d\n%d\n%d\n\n",ary[i],ary[k],ary[j]);
31           if ((a2+b1+c3)==15 && (c2+b1+a3==15))
//213
32             printf("%d\n%d\n%d\n\n",ary[j],ary[i],ary[k]);
33           if ((a2+b3+c1)==15 && (c2+b3+a1==15))
//231
34             printf("%d\n%d\n%d\n\n",ary[j],ary[k],ary[i]);
35           if ((a3+b2+c1)==15 && (c3+b2+a1==15))
//321
36             printf("%d\n%d\n%d\n\n",ary[k],ary[j],ary[i]);
37           if ((a3+b1+c2)==15 && (c3+b1+a2==15))
//312
38             printf("%d\n%d\n%d\n\n",ary[k],ary[i],ary[j]);
```

```
39          }
40        }
41    return 0;
42 }
```

继续深入，其实根据奇偶性质，满足条件的只有以下一种情况，如图4.15所示。

偶	奇	偶
奇	奇	奇
偶	奇	偶

图4.15

而偶数仅有2、4、6、8这4个数，正好占据了矩阵的4个角。分析到这里，完全可以根据现有的矩阵填充的数学规律，直接填写出满足条件的一个答案，如图4.16所示。

8	1	6
3	5	7
4	9	2

图4.16

图4.16的填充规律是：在第1行中间填1，然后1的右上角即2的位置（行越界则移到最后一行），2的右上角即3的位置（列越界则移到第1列）……依次类推。一直到碰到已经填好的数为止（例如3碰到了

右上角的1），此时数向下填。填好后将此序列进行图4.17所示的旋转90度和上下翻转即可。

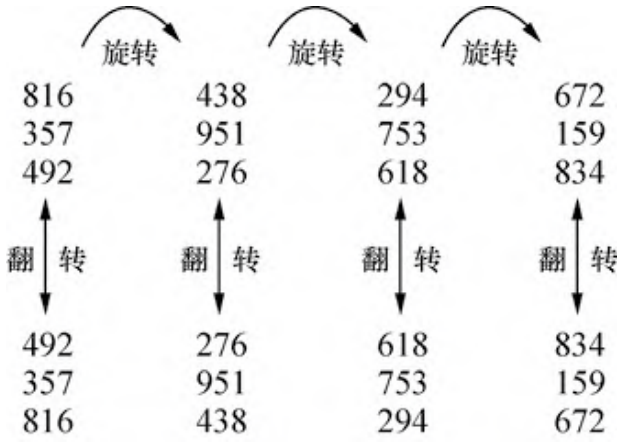


图4.17

参考程序如下所示。

```
 1 //神奇矩阵
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int a[3][3],b[3][3];                    //全局数组，值全部自动初始化为0
 6
 7 int main()
 8 {
 9   int x=0,y=1;
10   a[0][1]=1;                                          //填入1
11   for (int i=2; i<=9; i++)                            //依次填入2～9
12   {
13     int tx=(x+2)%3;
14     int ty=(y+1)%3;
15     if (a[tx][ty]==0)                                 //如果当前位置为
空
16     {
17       a[tx][ty]=i;                                    //填数
18       x=tx;
19       y=ty;
20     }
21     else                                              //当前位置已有数
22     {
```

```
23        x=(x+1)%3;                                          //向下放置
24        a[x][y]=i;
25      }
26    }
27    for (int i=0; i<=3; i++)
28    {
29      printf("%d%d%d\n",a[0][0],a[0][1],a[0][2]);
30      printf("%d%d%d\n",a[1][0],a[1][1],a[1][2]);
31      printf("%d%d%d\n\n",a[2][0],a[2][1],a[2][2]);
32      printf("%d%d%d\n",a[2][0],a[2][1],a[2][2]);    //上下翻转
33      printf("%d%d%d\n",a[1][0],a[1][1],a[1][2]);
34      printf("%d%d%d\n\n",a[0][0],a[0][1],a[0][2]);
35      for (int ii=0; ii<3; ii++)                          //旋转到辅助数
组
36        for (int jj=0; jj<3; jj++)
37          b[jj][2-ii]=a[ii][jj];
38      for (int ii=0; ii<3; ii++)
39        for (int jj=0; jj<3; jj++)
40          a[ii][jj]=b[ii][jj];
41    }
42    return 0;
43  }
```

4.2.8　蛇形矩阵1（snake1）

【题目描述】

在$n\times n$方阵里逆时针填入1、2、3、……，共$n\times n$个数，构成一个蛇形矩阵。例如当$n=3$
时，蛇形矩阵为：

7　8　1

6　9　2

5　4　3

【输入格式】

输入一个正整数n（$1<n<100$）。

【输出格式】

输出$n\times n$的蛇形矩阵，每个数的存储空间为5个字符，右对齐。

【输入样例】

3

【输出样例】

7 8 1

6 9 2

5 4 3

参考程序如下所示。

```
1 //蛇形矩阵1
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
6 {
7   int x=1,y,n,num=1;
8   int a[101][101]= {0};
9   cin>>n;
```

```
10    y=n;
11    a[x][y]=num;                          //把1填入第一行最后一列
12    while (num<n*n)                       //当矩阵未填满时
13    {
14      while (x+1<=n && !a[x+1][y])        //x+1未越界且下方未填充，则下移
15        a[++x][y]=++num;
16      while (y-1>0 && !a[x][y-1])         //y-1未越界且左方未填充，则左移
17        a[x][--y]=++num;
18      while (x-1>0 && !a[x-1][y])         //x-1未越界且上方未填充，则上移
19        a[--x][y]=++num;
20      while (y+1<=n && !a[x][y+1])        //y+1未越界且右方未填充，则右移
21        a[x][++y]=++num;
22    }
23    for (int i=1; i<=n; ++i)              //输出矩阵
24    {
25      for (int j=1; j<=n; ++j)
26        printf("%5d",a[i][j]);
27      printf("\n");
28    }
29    return 0;
30  }
```

【笔试测验】请手动计算程序运行的结果。

```
 1  #include<bits/stdc++.h>
 2  using namespace std;
 3
 4  int main()
 5  {
 6    int a[6][6];
 7    for (int i=1; i<=5; i++)
 8      for (int j=1; j<=5; j++)
 9        a[i][j]=(i>=j?j:i);
10    for (int i=1; i<=5; i++)
11    {
12      for (int j=1; j<=5; j++)
13        cout<<a[i][j];
14      cout<<endl;
15    }
16    return 0;
17  }
```

4.2.9 回型方阵（matrix）

【题目描述】

输入一个正整数n，输出$n\times n$的回型方阵（行数与列数相等的矩阵）。例如当$n=5$时，输出示例如下。

1 1 1 1 1

1 2 2 2 1

1 2 3 2 1

1 2 2 2 1

1 1 1 1 1

【输入格式】

输入一个正整数n（$2\leqslant n\leqslant 9$）。

【输出格式】

输出n行，每行包含n个正整数，每个数的存储空间为5个字符，右对齐。

【输入样例】

4

【输出样例】

1 1 1 1

1 2 2 1

1 2 2 1

1 1 1 1

4.2.10 蛇形矩阵2（snake）

【题目描述】

输入一个正整数n，输出$n\times n$的蛇形矩阵。例如当$n=5$时，输出示例如下。

15 7 6 2 1

16 14 8 5 3

22 17 13 9 4

23 21 18 12 10

25 24 20 19 11

【输入格式】

输入一个正整数n（$2 \leqslant n \leqslant 50$）。

【输出格式】

输出n行，每行包含n个正整数，每个数的存储空间为5个字符，右对齐。

【输入样例】

4

【输出样例】

7 6 2 1

13 8 5 3

14 12 9 4

16 15 11 10

4.2.11 蛇形矩阵3（snake）

【题目描述】

取*n*行*n*列的蛇形矩阵（其中*n*为不超过100的奇数），在矩阵中心从1开始以逆时针方向绕行，逐圈扩大，直到*n*行*n*列填满数字。图4.18所示即3行3列的蛇形矩阵。请输出*n*行*n*列的蛇形矩阵和矩阵的对角线数字之和。

5	4	3
6	1	2
7	8	9

图4.18

【输入格式】

输入一个正整数*n*（*n*行*n*列）。

【输出格式】

输出*n*＋1行，前*n*行为组成的蛇形矩阵，最后一行为对角线数字之和。

【输入样例】

3

【输出样例】

5 4 3

6 1 2

7 8 9

25

4.3 字符数组



扫码看视频

用于存放字符的数组是字符数组，例如char c[5]可存放5个字符，依次赋值如下。

```
c[0]='H';c[1]='E';c[2]='L';c[3]='L';c[4]='O'; //单个字符要用单引号标注
```

则其在内存中的保存形式如图4.19所示。

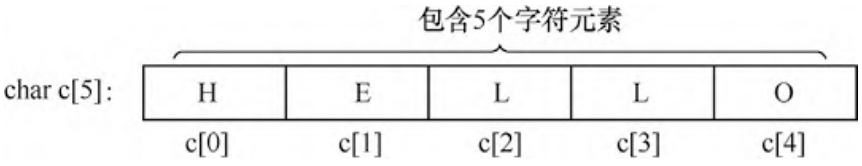


图4.19

定义字符数组时，可同时对数组元素赋以初始值，例如char c[5]={'H','E','L','L','O'};。

定义和初始化二维字符数组的示例如下。

```
char a[3][3]={{'a','b','c'}, {'1','2','3'}, {'*','',''}};
```

输出字符数组的程序如下所示。

```
 1 //输出字符数组
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   char c[5]= {'H','e','l','l','o'};
 8   for (int i=0; i<5; i++)
 9     cout<<c[i];                              //一个元素一个元素地输出
10   return 0;
11 }
```



为方便起见，可以使用字符串常量初始化字符数组。例如：

char c[]={"I am ok"};//定义字符数组时全部元素均赋值，数组大小定义可省略

也可以省略花括号，直接写成char c[]="I am ok"; 。

这两种初始化形式与下面的初始化形式等价。

char c[]={'I',' ','a','m',' ','o','k','\0'};//'\0'为字符串结束标记

但不与char c[]={'I',' ','a','m',' ','o','k'}等价。

前者的字符长度为8，后者的字符长度为7。

由于系统在字符串常量的末尾自动添加隐含的'\0'表示字符串结束,因此为了使字符数组和字符串的处理方法一致，在字符数组中也常常人为地添加'\0'。如char c[]={'o','k','!',''\0'};。

常用字符串处理函数介绍如下。

1. gets()和puts()

gets()可以输入整行字符串（包含空格），而cin()或scanf()在输入时遇到空格就结束输入。puts()的功能是输出字符串（行末自动换行）。例如有下面3行代码。

```
char str[30];
gets(str);
puts(str);              //输出该行字符串后会自动换行
```

运行时用键盘输入“I am ok”，结果是将此字符序列的7个字符和'\0'字符共同存入str并输出（行末自动换行）。



gets()不检查字符串str的大小，必须遇到换行符或在文件结尾处才会结束输入，因此如果输入的字符串长度超过了字符数组的长度，就会导致缓存溢出使程序崩溃。所以请谨慎使用该函数，C++11标准里已将该函数废除。

2.cin.getline()

cin.getline()可以接收带空格的字符串，它有3个参数，即cin.getline(接收字符串的变量,接收字符个数,结束字符)，当第3个参数省略时，系统默认为'\0'。

参考程序如下所示。

```
 1 //cin.getline()的使用
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   char a[40];             //实际只能输入39个字符，最后一位用于标记结束
 8   cin.getline(a,39);      //例如输入a b c d e f g
 9   cout<<a<<endl;          //输出a b c d e f g
10   cin.getline(a,39,'e');//例如输入a b c d e f g
11   cout<<a<<endl;          //因为'e'为结束字符，所以输出a b c d
12   return 0;
13 }
```

3.strlen()

strlen(字符串)的功能是返回字符串的长度，不包括字符串结束标记'\0'。

参考程序如下所示。

```
 1 //strlen()的使用
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   char a[300];
 8   cin.getline(a,299);
 9   int n=strlen(a);
10   for (int i=0; i<n; ++i)
11     cout<<a[i];
12   return 0;
13 }
```

4.3.1 求逆序字符串（reverse）

【题目描述】

输入一行字符串，将它逆序赋值给另一个字符串后顺序输出。

【输入格式】

输入一行字符串（不超过100个字符）。

【输出格式】

输出它的逆序字符串。

【输入样例】

abc

【输出样例】

cba

```
 1 //求逆序字符串
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4 const int N=100;                                  //定义常量N=100
 5
 6 int main()
 7 {
 8   char s1[N],s2[N];
 9   cin.getline(s1,N-1);
10   int len=strlen(s1);
11   for (int i=len-1; i>=0; i--)
12     s2[len-i-1]=s1[i];                            //反转字符串赋值
13   s2[len]='\0';                                   //末尾添加字符串结束标记
14   puts(s2);
15   return 0;
16 }
```

4.3.2　求数的和（sumN）

【题目描述】

输入一个整数N，求各位上的数的和。

【输入格式】

输入一个整数*N*，*N*最多200位。

【输出格式】

输出一个整数，即整数*N*各位上的数的和。

【输入样例】

12345000000000000000000000000000000000000000000000000000000000001

【输出样例】

16

【算法分析】

因为整数*N*的位数过多，所以只能使用一维字符数组读取、存储整数*N*。

4.3.3　查找子串（findchar）

【题目描述】

请编写一个程序，判定一个字符串是不是另一个字符串的一部分（即子串）。例如，图4.20所示的主字符串为aababcabcdabcde，子串为abcd，则位置为6。

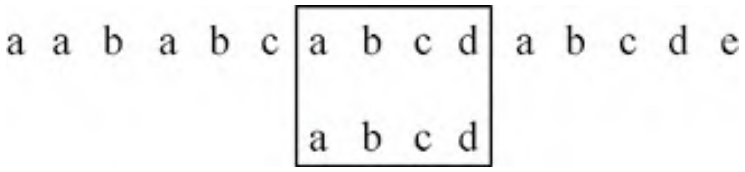


图4.20

【输入格式】

输入两个字符串，各占一行，其中第一行为主字符串。

【输出格式】

输出子串所在的位置，若未找到则输出-1。

【输入样例】

aababcabcdabcde

abcd

【输出样例】

6

4.3.4　字符串游戏（string）

【题目描述】

有一个由小写字母构成的字符串和一堆卡片，每张卡片上写着一个字母，可以取出若干张卡片，覆盖黑板上的一些字母（也可以一张都不取）。

请编程找出覆盖黑板上的字母之后字典序最大的字符串。所谓字典序，就是基于字母顺序排列的单词按字母顺序排列的方法，例如aa、ab、ba、bb、bc就是按顺序从小到大排好的字典序。

【输入格式】

第一行输入由小写字母构成的字符串（字符串长度$\leqslant 50$），即黑板上的字符串。

第二行输入由小写字母构成的字符串（字符串长度$\leqslant 50$），即若干张卡片。

【输出格式】

输出一行，即覆盖黑板上的字母之后字典序最大的字符串。

【输入样例】

abcdefg

abc

【输出样例】

cbcdefg

【算法分析】

将卡片上的字母按ASCII值由大到小排好序后，从左到右依次比较黑板上的字母，若覆盖后得到的字符串的字典序更大，则覆盖黑板上的字母。

4.3.5 柱状图（chart）

【题目描述】

读取4行字符，然后用柱状图输出每个字符出现的次数。

【输入格式】

输入4行字符，无小写字母，每行不超过100个字符。

【输出格式】

由若干行组成，前几行由空格和“*”组成，最后一行则由空格和字母组成。任何一行末尾无空格，有换行，不要打印任何空行。

【输入样例】

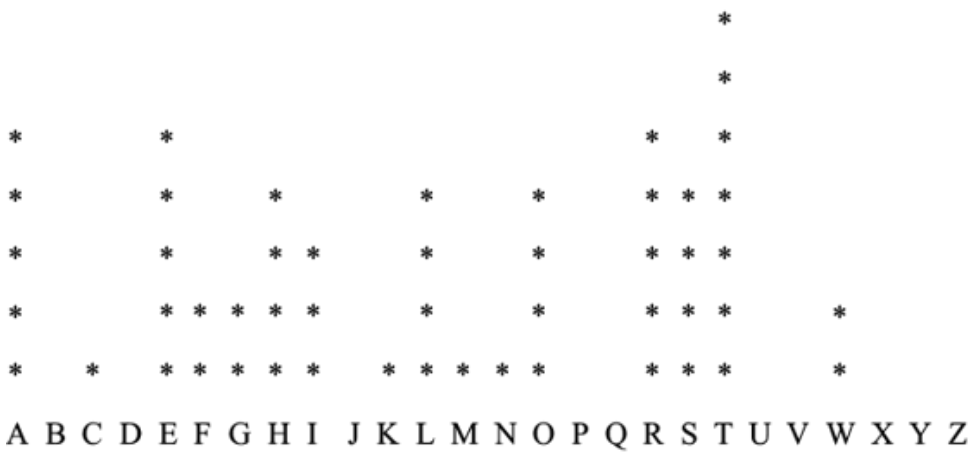
GREAT OAKS FROM LITTLE

ACORNS GROW.

THE FIRST WEALTH

IS HEALTH .

【输出样例】



4.strcmp()

两个字符数组不能直接比较大小，应使用strcmp()函数。strcmp(字符串1,字符串2)用于比较两个字符串，比较规则是从左到右

依次按字符的ASCII值的大小比较，一旦某两个字符ASCII值大小不同，则立即结束比较，返回一个整数。若字符串1等于字符串2，则返回0；若字符串1小于字符串2，则返回一个负整数；若字符串1大于字符串2，则返回一个正整数。

例如：

```
char str1[]="Hello!";
char str2[]="hello!";
cout<<strcmp(str1,str2);
```

4.3.6 字符串排序（charsort）

【题目描述】

输入10个字符串（长度不超过20个字符），将所有小写字母转为大写字母后按ASCII值从小到大排序后输出。

参考程序如下所示。

```
 1 //字符串排序
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   char a[10][21];
 8   for (int i=0; i<=9; ++i)
 9   {
10     cin.getline(a[i],20);
11     for (int j=0; j<strlen(a[i]); j++)
12       if (a[i][j]>='a' && a[i][j]<='z')
13         a[i][j]-=32;
14   }
15   for (int i=0; i<9; ++i)                                    //冒泡排序
16     for (int j=0; j<9-i; ++j)
```

```
17        if (strcmp(a[j],a[j+1])>0)
18          swap(a[j],a[j+1]);
19    for (int i=0; i<=9; ++i)
20      puts(a[i]);
21    return 0;
22  }
```

5. strcpy()

不能用赋值语句将字符串常量直接赋给字符数组，但可以使用strcpy()函数进行赋值。strcpy(字符数组1,字符串2)是将“字符串2”复制到“字符数组1”中。例如str="abcde";是错误的，应改为strcpy(str,"abcde");。

6. strcat()

strcat(字符数组1,字符数组2)是把“字符数组2”连接到“字符数组1”的后面，结果放在“字符数组1”中。例如：

```
char a[]="12345", b[]="67890";
strcat(a,b);
cout<<a;                          //输出1234567890
```

4.3.7　最大整数（BigNum）

【题目描述】

设有*n*个正整数（不超过整数的取值范围），将它们连接成一排组成一个最大的多位正整数。

例如n=3时，13、312、343这3个正整数连接成的最大正整数为34 331 213。

例如n=4时，7、13、4、246这4个正整数连接成的最大正整数为7 424 613。

【输入格式】

第一行为一个正整数n（$n \leqslant 20$）。

第二行为n个正整数。

【输出格式】

输出一个正整数，表示连接成的最大正整数。

【输入样例】

3

13 312 343

【输出样例】

34331213

【算法分析】

将n个数以字符串的方式进行处理。比较方法是以两个数连接后的大小来决定数的先后顺序。

例如有两个数a和b，如果a＋b＞b＋a，则a排在前面，否则a排在后面。

参考程序如下所示。

```
 1 //最大整数
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   char s[101][30],t1[65],t2[65];
 8   int n;
 9   scanf("%d",&n);
10   for (int i=1; i<=n; i++)
11     scanf("%s",&s[i]);
12   for (int i=1; i<n; i++)                          //冒泡排序
13     for (int j=1; j<=n-i; j++)
14     {
15       strcpy(t1,s[j]);
16       strcpy(t2,s[j+1]);
17       strcat(t1,s[j+1]);                          //s[i]+s[j+1]
18       strcat(t2,s[j]);                            //s[j+1]+s[i]
19       if (strcmp(t1,t2)<0)
20         swap(s[j],s[j+1]);
21     }
22   for (int i=1; i<=n; ++i)
23     printf("%s",s[i]);
24   printf("\n");
25   return 0;
26 }
```

7. strstr()

strstr(str1,str2)用于判断字符串str2是不是字符串str1的子串。如果是则该函数返回str2在str1中首次出现的地址，否则返回0（空指针NULL）。例如：

```
char c1[100],c2[100];
scanf("%s%s",c1,c2);
printf("十六进制地址值为:0x%x",strstr(c1,c2));//十六进制用0x标识，%x表示输出十六进制数
```

4.3.8 统计单词数（stat）

【题目描述】

给定一个单词，请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，即给定的单词必须与文章中的某一独立单词完全相同。（题目来源：NOIP 2011普及组）

【输入格式】

第一行为字符串，其中只含字母，表示给定的单词。

第二行为字符串，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的单词则输出两个整数，两个整数之间用一个空格隔开，分别表示单词在文章中出现的次数和第一次出现的位置（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）；如果单词没有在文章中出现，则输出整数-1（给定的单词仅是文章中某一单词的一部分不算匹配）。

【输入样例1】

```
To

to be or not to be is a question
```

【输出样例1】

```
2 0
```

【输入样例2】

```
to

Did the Ottoman Empire lose its power at that time
```

【输出样例2】

```
-1
```

【算法分析】

为便于查找，首先将两个字符串中的字母统一转为大写字母或小写字母。

使用strstr(str1,str2)在字符串str1中查找字符串str2的位置，获得的是地址值，将此地址值减去字符串的首地址值，即可知位于字符串str1的第几个字符。

每查找到一个匹配的位置，就将从字符串开头到匹配位置的内容删除（使用strcpy()实现，虽耗时但易理解），以便能再次使用strstr(str1,str2)查找下一个匹配的位置，但下一次查找到的位置并不是在之前完整的原始字符串中查找的，所以查找到的位置需要加上之前删除的字符串长度才是原始字符串中的真正位置。

参考程序如下所示（本程序仅为演示之前几个字符串函数的使用，并不是最优算法，部分大数据会超时，请自行考虑更简单的算法，后文会有针对string类操作的算法讲解）。

```
 1 //统计单词数
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4 char str1[1000010],str2[21];
 5
 6 int main()
 7 {
 8   cin.getline(str2,20);
 9   cin.getline(str1,1000000);
10   int len1=strlen(str1);
11   int len2=strlen(str2);
12   for (int i=0; i<len1; i++) //全转为大写字母
13     if (str1[i]>='a' && str1[i]<='z')
14       str1[i]-=32;
```

```
15   for (int i=0; i<len2; i++) //全转为大写字母
16     if (str2[i]>='a' && str2[i]<='z')
17       str2[i]-=32;
18   strcat(str2," ");//后面加空格降低匹配难度（因为单词以空格间隔）
19   strcat(str1," ");//后面加空格降低匹配难度
20   int first=-1;//first:第一个匹配位置， -1表示还没开始匹配
21   int count=0, p=0,Len=0;//Len统计前面已经删除了多少个字符
22   while (strstr(str1,str2))//如果str1包含str2，即可以匹配
23   {
24     p=strstr(str1,str2)-str1;//包含str2的地址-数组首地址，即偏移量
25     if ((p==0 || str1[p-1]==' ') && first==-1)//第一次匹配的处理
26       first=p+Len,count++;//first要还原回初始的正确位置
27     else  if(str1[p-1]==' ')//因为单词以空格间隔
28       count++;
29     strcpy(str1,str1+p+1);//将str1后面一段复制，即前面一段删除(耗时)
30     Len+=p+1;//每次str1前面删除的字符串长度都累计到Len
31   }
32   if (count==0)
33     cout<<"-1\n";
34   else
35     cout<<count<<' '<<first<<endl;
36   return 0;
37 }
```

实际上，之前提到的string类使用起来比字符数组更为方便。例如判断两个string类的字符串s和s1的子串问题，string类提供的部分相关函数如下。

s.find(s1)：查找s中第一次出现s1的位置，如果找不到就返回-1。

s.find(s1,p)：从s的第p个位置开始查找第一次出现s1的位置，找不到就返回-1。

参考程序如下所示。

```
1 //统计单词数——用string类处理
2 #include <bits/stdc++.h>
```

```
 3 using namespace std;
 4
 5 int main()
 6 {
 7   string a,b;            //string不是基本数据类型，不能用scanf()读取
 8   getline(cin,a);        //读取带空格的字符串到string，推荐用
getline()
 9   getline(cin,b);        //读取不带空格的字符串到string，直接用cin>>
10   for (int i=0; i<a.length(); ++i)
11     a[i]=tolower(a[i]); //转小写字母，转大写字母为toupper
12   for (int i=0; i<b.length(); ++i)
13     b[i]=tolower(b[i]);
14   a=' '+a+' ';           //单词前后加空格，可以用运算符"+"实现
15   b=' '+b+' ';           //文章同步前后加空格
16   if (b.find(a)==-1)     //先测试是否有匹配位置，找不到值为-1
17     cout<<-1<<endl;
18   else
19   {
20     int first=b.find(a),count=0;
21     for (int P=b.find(a); P<b.length(); P=b.find(a,P+1))
22       ++count;
23     cout<<count<<" "<<first<<endl;
24   }
25   return 0;
26 }
```

上面程序的第14、15行只用了操作符“+”就将字符串连接了起来。string类的操作符如表4.1所示。

表4.1

操作符	示例	注释
+	s+t	将字符串s和t连接成新字符串，注意先后顺序
=	s=t	用字符串t更新字符串s

操作符	示例	注释
+=	s+=t	等价于s=s+t
==	s==t	判断字符串s与字符串t是否相同
!=	s!=t	判断字符串s与字符串t是否不相同
<	s<t	判断字符串s是否小于字符串t（按ASCII值比较）
<=	s<=t	判断字符串s是否小于等于字符串t（按ASCII值比较）
>	s>t	判断字符串s是否大于字符串t（按ASCII值比较）
>=	s>=t	判断字符串s是否大于等于字符串t（按ASCII值比较）
[ ]	s[i]	访问字符串中下标为i的字符

使用了string类的部分操作符的参考程序如下所示。

```
 1 //string类的部分操作符的使用
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   string s1,s2="abc";              //定义s1为空字符串，s2赋值为"abc"
 8   cin>>s1;
 9   cout<<(s1==s2 ? "s1==s2":"s1!=s2")<<endl;
10   cout<<(s1>=s2 ? "s1>=s2":"s1<s2" )<<endl;
```

```
11    s1="new";                                  //s1更新为"new"
12    s1=s1+'_';                                 //s1更新为"new_"
13    s1+=s2;                                    //s2连接到s1的后面,s1
为"new_abc"
14    for (int i=s1.length()-1; i>=0; i--)
15      cout<<s1[i];                             //逆序逐个输出各元素即"cba_wen"
16    return 0;
17 }
```

string类的用法还有很多，此处仅列出了一些常见的用法。需要注意的是：虽然string类使用起来非常方便，但是其在时间效率上是不如字符数组的。

【笔试测验】请手动算出下面程序执行时输入I am a magician.的结果。

```
 1 #include <bits/stdc++.h>
 2 using namespace std;
 3
 4 int main()
 5 {
 6   string str;
 7   getline(cin, str);
 8   int len=str.size();  //size()和length()一样,都是统计字符串的元素个
数
 9   for (int i=0; i<len; i++)
10     if(str[i]>='a' && str[i]<='z')
11       str[i]=str[i]-'a'+'A';
12   cout<<str<<endl;
13   return 0;
 14 }
```

4.3.9 子串包含问题（substr）

【题目描述】

输入两个字符串s1和s2，要求判断其中一个字符串是不是另一个字符串通过若干次循环移位后的新字符串的子串。循环移位是指将字符串的第一个字符移动到末尾形成新的字符串。例如CDMA是MAUVCD两次移位后产生的新字符串UVCDMA的子串，而CDMA与AMCD则无论如何移位也不可能实现。

【输入格式】

输入两个字符串。

【输出格式】

如果第一个字符串是第二个字符串通过若干次循环移位后的新字符串的子串，则输出“Yes”，否则输出“No”。

【输入样例】

CDMA

MAUVCD

【输出样例】

Yes

【算法分析】

将长字符串自身复制一次，例如“MAUVCD”变为“MAUVCDMAUVCD”后查找子串即可。

参考程序如下所示。

```
1 //子串包含问题
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
6 {
7   string s1,s2;
8   cin>>s1>>s2;
9   if (s1.size()<s2.size())     //比较两个字符串的长度，将长字符串放在
前面
10     swap(s1,s2);
11  s1+=s1;                      //复制s1
12  cout<<(s1.find(s2)==-1?"No":"Yes")<<'\n';
13  return 0;
14 }
```

4.3.10　命名（namenum）

【题目描述】

母牛们都有手机了，它们将身上烙的编号按标准手机数字按键的排列转换成对应的字母后为彼此取名。手机按键的数字分别对应下面字母之一（除了“Q”和“Z”）。

2：A，B，C。　　5：J，K，L。　　8：T，U，V。

3：D，E，F。　　6：M，N，O。　　9：W，X，Y。

4：G，H，I。　　7：P，R，S。

母牛们喜欢的名字都在一个按字典序排列的列表里，列表中所有名字均为大写，试从列表中找出编号对应的所有有效名字。例如编号25可以转换成AJ、AK、AL、BJ、BK、BL、CJ、CK及CL。而列表中只能找到AK和AL，所以有效的名字就是AK和AL。

【输入格式】

第一行是给出的编号，第二行是列表的名字数量*n*，随后为*n*行，每行一个大写的名字。

【输出格式】

按字典序输出有效名字的不重复列表，一行一个名字。如果没有则输出“NONE”。

【输入样例】

25

5

AB

AK

AL

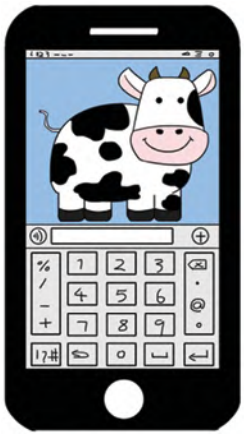
BA

BB

【输出样例】

AK

AL



使用“内容市场”
APP扫描看视频

4.3.11　古风排版（typesetting）

【题目描述】

古人书写文字是从右向左竖向排版的。请编写程序，把一段文字按古风格式排版。

【输入格式】

第一行输入一个正整数N（N≤100），表示每一列的字符数。第二行输入一个长度不超过
1 000的非空字符串，以换行结束。

【输出格式】

按古风格式排版给定的字符串，每列N个字符（最后一列可能不足N个）。

【输入样例】

```
4
This is a test case
```

【输出样例】

```
asa T
st ih
e tsi
 ce s
```

【算法分析】

首先要计算出显示列数，再按照一定的规则将字符串依次存入二维字符数组的相应位置输出即可。

4.4　滚动数组



扫码看视频

4.4.1　截铁丝（line）

【题目描述】

现有长度为*total*的铁丝，要将之截成*n*小段（*n*＞2），每段的长度不小于1，如果其中任意3小段铁丝都不能拼成三角形，则*n*的最大值为多少？

【输入格式】

输入一个整数*total*（5＜*total*＜150 000 000）。

【输出格式】

输出一个整数n。

【输入样例】

144

【输出样例】

10

【算法分析】

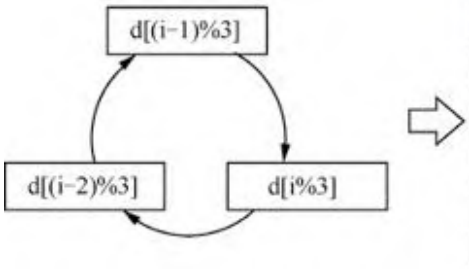
只有任何两边之和大于第三边才可以构成一个三角形，因此无法构成三角形的条件就是任意两边之和不超过第三边。

题目要求截尽可能多的铁丝段，所以初始时应该是两个长度最小为1的铁丝段，而第三段铁丝的长度应该是2（为了使得n最大，则要使剩下来的铁丝尽可能长，所以每一段铁丝的长度应该是前面的相邻两段铁丝的长度的和）。显然这是一个斐波那契数列，依次为1、1、2、3、5、8、13、21、34、55，以上各数之和为143，与144相差1。因此如果铁丝长度为144，则最后一段铁丝的长度可以取56。

普通写法的参考代码如下所示。

```
 1 //截铁丝——普通写法
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   long long d[100]= {1,1};
 8   int total,i,sum=2;                    //sum为已截的铁丝长度和，初始为2
 9   scanf("%d",&total);
10   for (i=2; ; i++)
11   {
12     d[i]=d[i-1]+d[i-2];
13     if (total-sum>=d[i])
14       sum+=d[i];                        //已经截去的铁丝总长
15     else
16       break;
17   }
18   printf("%d\n",i);
19   return 0;
20 }
```

d[]数组使用了100个存储单元的空间，但其实程序运行时始终只有d[i]、d[i-1]及d[i-2]3个数组元素参与计算。因此可以定义一个只有3个数组元素的数组d[3]，将d[i]＝d[i-1]＋d[i-2]改成d[i%3]＝d[(i-1)%3]＋d[(i-2)%3]滚动循环计算，因为对3取模的结果永远在0、1、2这3个数中循环。如图4.21所示，当i=2时，对应d[2]＝d[1]＋d[0]；当i=3时，对应d[0]＝d[2]＋d[1] ……



	d[i%3]=d[(i-1)%3]+d[(i-2)%3]
i=2	d[2] = d[1] + d[0]
i=3	d[0] = d[2] + d[1]
i=4	d[1] = d[0] + d[2]
i=5	d[2] = d[1] + d[0]
i=6	d[0] = d[2] + d[1]
…	… … …

图4.21

使用滚动数组的代码如下所示。可以看到，滚动数组节约了空间，但效率不高。

```
 1 //截铁丝——滚动数组
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   long long d[3]= {1,1};
 8   int total,i,sum=2;              //sum为已截的铁丝长度和，初始为2
 9   scanf("%d",&total);
10   for (i=2; ; i++)
11   {
12     d[i%3]=d[(i-1)%3]+d[(i-2)%3];
13     if (total-sum>=d[i%3])
14       sum+=d[i%3];                //更新已经截去的铁丝总长
15     else
16       break;
17   }
18   printf("%d\n",i);
19   return 0;
20 }
```

4.4.2　太阳能电池（battery）

【题目描述】

太空站需要将1×1和2×2两种规格的太阳能电池不重叠地铺满*n*×3的电池板，求有多少

种不同的铺设方案。

【输入格式】

输入一个整数n（0＜n＜110 000）。

【输出格式】

输出一个整数，即铺设方案数%12 345的值。

【输入样例】

2

【输出样例】

3

【算法分析】

用f(n)表示对于n×3的电池板有多少种不同的铺设方案，容易得到f(0)＝1、f(1)＝1、
f(2)＝3。那么f(3)是多少呢？

显然，f(3)是由f(1)和f(2)的铺设方案数递推而来的，分以下两种情况考虑：

（1）由f(2)推导时，只有将第3行用3块1×1的电池铺设的一个方案；

（2）由f(1)推导时，只有将第2行和第3行用1块2×2的电池和2块1×1的电池铺设的两个方案（只用1×1的电池铺设的方案与f(2)中的方案重复）。

根据加法原理和乘法原理，得递推关系式f(i)＝f(i-1)＋f(i-2)*2。参考程序如下所示。

```
 1 //太阳能电池
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4 #define F(i) f[(i)%3]   //定义程序中的f[(i)%3]用F(i)表示，i为变量
 5
 6 int main()
 7 {
 8   int n;
 9   int f[3]= {1,1};
10   scanf("%d",&n);
11   for (int i=2; i<=n; i++)
12     F(i)=(F(i-1)+F(i-2)*2)%12345;
13   printf("%d\n",F(n));
14   return 0;
15 }
```

滚动数组更多地应用于二维数组，试观察下面的程序段。

```
 1 for (int i=n-1; i>=1; i--)
 2   for (int j=1; j<=i; j++)
 3     f[i][j]+=max(f[i-1][j],f[i-1][j-1]); //max(a,b)可得a、b的最大值
```

程序在运行时，始终只有两行数据参与计算，即下一行的值根据上一行的值来确定。因此，为节省程序运行空间，可以只定义一个由两个一维数组组成的二维数组，例如f[2][n]，采取边读取边计算的方式。计算的时候，对数组下标i和i-1对2取模实现数组的滚动操作，即f[i%2][j]=max(f[(i-1)%2][j],f[(i-1)%2][j-1])。

为方便起见，也可以改成预处理指令：

```
#define F(i,j) f[(i)%2][j]    //定义程序中的f[(i)%2][j]用F(i，j)表示
```

则F(i,j) = max(F(i-1,j)，F(i-1,j-1))。

滚动数组的运行过程如图4.22所示。

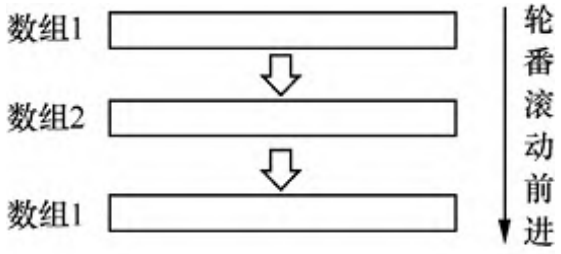


图4.22

第05章　阶段检测1

5.1　笔试检测

（1）请手动算出下面程序执行的结果。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int a[]= {1, 2, 3, 4, 5, 6, 7, 8},pa=0,pb=7;
  while (pa<pb)
  {
    int t=a[pa];
    a[pa]=a[pb];
    a[pb]=t;
    pa++,pb--;
  }
  do
    cout<<a[pa]<<' ';
  while(pa--);
  return 0;
}
```

（2）请手动算出下面程序执行时输入5 2 94 -7 0 19 82的结果。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int d[10], i, j, cnt, n, m;
  scanf("%d %d\n", &n, &m);
  for (i=1; i<=n; i++)
    scanf("%d", &d[i]);
  for (i=1; i<=n; i++)
```

```
11   {
12     cnt=0;
13     for (j=1; j<=n; j++)
14       if ((d[i]<d[j]) || (d[j]==d[i] && j<i))
15         cnt++;
16       if (cnt==m)
17          printf("%d\n", d[i]);
18   }
19   return 0;
20 }
```

（3）请手动算出下面程序执行的结果。

```
 1 #include <bits/stdc++.h>
 2 using namespace std;
 3
 4 int main()
 5 {
 6   int a[3],b[3]= {2,3,5};
 7   for (int i=0; i<3; ++i)
 8   {
 9     a[i]=0;
10     for (int j=0; j<=i; ++j)
11     {
12       a[i]+=b[j];
13       b[a[i]%3]+=a[j];
14     }
15   }
16   int ans=1;
17   for (int i=0; i<3; ++i)
18   {
19     a[i]%=10;
20     b[i]%=10;
21     ans*=a[i]+b[i];
22   }
23   cout << ans << endl;
24   return 0;
25 }
```

（4）请手动算出下面程序执行时输入Believe in yourself的结果（提示：空格符的ASCII码值为32，'0'的ASCII码值为48，'A'的ASCII码值为65，'a'的ASCII码值为97）。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  string s;
  getline(cin, s);
  char m1 = ' ',m2 = ' ';
  for (int i=0; i<s.length(); i++)
    if (s[i]>m1)
    {
      m2=m1;
      m1=s[i];
    }
    else if (s[i]>m2)
      m2=s[i];
  cout<<int(m1)<<' '<<int(m2)<<endl;
  return 0;
}
```

（5）请手动算出下面程序执行的结果。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int a[4][4]=
{1,2,-3,-4,0,-12,-13,14,-21,23,0,-24,-31,32,-33,0};
  int i,j,s=0;
  for (i=1; i<4; i++)
    for (j=0; j<4; j++)
    {
      if (a[i][j]<0) continue;
      if (a[i][j]==0) break;
      s+=a[i][j];
    }
  cout<<s<<endl;
  return 0;
}
```

5.2 上机检测

扫码看视频

5.2.1　猴子吃桃（peach）

【题目描述】

一只猴子第一天摘下若干个桃子，当即吃掉一半，还不过瘾，又多吃了一个。第二天早上猴子又将剩下的桃子吃掉一半，并又多吃了一个。以后每天早上猴子都吃掉前一天剩下的桃子的一半再多一个。到第10天早上猴子想再吃桃子时，只剩一个桃子了。求第一天共摘了多少桃子？



使用“内容市场”
APP扫描看视频

【输入格式】

无输入。

【输出格式】

输出桃子个数。

【输入样例】

无。

【输出样例】

略。

5.2.2　卖宝石（stone）

【题目描述】

一堆宝石分5次出售，第1次卖出全部的一半加1/2个；第2次卖出余下的1/3加1/3个；第3次卖出余下的1/4加1/4个；第4次卖出余下的1/5加1/5个；最后卖出余下的11个。问原来共有多少宝石？

【输入格式】

无输入。

【输出格式】

输出宝石数。

【输入样例】

无。

【输出样例】

略。

5.2.3　取石子游戏（stone）

【题目描述】

A和B两人玩取石子游戏，一共有两堆石子，每次可以选择一堆取任意个石子，当一个人没有石子取时就输了。

A和B都非常聪明，每次取石子都是按最优策略选择，假设A先取，问最后谁能赢。

【输入格式】

输入两个数x、y分别表示两堆石子的数量（$x \leqslant 100\,000$，$y \leqslant 100\,000$）。

【输出格式】

输出赢家的名字，即输出“A”或者“B”。

【输入样例】

1 2

【输出样例】

A

5.2.4 打电话（telephone）

【题目描述】

通信公司规定通话时间小于等于3分钟话费都是a元，大于3分钟则每分钟按b元收费，

小光有t元，问最多能打多少分钟电话。

【输入格式】

输入3个整数a、b、t（$1 \leqslant a,b \leqslant 100, 1 \leqslant t \leqslant 10\,000$）。

【输出格式】

输出一个整数，即最多分钟数。

【输入样例】

2 1 4

【输出样例】

6

5.2.5 神秘字符串（ride）

【题目描述】

你需要将两个字符串均以下列方式转换成一个数字，最终数字就是字符串中所有字母的积，其中字母A～Z对应数字1～26。例如，USACO就是21×19×1×3×15＝17 955。如果两个字符串转换为数字后% 47的值相等则输出“GO”，否则输出“STAY”。

【输入格式】

第一行为长度为1～6的大写字母的字符串，第二行为长度为1～6的大写字母的字符串。

【输出格式】

输出“GO”或“STAY”。

【输入样例】

COMETQ

HVNGAT

【输出样例】

GO

5.2.6　合法C++标识符（c）

【题目描述】

给定一个不包含空格符的字符串，请判断其是否为C++语言合法的标识符。

C++语言标识符的要求如下。

（1）非关键字（注：题目保证这些字符串一定不是C++语言的关键字）。

（2）只包含字母、数字及下划线（_）。

（3）不以数字开头。

【输入格式】

输入一个字符串，字符串中不包含任何空格符，且长度不大于20。

【输出格式】

如果它是C++语言的合法标识符，则输出“yes”，否则输出“no”。

【输入样例】

123You

【输出样例】

no

5.2.7　求立方根（cuberoot）

【题目描述】

输入一个正整数a，求a的立方根。已知求立方根的迭代公式为$x_{n+1}=x_n\times 2/3+a/(3\times x_n\times x_n)$，其中$x_0=a$。

【输入格式】

输入一个整数a。

【输出格式】

输出a的立方根，保留小数点后3位。

【输入样例】

27

【输出样例】

3.000

迭代公式是这样使用的：例如求18的立方根，将$x_0=a$代入迭代公式得$x_1=12.018\ 5$，再将$x_1=12.018\ 5$代入迭代公式得$x_2=8.053\ 88$，如此循环，直到x_{n+1}和x_n的值无限逼近（例如差值小于0.000 01）时，x_{n+1}即所求的答案。

5.2.8 插入排序（insert）

【题目描述】

如图5.1所示，插入排序类似于玩扑克时抓牌的过程，玩家每拿到一张牌都要插入手中已有的牌，使之从小到大排好序。

现使用排好序的数组模拟插入排序，即输入一个数时，要求按从小到大的排序规律将它插入数组中。



图5.1

【输入格式】

输入共3行，第1行为一个数N （$N \leqslant 10000$），表示原数组元素的个数；第2行为N个数，

即原数组的各元素值；第3行为一个数x，即输入的数。

【输出格式】

输出一行，即排好序的数组，以空格间隔，行末无空格，有换行。

【输入样例】

10

1 2 3 4 5 6 8 9 10 11

7

【输出样例】

1 2 3 4 5 6 7 8 9 10 11

【算法分析】

以输入样例来说，从左到右扫描已排好序的数组，找到第一个大于7的数8，8所在位置是7要插入的位置，但是插入之前必须要把8暂存到一个临时变量例如t中，否则8就会被7覆盖；7插入后，8变成了要插入的数，要把8插入后面9的位置……显然这是一个相似的操作过程，如图5.2所示。

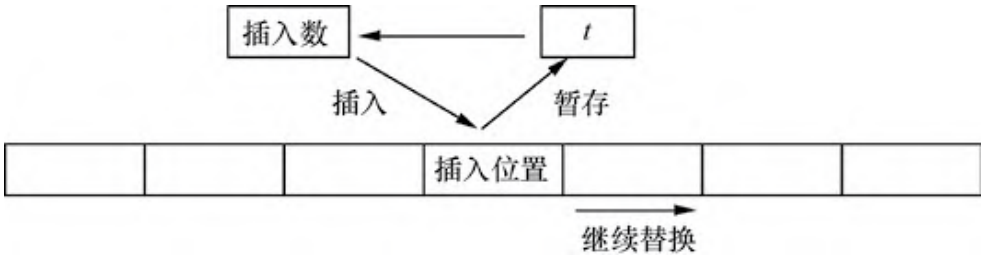


图5.2

或者从右向左扫描数组中的每个元素，若数组中的元素大于要插入的数，则将该数组元素后移一位，否则将插入的数放入之前因数组元素后移而空出的位置。

5.2.9 排名次（billing）

【题目描述】

对N个选手的成绩排名次，排名次需要考虑同分同名次的情况。

【输入格式】

第一行为数字N（$N \leqslant 1\,500$），第二行为N个选手的分数，以空格间隔。

【输出格式】

输出共N行，每一行两个整数，分别为成绩和名次。

【输入样例】

5

90 90 100 88 72

【输出样例】

90 2

90 2

100 1

88 4



这道题可以使用逆向思维，即假设每位选手为第一名，然后用每一位选手的成绩与所有选手的成绩比较，如果有成绩比这位选手高的，成绩低的选手的名次加1，最终可求出所有选手的名次，且同分的选手的名次相同。

5.2.10 ISBN（isbn）

【题目描述】

某图书馆的检索号包括9位数字、1位识别码和3位连接符，其格式如“x-xxx-xxxxx-x”，其中符号“-”是连接符（键盘上的减号），最后一位是识别码。例如0-670-82162-4就是一个标准的检索号。检索号的首位数字表示书籍的出版语言，例如0代表英语；第一个连接符“-”之后的3位数字代表出版社，例如670代表出版社；第二个连接“-”之后的5位数字代表该书在出版社的编号；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 % 11，所得的余数即识别码。如果余数为10，则识别码为大写字母X。例如检索号0-670-82162-4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,…,9，再求和，即0×1＋6×2＋…＋2×9＝158，然后取158 % 11的结果4作为识别码。

请编写程序判断输入的检索号中的识别码是否正确，如果正确，则仅输出“Right”；如果错误，则输出你认为是正确的检索号。

【输入格式】

输入一个字符序列，即一本书的检索号（保证符合检索号的格式要求）。

【输出格式】

输出共一行，假如输入的检索号的识别码正确，那么输出“Right”，否则按照规定的格式输出正确的检索号（包括连接符“-”）。

【输入样例】

0-670-82162-0

【输出样例】

0-670-82162-4

5.2.11　救援顺序（rescue）

【题目描述】

N个人被困在了M（$1 \leqslant M \leqslant 1\,000$）个山洞里，他们每个人都发送了一条信息告知了被困的山洞编号，救援人员的救援规则是：先去被困人数最多的山洞救援，若两个山洞被困人数相同，则优先去编号较小的山洞救援。

【输入格式】

第一行为一个整数N（$3 \leqslant N \leqslant 100\,000$），表示被困人数。

第二行为N个数，表示N个人被困的山洞编号。

【输出格式】

输出救援人员的救援顺序。

【输入样例】

8

5 5 5 3 3 2 2 1

【输出样例】

5->2->3->1

5.2.12 比例简化（ratio）

【题目描述】

为评出最佳选手进行了网络投票，例如对某一选手表示支持的有1498人，表示反对的有902人，那么支持与反对的比例可以简单地记为1498：902。

但是这个比例数值太大，很难一眼看出它们的关系，如果将比例记为5：3，虽然与真实结果有一定误差，但很直观，也能准确地反映调查结果。

现在给出支持人数A、反对人数B，以及上限L，请将A/B简化为A'/B'，要求在A'和B'均不大于上限L且A'和 B'互质（两个整数的最大公约数为1）的前提下，$A'/B' \geqslant A/B - A/B$的值尽可能小。

【输入格式】

输入一行3个正整数A、B、L，每两个正整数之间用一个空格隔开，分别表示支持人数、反对人数以及上限。其中，$1 \leqslant A \leqslant 1\,000\,000$，$1 \leqslant B \leqslant 1\,000\,000$，$1 \leqslant L \leqslant 100$，$A/B \leqslant L$。

【输出格式】

输出一行两个正整数A'和B'，中间用一个空格隔开，表示简化后的比例。

【输入样例】

1498 902 10

【输出样例】

5 3

5.2.13 黑色星期五（friday）

【题目描述】

传说若13日是星期五则不是一个“吉利”的日期。试编程计算从1900年1月1日起到1900＋n-1年12月31日为止，13日为星期一、星期二、……、星期日的次数。

已知1900年1月1日是星期一，4、6、9及11月有30天，其他月份除了2月都有31天。闰年2月有29天，平年2月有28天。

【输入格式】

输入一个整数*n*，*n*是一个非负数且不大于400。

【输出格式】

输出7个整数，它们代表13日是星期六、星期日、星期一、……、星期五的次数。

【输入样例】

20

【输出样例】

36 33 34 33 35 35 34

5.2.14 序列变换（change）

【题目描述】

有一个由n个0组成的数字序列需要通过两种操作将这个数字序列变换成目标序列a。

操作1：给$a_j, a_{j+1}, \cdots, a_n$都加上1。

操作2：给$a_j, a_{j+1}, \cdots, a_n$都减去1。

试计算至少需要操作多少次可以把原数字序列变换成目标序列。

【输入格式】

第一行为一个整数n（$1 \leqslant n \leqslant 200\,000$）。

第二行为n个整数，即目标序列a（$-10^9 \leqslant a_i \leqslant 10^9$）。

【输出格式】

输出最少的操作次数。

【输入样例】

5

1 2 3 4 5

【输出样例】

5

5.2.15　打猎（hunt）

【题目描述】

森林里有$M\times N$棵树，组成一个M行N列的矩阵，水平或垂直相邻的两棵树的距离为1，猎人和熊各在一棵树下。如果猎人与熊之间没有其他的树遮挡视线，猎人就可以开枪打到熊。

已知猎人和熊的位置，试判断熊所在的位置是否安全。

【输入格式】

第一行为n，表示有n（$n\leqslant 100\,000$）组数据，每组数据均为一行，分别为4个正整数，表示猎人和熊所在位置的x坐标和y坐标（$1\leqslant x\leqslant 100\,000\,000$，$1\leqslant y\leqslant 100\,000\,000$）。

【输出格式】

输出n行，每行为“yes”或“no”，表示熊的位置是否安全。

【输入样例】

1

1 1 2 3

【输出样例】

no

5.2.16 矩阵排序（sort）

【题目描述】

有多组$n\times m$的数字矩阵，需要将这个矩阵按行由小到大排序。矩阵的一行比另一行小，当且仅当这一行的字典序比另一行小。

【输入格式】

第一行两个整数n和m（$1 \leqslant n$，$m \leqslant 500$）。

之后为n行，每行m个0～100的整数。

【输出格式】

输出一个矩阵。

【输入样例】

3 5

8 1 8 5 3

5 4 4 6 7

1 2 3 8 3

【输出样例】

1 2 3 8 3

5 4 4 6 7

8 1 8 5 3

5.2.17 “语言之争”（language）

【题目描述】

Java与C++语言的最大区别之一在于标识符的命名。由多个单词组成的Java标识符的命名规则如下：第一个单词的首字母都是小写字母，接下来的其他单词都是大写字母开头，单词之间没有其他分隔符，例如longAndMnemonicIdentifier、name、nEERC都是符合规则的Java标识符。

C++标识符只用小写字母，单词之间用下划线分隔，例如c_identifier、long_and_mnemonic_identifier、name、n_e_e_r_c都是符合规则的C++标识符。

【输入格式】

输入一行不超过100个字符的只含有英文字母（或含英文字母和下划线）的一个标识符。

【输出格式】

如果输入的是Java标识符，则输出对应的C++标识符；如果输入的是C++标识符，则输出对应的Java标识符；如果都不是（或者都是），则输出“Error!”

【输入样例】

whenIFirstMeetU

【输出样例】

when_i_first_meet_u

【注意事项】

对于C++标识符：

（1）必须都是小写字母；

（2）注意“_”的位置；

（3）单词之间只能用 “_”分隔。

对于Java标识符：

（1）第一个单词首字母小写；

（2）单词之间没有“_”分隔。

两种规则混在一起的输出“Error!”（不包括引号）：

（1）既有大写字母又有“_”；

（2）符合两种规则情况。

在符合规则的情况下，一定要注意正确转换。



可定义两个布尔变量Upper和Under标记字符串中是否有大写字母和下划线,这样就可以根据Upper和Under的值判断出字符串是否符合规

则了，如果不符合规则，输出“Error!”退出程序，否则进行语言转换输出。

5.3　头脑风暴



扫码看视频



头脑风暴的目的是通过对题目的深入研究和讨论，使大家能够开拓思维、产生灵感，从而更自由地思考，以产生更好、更多的解决办法。

5.3.1　五位数

【题目描述】

有这样一些五位数，它的前两位和后两位能被6整除，中间一位也能被6整除，试编程计算这样的数有多少个？

【输入格式】

无输入。

【输出格式】

输出满足条件的数的个数。

【输入样例】

无。

【输出样例】

略。



这道题最简单的解答方法之一是枚举，但其实可以用更巧妙的方法来解答。

把五位数分成3个部分：前两位、中间一位及后面两位。

先分析前两位。由于首位不能为0，只能从1开始，因此前两位的情况有12, 18, 24, …, 96。显然这是个等差数列，使用等差数列公式$a_n = a_1 + (n-1) \times d$，变形后求得$n = (96-12) \div 6+1 = 15$。

再分析中间一位。中间一位能被6整除的数只能为0或6，有两种。

最后分析后面两位。和前两位类似，求0, 6, 12, …, 96的项数，为$(96-0) \div 6+1=17$。所以根据排列组合的乘法公式，所有情况为$15 \times 2 \times 17 = 510$（种）。

5.3.2　六位数

【题目描述】

一个六位数的个位数为7，现将个位数移至首位（十万位），而其余各位数顺序不变，均后移一位，得到一个新的六位数。假如新数为旧数的4倍，求原来的六位数。

这道题可以用数学方法求解。

设x为新数，y为旧数，可知：

（1）$x=4y$;

（2）$x=7\times100\ 000+(y-7)/10$。

因此$4y=700\ 000+(y-7)/10$，故$y=179\ 487$。

5.3.3 计算S的值（tower）

【题目描述】

输入正整数n，计算$S=1+(1+2)+(1+2+3)+\cdots+(1+2+3+\cdots+n)$。

【输入格式】

输入一个正整数n（$n<80$）。

【输出格式】

输出S的值。

【输入样例】

5

【输出样例】

35

这道题可以用数学方法求解。

$$1+2+3+\cdots+n=\frac{(n+1)n}{2}=\frac{n^2}{2}+\frac{n}{2}$$
$$S=\left(\frac{1^2}{2}+\frac{2^2}{2}+\frac{3^2}{2}+\cdots+\frac{n^2}{2}\right)+\left(\frac{1}{2}+\frac{2}{2}+\cdots+\frac{n}{2}\right)$$
$$=\frac{1^2+2^2+3^2+\cdots+n^2}{2}+\frac{n(n+1)}{4} \qquad (5\text{-}1)$$

$1^2+2^2+3^2+\cdots+n^2$可顺序排成左边的三角形（以n=8为例），根据左边的三角形可旋转得出右边的两个三角形。

现在我们发现3个三角形对应的每3个点相加的和均为（2n+1）。那么有多少个（2n+1）呢？显然根据等差数列公式即可求出，即n(n+1)/2。

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

则一个三角形的和应为：

$$1^2+2^2+3^2+\cdots+n^2=n(n+1)(2n+1)/6 \qquad (5\text{-}2)$$

将式（5-2）代入式（5-1），化简得$S=n(n+1)(n+2)/6$。

5.3.4　求0的个数（factorial）

【题目描述】

求1×2×3×…×N所得的数末尾有多少个0。（$1 \leqslant N \leqslant \max$，max为无符号整数类型最大值unsigned INT_MAX）

【输入格式】

输入共一行一个数，即N。

【输出格式】

输出共一行，即0的个数。

【输入样例】

5

【输出样例】

1



一般的方法是从1乘到N，每乘一次判断一次，若数的末尾有0则去掉0，并记下去掉的0的个数。并且为了不超出数的取值范围，去掉前面与0无关的数，只保留3位有效数。

```
//求N! 所得的数末尾的0的个数——方法1
#include <bits/stdc++.h>
using namespace std;

int main()
{
  int ans=0,n,sum=1;
  scanf("%d",&n);
  for (int i=1; i<=n; i++)
  {
    sum*=i;
    while (sum%10==0)                    //若数的末尾有0，则去掉0并计数
    {
      sum/=10;
      ans++;
    }
    sum%=1000;                           //保留3位有效数即可
  }
  printf("%d\n",ans);
  return 0;
}
```



进一步的优化：由于影响生成0的数只有2的倍数和5的倍数，这些数的分解数中含因子2的数多于含因子5的数，显然真正影响生成0的数是5的倍数。因此可以得出这样的结论：$N!$ 的分解数中有多少个因子5，末尾就有多少个0。这样可使外面的大循环的循环次数减少4/5。

```
1 //求N! 所得的数末尾的0的个数 —— 方法2，需N/5次循环
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
6 {
7   int ans=0,n;
8   scanf("%d",&n);
9   for (int j=5; j<=n; j+=5)                    //从5开始，步长为5
10    for (int i=j; i%5==0; ans++)               //i能被5整除，就说明有因子5
11      i/=5;
12  printf("%d\n",ans);
13  return 0;
14 }
```



可以使用“层层剥皮法”直接求5的个数：先以5为步长，执行一次循环，进行第一次剥皮，求出含5的个数（个数为*N*/5取整）；再以25为步长（因为有的数可以被5整除两次，例如25），执行第二次循环，进行第二次剥皮，求出含25的个数；再以125为步长……直到步长大于或等于*N*则退出循环。例如当*N*＝1 000时，公式为num＝1 000／5＋1 000／(5×5)＋1 000／(5×5×5)＋1 000／(5×5×5×5)。

```
1 //求N! 所得的数末尾的0的个数——方法3，仅需log5(N)次循环
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
6 {
```

```
 7    int ans=0,n;
 8    scanf("%d",&n);
 9    for (int i=n; i>=5; ans+=i)
10      i/=5;
11    printf("%d\n",ans);
12    return 0;
13  }
```

5.3.5　多项式求值（poly）

【题目描述】

输入x和系数$a_0, a_1, a_2, \cdots, a_n$，计算$a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \cdots + a_1 x + a_0$的值。

【输入格式】

第一行两个整数n（$1 \leqslant n \leqslant 10$）和$x$（$-5 \leqslant x \leqslant 5$）。

第二行$n+1$个整数，表示$a_0, a_1, \cdots, a_n$（$-10 \leqslant a_i \leqslant 10$）的值。

【输出格式】

输出一个整数，即多项式的值。

【输入样例】

3 2

1 2 3 4

【输出格式】

49

【算法分析】

一般地，一元n次多项式的求值需要经过$[n(n+1)]/2$次乘法和n次加法，而秦九韶公式只需要n次乘法和n次加法，大大简化了运算过程。推导过程如下。

$$
\begin{aligned}
&f(x)=a_nx^n+a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+\cdots+a_1x+a_0\\
&\rightarrow f(x)=\left(a_nx^{n-1}+a_{n-1}x^{n-2}+a_{n-2}x^{n-3}+\cdots+a_1\right)x+a_0\\
&\rightarrow f(x)=\left(\left(a_nx^{n-2}+a_{n-1}x^{n-3}+a_{n-2}x^{n-4}+\cdots+a_2\right)x+a_1\right)x+a_0\\
&\cdots\\
&\rightarrow f(x)=(\cdots((a_nx+a_{n-1})x+a_{n-2})x+\cdots+a_2)x+a_1)x+a_0
\end{aligned}
$$

参考程序如下所示。

```
 1 //多项式求值
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4 int a[20];
 5
 6 int main()
 7 {
 8   int n,x;
 9   cin>>n>>x;
10   for (int i=0; i<=n; i++)
11     cin>>a[i];
12   int ans=a[n];
13   for (int i=n-1; i>=0; i--)
14     ans=ans*x+a[i];
15   cout<<ans<<endl;
```

```
16    return 0;
17 }
```

5.3.6　二分法求方程的解（equation）

【题目描述】

假设有一组数据是按升序（降序同理）排序的，现查找给定值x。一种方法是从数列的中间位置开始比较，如果当前位置值等于x，则查找成功；若x小于当前位置值，则在数列的前半段中查找；若x大于当前位置值，则在数列的后半段中查找，直到找到为止。这就是所谓的二分法。

试用二分法原理求方程$2x^3-4x^2+3x-6=0$的解，其中x的取值范围为（-10，10）。

【输入格式】

无输入。

【输出格式】

输出一个浮点数，保留小数点后两位（不考虑四舍五入）。

【输入样例】

无。

【输出样例】

略。

【算法分析】

二分法求方程采用的是逐步逼近法。如图5.3所示，将方程转为$y=2x^3-4x^2+3x-6$，绘制于坐标系上是一条连续的曲线，则问题变为了当$y=0$时，x的值是多少，显然方程曲线与x轴相交处即方程的解。

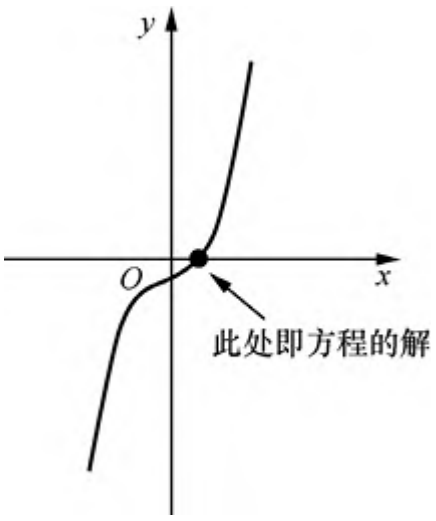


图5.3

题目中x的取值范围为(-10,10)，则取首尾两数$x1=10$、$x2=-10$，分别将$x1$和$x2$代入方程求解得fx1，fx2。

再取$x0=(x1+x2)/2$，将$x0$代入方程得fx0。此时分两种情况讨论，如图5.4所示。

如此反复循环直至fx0小于1e-5（科学记数法1×10^{-5}，即0.00001）为止。

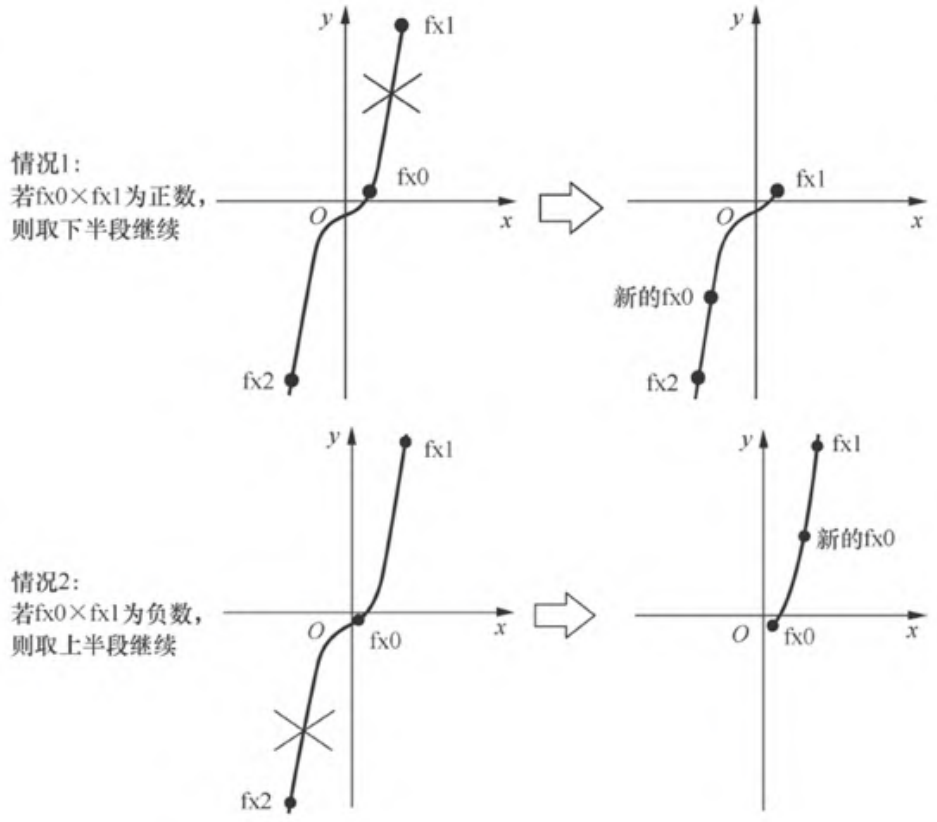


图5.4

参考程序如下所示

```
1 //二分法求方程的解
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
6 {
7   float x0,x1,x2,fx0,fx1,fx2;
```

```
 8    do
 9    {
10      scanf("%f%f",&x1,&x2);
11      fx1=x1*((2*x1-4)*x1+3)-6;
12      fx2=x2*((2*x2-4)*x2+3)-6;
13    } while(fx1*fx2>0);                                    //必须输入正确值
14    do
15    {
16      x0=(x1+x2)/2;
17      fx0=x0*((2*x0-4)*x0+3)-6;
18      if ((fx0*fx1)<0)        //取互为相反数那一段，正数与负数间必有0
19      {
20        x2=x0;
21        fx2=fx0;
22      }
23      else
24      {
25        x1=x0;
26        fx1=fx0;
27      }
28    } while(fabs(fx0)>=1e-5);
29    printf("%0.2f\n",x0);
30    return 0;
31  }
```

5.3.7　求最大子序列和（sum）

【题目描述】

给一串整数a[1],…,a[n]，求出它的最大子序列和，即找出$1 \leqslant i \leqslant j \leqslant n$，使得a[$i$]＋a[$i$＋1]＋…＋a[$j$]最大。

【输入格式】

输入第一行为一个整数n，表示有n（$n \leqslant 35\,000$）个整数，第二行为n个整数。

【输出格式】

输出共一行，即最大子序列和。

【输入样例】

5

1 2 5 -10 7

【输出样例】

8



这道题可以用最朴素的算法，其时间复杂度为$O(n^3)$。

试考虑能否优化。

参考程序如下所示。

```
1 //求最大子序列和—方法1
2 #include <bits/stdc++.h>
3 using namespace std;
```

```
 4 const int MAX=35005;
 5
 6 int main()
 7 {
 8   int n,sum,a[MAX]= {0};
 9   scanf("%d",&n);
10   for (int i=1; i<=n; i++)
11     scanf("%d",&a[i]);
12   int ans=a[1];
13   for (int i=1; i<=n; i++)
14     for (int j=i; j<=n; j++)
15     {
16       sum=0;
17       for (int k=i; k<=j; k++)
18         sum+=a[k];
19       ans=max(sum,ans);      //max(sum,ans)返回两者中的最大值给ans
20     }
21   printf("%d\n",ans);
22   return 0;
23 }
```



使用前序和思想优化：例如有A、B、C、D、E共5个人，每人手上有一定数量的糖果，其中A只知道他自己的糖果数a，B知道A和B两人的糖果总数b，C知道A、B、C这3人的糖果总数c，D知道A、B、C、D 4人的糖果总数d，E知道全部5人的糖果总数e，那么C、D、E这3人的糖果总数就是$e-b$。

所以此题可以先求出前i个数的和存入前序和数组s[i]。

例如有数组a的各元素值为1、2、5、-10、7，则对应数组s的各元素值为1、3、8、-2、5。

可得a[i]+a[i+1]+…+a[j]＝s[j]-s[i-1]，因此可以去掉k，时间复杂度为$O(n+n^2)=O(n^2)$。例如求a[1]到a[3]这一段的子序列和，有s[3]-s[0]＝-2-1＝-3。

参考程序如下所示。

```
 1 //求最大子序列和—方法2
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4 const int MAX=35005;
 5
 6 int main()
 7 {
 8   int n,s[MAX]= {0},a[MAX]= {0};
 9   scanf("%d",&n);
10   for (int i=1; i<=n; i++)                    //注意从下标1开始
11   {
12     scanf("%d",&a[i]);
13     s[i]=s[i-1]+a[i];
14   }
15   int ans=a[1];
16   for (int i=1; i<=n; i++)
17     for (int j=i; j<=n; j++)
18       ans=max(ans,s[j]-s[i-1]);                //更新最大值给ans
19   printf("%d\n",ans);
20   return 0;
21 }
```



上面的优化程序是先求出前i个数的和s[i]，则：

a[i]+a[i+1]+…+a[j]＝s[j]-s[i-1]

那么对于给定的s[j]来说，能否直接找到最小的s[i-1]值呢？

参考程序如下所示。

```
 1 //求最大子序列和—方法3
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int n,s[35005]= {0},a[35005]= {0},MAX=-2147483647;
 8   scanf("%d",&n);
 9   for (int i=1; i<=n; i++)
10   {
11     cin>>a[i];
12     s[i]=s[i-1]+a[i];
13     MAX=max(MAX,a[i]);          //找到最大值的元素
14   }
15   int Min=a[1],ans=a[1];        //Min用来保存到目前为止的元素的最小值
16   for (int i=1; i<=n; i++)
17   {
18     Min=min(Min,s[i]);          //min(a,b)可得到a、b的最小值
19     ans=max(ans,s[i]-Min);      //更新最大值给ans
20   }
21   printf("%d\n",ans?ans:MAX);  //元素如果全是负数，则输出最大值的元素
22   return 0;
23 }
```



继续改进：首先设最大值*sum*为一个极小值，输入数组数据后，从最后一个数开始向前依次相加，并将和存入变量*b*（*b*的初始值为0）。如果*b*＞*sum*，则更新*sum*的值为*b*；如果*b*＜0，则把*b*清零。

重复上述操作，最后即可得出最大值*sum*。

参考程序如下所示。

```
 1 //求最大子序列和—方法4
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int n,sum=-2147483647,b=0,a[35005]= {0};
 8   scanf("%d",&n);
 9   for (int i=1; i<=n; ++i)
10     scanf("%d",&a[i]);
11   for (int i=n; i>=1; --i)
12   {
13     b+=a[i];
14     sum=max(b,sum);
15     if (b<0)
16       b=0;
17   }
18   printf("%d\n",sum);
19   return 0;
20 }
```

5.3.8　猴子选大王（monkey）

【题目描述】

如图5.5所示，从1到*M*进行编号的猴子围成一圈选大王。规则是从第一只猴子开始从1循环报数，数到*N*的猴子出圈，然后下一只猴子再重新从1报数……剩下的最后一只猴子就是大王。试编程计算成为大王的猴子的编号。

【输入格式】

输入两个数字，即M（$M \leqslant 100$，$N \leqslant 100$）。

【输出格式】

输出一个数字，即成为大王的猴子的编号。

【输入样例】

3 2

【输出样例】

3

【算法分析】

这道题由于数据规模不大，因此可以使用一般的模拟法，即用数组模拟猴子选大王的过程。

参考伪代码如下所示。

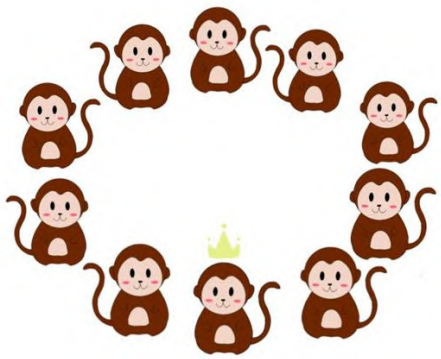


图5.5

使用“内容市场”

APP扫描看视频

```
 1 定义a[m]，所有数组元素初始值为1，代表每一只猴子均在圈内
 2 定义变量k，k相当于一个指向猴子的指针，用于模拟循环报数
 3 for(m次循环，每次循环出圈一只猴子)
 4 {
 5   定义变量Count用于计数，每次循环前均初始化为0，表示还未开始计数
 6   while(Count还未数到n时)
 7   {
 8    k指针依次后移模拟报数过程，如果移到最后一个，则跳回到第一个
 9    if(k指针指向位置的猴子没有出圈即a[k]==1)
10      计数器Count加1
11 }
12    Count数到n了，则此时k指针指向位置的猴子出圈，即a[k]=0
13 }
14 直接输出最后一只出圈猴子所在的位置
```

当猴子出圈后，程序依然会检查该猴子是否出圈，这浪费了大量的时间。可以设数组元素的值存放下一个要报数的猴子的编号，以8只猴子为例，a[1]的值为2，即下一只猴子是2号猴子；a[8]的值为1，即下一只猴子是1号猴子，如表5.1所示。

表5.1

数组元素	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
元素值	2	3	4	5	6	7	8	1

这样构成了一个环，如果5号猴子出圈，就用语句a[4]＝a[a[4]]，以后报数到4号猴子时，就直接指向6号猴子。

请试着优化该程序。

5.3.9 NOI（NOI）

【题目描述】

统计有多少次NOI出现在一个仅由3个大写字母“N” “O” “I”组成的字符串中的方法是：只要NOI 3个字母的顺序正确，即使其内插入了其他字符也可以，甚至共享字符也是可以的。例如NIOI出现了1次NOI，NNOI出现了2次NOI，NNOOII出现了8次NOI。

【输入格式】

第一行为一个整数*N*。第二行为*N*个字符的一个字符串，每个字符是 “N”、“O”或“I”。

【输出格式】

输出一个整数（其取值范围为不超过64的整数），即NOI出现的次数。

【输入样例】

6

NNOOII

【输出样例】

8

【算法分析】

设整型变量N、NO和NOI分别代表“N”“NO” “NOI”出现的次数，从左到右扫描字符串，

每扫描到 “N”，变量N++；每扫描到 “O”，变量NO+=N；每扫描到“I”，变量NOI+=NO，最后输出NOI的值即可。

参考程序如下所示。

```
1 //NOI
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 long long len,N,NO,NOI;          //题目的取值范围为不超过64位的整数
6
7 int main()
```

```
 8 {
 9   cin>>len;
10   getchar();                          //消去回车符
11   string st;
12   getline(cin,st);
13   for (int i=0; i<len; i++)
14   {
15     if (st[i]=='N')
16       N++;
17     if (st[i]=='O')
18       NO+=N;
19     if (st[i]=='I')
20       NOI+=NO;
21   }
22   cout<<NOI<<endl;
23   return 0;
24 }
```

5.3.10 NOI字符串（NOI）

【题目描述】

统计有多少次NOI出现在一个仅由3个大写字母“N”“O”“I”组成的字符串中的方法是：只要NOI这3个字母的顺序正确，即使其内插入了其他字符也可以，甚至共享字符也是可以的。例如NIOI出现了1次NOI，NNOI出现了2次NOI，NNOOII出现了8次NOI。

现在要将大写字母“N”“O”“I”中的任意一个插入某字符串中的任意位置，问最多会出现多少次NOI？

【输入格式】

第一行为一个整数N。第二行为N个字符的一个字符串，每个字符是 “N”、“O”或“I”。

【输出格式】

输出一个整数（其取值范围为不超过64位的整数），表示最多会出现多少次NOI。

【输入样例】

5

NOIOI

【输出样例】

6

【样例说明】

将大写字母“N”加到最前面即“NNOIOI”，会出现最多次NOI。

【算法分析】

如果插入“N”，显然“N”一定是插入字符串的最前端。

如果插入“I”，显然“I”一定是插入字符串的最末端。

如果插入“O”，显然需要尝试将“O”插入字符串的每一个位置i，使i前面的“N”的个数×i后面的“I”的个数乘积最大。尝试插入过程中，无须反复统计i前面的“N”的个数和i后面的“I”的个数，直接使用前序和思想统计出i前面的“N”的个数存入N[i]和i前面的“I”的个数存入I[i]，这样当“O”位于字符串的位置i时，组成“NOI”的个数即N[i-1]×(I[n]-I[i-1])。

5.3.11　组合数问题（problem）

【题目描述】

从集合(1,2,3)这3个人中选择两个人参加比赛，可以有(1,2)、(1,3)、(2,3)这3种方案。我们称从n个人中选择m个人参加比赛的方案数为组合数$C(n,m)$。组合数的一般公式为$C(n,m)=\frac{n!}{m!(n-m)!}$，规定$C(n,0)=1$。

现给定n、m及k，求对于所有的i和j（$0\leqslant i\leqslant n$，$0\leqslant j\leqslant i$），有多少个$C(i,,j)$是k的倍数。

【输入格式】

第一行为两个整数T（$1\leqslant T\leqslant 10000$）和$k$（$1<k\leqslant 21$），其中$T$为测试组数。

接下来的T行中，每行有两个整数即n（$3 \leqslant n \leqslant 2\,000$）和$m$（$3 \leqslant m \leqslant 2\,000$）。

【输出格式】

输出T行，每行为一个整数，即有多少个$C(i,j)$是k的倍数。

【输入样例】

1 2

3 3

【输出样例】

1

【样例说明】

只有$C(2,1)=2$是2的倍数。

【算法分析】

尝试计算$C(0,0)$，$C(1,0)$，$C(1,1)$，$C(2,0)$，$C(2,1)$…，可以发现得出的数列其实是杨辉三角数字序列，如图5.6所示。

n \ m	0	1	2	3	4	5	…
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
…			…				

图5.6

由此可推出：$C(n, m) = C(n-1, m-1) + C(n-1, m)$



还可以这样考虑：设从n个物品中随便拿出一个物品t放在一边，则从n个物品中取m个物品的方案数可以看成两部分方案数的和。

一部分为包含这一个物品t的方案数，即从$n-1$个物品中取$m-1$个物品的方案数（此时加上这一个物品t即m个）。

一部分为不包含这一个物品t的方案数，即从$n-1$个物品中取m个物品的方案数（不考虑放入物品t）。

故$C(n, m) = C(n-1, m-1) + C(n-1, m)$。

预处理所有组合数对k取模的值，例如当k=2时，预处理结果如图5.7所示。

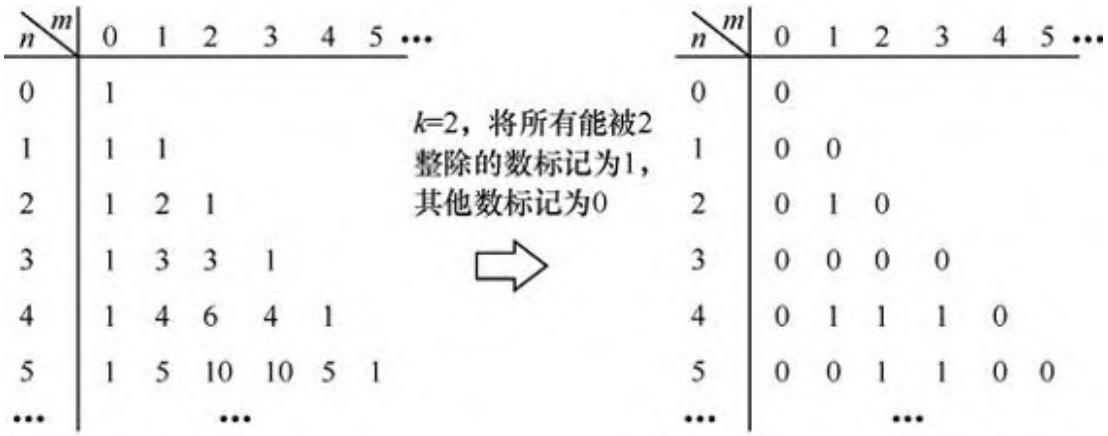


图5.7

考虑到题目中测试组数T最多有10 000组，所以如果一遍一遍地反复累加肯定会超时。容易想到的改进方法是借用前序和的思想，从上到下、从左到右逐个统计前序和存入二维数组f[][]，如图5.8所示。

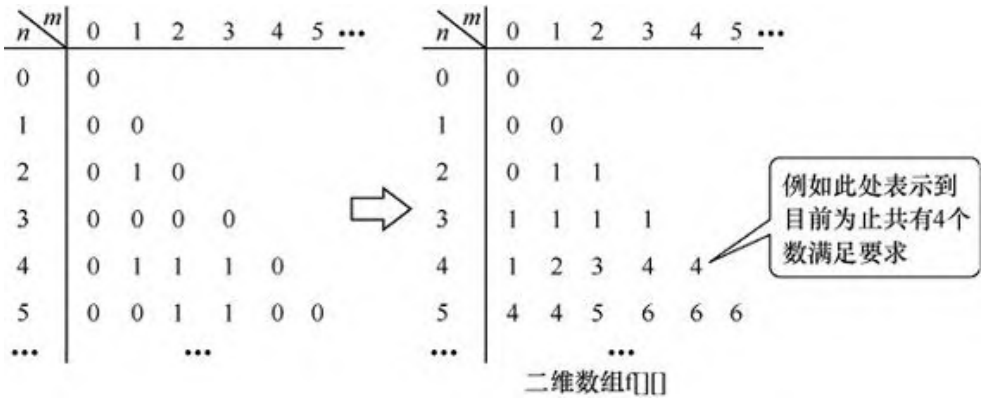


图5.8

那么f[n][m]的值就是要求的答案吗？显然不是的。例如当n=5、m=2、k=2时，如图5.9所示，f[5][2]统计的是整个黑色框里满足条

件的数的个数，而答案应该是黑色框里灰色区域内满足条件的数的个数。

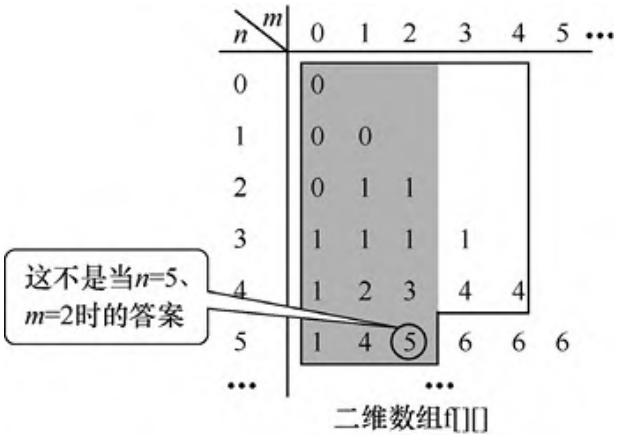


图5.9

所以我们需要改进二维数组f[i][j]，使之能直接表示有多少个$C(i,j)$是k的倍数，如图5.10所示。

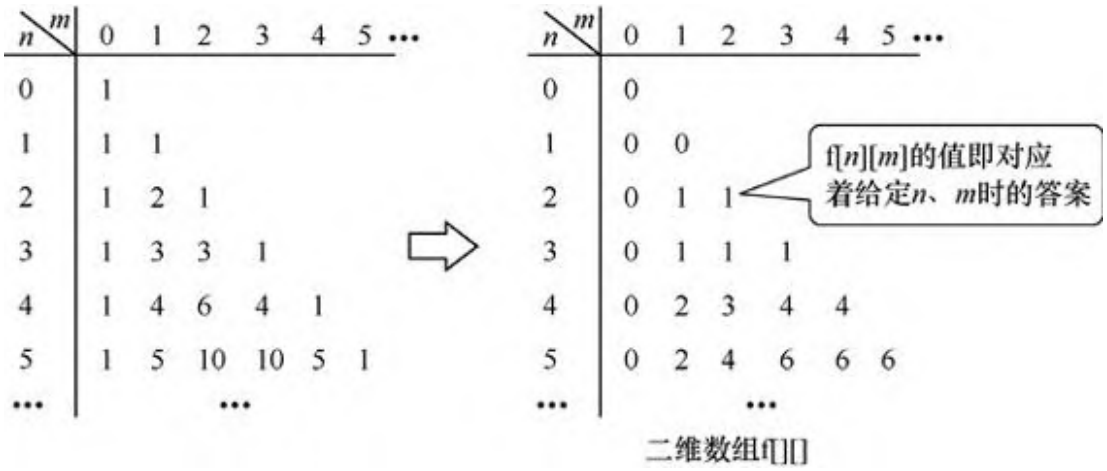


图5.10

二维数组f[][]的生成利用了矩阵前序和的优化技巧（容斥原理），即f[i][j]＝f[i-1][j]＋f[i][j-1]-f[i-1][j-1]＋(!(c[i]

[j]%k))，其原理如图5.11所示。

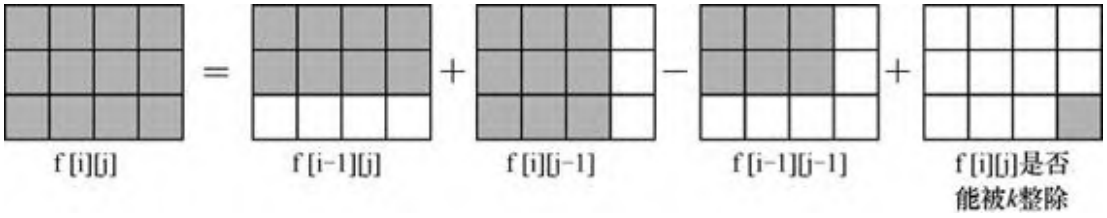


图5.11

第06章　函数

为了降低编程难度，复杂的程序不应该将所有的代码都写在main()函数里，通常的做法是将任务分成若干小块，由其他的程序模块分担完成，在C++语言里，这些程序模块叫作函数。

6.1　初识函数



扫码看视频

C++程序可由一个主函数(main()函数)和若干个子函数构成。主函数可以调用其他子函数，其他子函数也可以互相调用，但主函数不可被子函数调用。

例如下面这个求两个整数中的最大数的程序中，求两个整数中的最大数的功能是由Max()子函数完成的而不是由main()函数完成的，main()函数只是直接调用了Max()子函数的计算结果，“Max”是用户自定义的名称。

```
 1 //求两个整数中的最大数
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int Max(int a,int b)          //Max()为子函数
 6 {
 7   return (a>b?a:b);           //返回值为a、b中的最大值
 8 }
 9
10 int main()
11 {
12   int x,y;
13   scanf("%d %d",&x,&y);
14   printf("%d\n",Max(x,y));    //调用Max()子函数，并将x、y的值传递给
Max()子函数
15   return 0;
16 }
```

上例中的函数说明如图6.1所示。

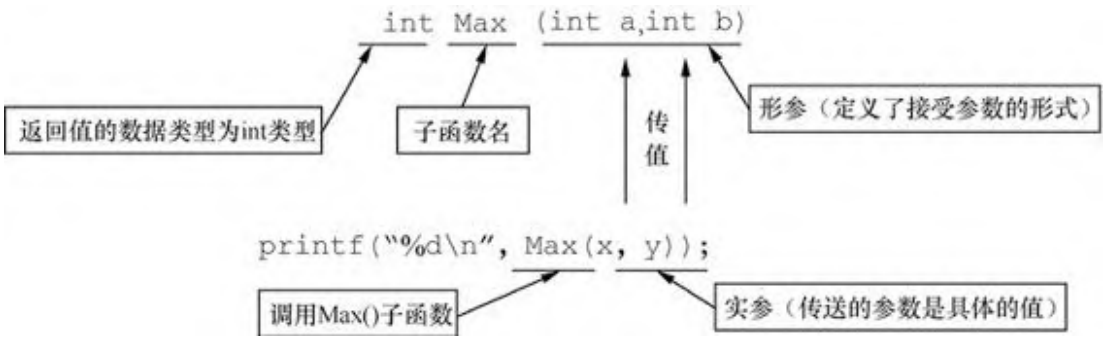


图6.1

可以这样理解：作为“领导”的main()函数不想亲自比较两个数哪个大，于是把任务派给“下属”Max()子函数去完成，显然main()函数要先把两个具体的数即实参x和y告知给“下属”，“下属”使用形

参a和b接收到这两个数后进行比较，再报告给main()函数一个结果，这个结果就是返回值。对于main()函数来说，“下属”如何完成任务它不必管，它只需要告诉“下属”两个数就肯定可以得到结果。

“两人”做事十分严谨，“领导”传送的实参个数和数据类型都必须和“下属”接收时用的形参个数和数据类型一一对应。如果数据类型不匹配或者参数个数不匹配，就可能发生错误或者无法编译。

“下属”返回的结果的数据类型也是事先指定好的，例如上例中Max()子函数返回的结果的数据类型是int类型，所以返回值的数据类型也应该是int类型，如果返回值的数据类型不是int类型，则系统强制将返回值的数据类型转换为int类型。

再看一个求两个浮点数的和的程序。

```
 1 //求两个浮点数的和
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 float Add(float x,float y)//返回数据类型为float类型，有两个形参x和 y
 6 {
 7   float z;                  //注意，定义了z
 8   z=x+y;
 9   return x+y;               //返回值z的数据类型为float类型，与返回值数
据类型匹配
10 }
11
12 int main()
13 {
14   float x,y,z;              //注意，定义了x、y、z
15   scanf("%f %f",&x,&y);
16   z=Add(x,y);    //调用Add()函数后返回一个float类型的值，并将该值赋给z
17   printf("%0.2f\n",z);
```

```
18    return 0;
19 }
```



在这个程序里两次定义了z，这是被允许的。因为一个z定义在main()函数里，另一个z定义在Add()函数里，两个z各自从属于自己的函数，它们是没有冲突的。

第16行将main()函数里的x、y的值传给了Add()函数里的x、y，Add()函数里的x、y的值无论如何改变，也不会影响到main()函数里的x、y的值。

如果子函数有返回值，则只能返回一个数值。

6.1.1　浮点数求最大值（Max）

【题目描述】

输入3个浮点数，求其中的最大值，请使用函数编程解决。

【输入格式】

输入3个浮点数。

【输出格式】

输出其中的最大值。

【输入样例】

1.1 2.2 3.3

【输出样例】

3.3

6.1.2　判断平方数（square）

【题目描述】

写一个判断平方数的程序，判断输入的整数x是否为平方数并输出结果。

【输入格式】

有多组数据，每组一行，为一个整数x。

【输出格式】

每组输出一行，如果x是平方数则输出1，否则输出0。

【输入样例】

25

99

【输出样例】

1

0

参考程序如下所示。

```
 1 //判断平方数
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 void Square(int n);              //对子函数的声明
 6
 7 int main()
 8 {
 9   int x;
10   while (cin>>x)
11     Square(x);
12   return 0;
13 }
14
15 void Square(int n)              //void表示函数无返回值
16 {
17   for (int i=1; n>0; i+=2)      //由1,1+3,1+3+5,…为平方数推导而来
```

```
18      n-=i;
19    cout<<(n==0?1:0)<<endl;        //直接输出结果，所以无返回值
20  }
```



由于被调用的函数Square()在调用的函数main()之下，因此在第5行要对Square()函数事先进行声明才可以使用。

由于输出结果直接在Square()函数完成，无须返回结果值，因此在Square()函数名前需加void，表示无返回值。

6.1.3　哥德巴赫猜想（Goldbach）

【题目描述】

输入整数a和b，试验证a～b的所有正偶数都能够分解为两个质数之和（验证哥德巴赫猜想对a～b的正偶数成立）。

【输入格式】

输入两个整数a和b（$2 < a < b \leqslant 500\,000$，$b - a < 200\,000$）。

【输出格式】

输出a～b的正偶数的质数之和，每个占一行，例如4＝2＋2。如果有多种可能则只输出一种，即第一个质数最小的。

【输入样例】

3 6

【输出样例】

4=2+2

6=3+3

一般的参考程序如下所示。该程序需要反复验证某数是否为质数，效率较低。

```
 1 //哥德巴赫猜想 —— 一般写法
 2 #include<bits/stdc++.h>
 3 using namespace std;
 4
 5 int Prime(int i)                          //判断是否为质数
 6 {
 7   if (i==2)
 8     return 1;
 9   if (!(i%2))                             //如果是偶数,返回 0
10     return 0;
11   int k=sqrt(i);
12   for (int j=3; j<=k; j+=2)
13     if (!(i%j))
14       return 0;
15   return 1;                                //如果是质数,返回1
16 }
```

```
17
18 int main()
19 {
20   int a,b;
21   cin>>a>>b;
22   if (a%2)
23     a++;
24   for (int i=a; i<=b; i+=2)
25     for (int n=2; n<i; n++)              //将偶数i分解为两个整数
26       if (Prime(n) && Prime(i-n))       //判断分解成的两个整数是否均为
质数
27       {
28         printf("%d=%d+%d\n",i,n,i-n);//若均是质数则输出
29         break;
30       }
31   return 0;
32 }
```



进一步的优化是用筛选法把100 000以内的所有质数存到一个数组中，然后判断某个正偶数是否满足哥德巴赫猜想时枚举该正偶数的所有分解方案，若分解成的两个整数均在质数数组中，则输出该分解方案。

对于输入大量整数的操作，可以采用基于秦九韶算法思想的快速输入方法。

快速输入方法使用getchar()函数，用数字累加的方法输入整数，参考程序如下所示。

```
 1 //快速输入整数
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 inline int Read()        //快速读入整数,inline表示该函数为内联函数
 6 {
 7   int x=0,f=1;
 8   char ch=getchar();                        //使用getchar()函数比使用
scanf()函数约快10倍
 9   for (; ch<'0' || ch>'9'; ch=getchar())   //对非数字的处理
10     if (ch=='-')                            //对负号的处理
11       f=-1;
12   for (; ch>='0' && ch<='9'; ch=getchar())//对数字的处理
13     x=x*10+ch-'0';
14   return x*f;
15 }
16
17 int main()
18 {
19   while (1)
20   {
21     int x=Read();
22     printf("%d ",x);
23   }
24   return 0;
25 }
```



函数是通过栈操作的方式调用的，对于一些频繁调用、简单的小函数来说，需要较大的资源开销，并会导致程序运行效率降低。如果在这些小函数前加上关键字inline，编译器就会在编译阶段直接将函数体嵌入调用该函数的语句块中以提高程序运行效率，这种函数称为内联函数。

但inline函数最终能否真正内联，实际由编译器决定，它如果认为函数不复杂，能在调用点展开就会真正内联，并不是说声明了内联就会内联。

实际上，现代的多数编译器会自动将短小的可以内联的函数内联，并不是只有前面带有inline标识的函数才会将其视作内联函数。

6.1.4 约分（comm）

【题目描述】

3/6的约分形式应该是1/2，试编程输入n个分数，输出其对应的约分形式。

【输入格式】

第一行为一个整数n（$n \leqslant 500\,000$），表示有n个分数。随后为n行，每行为一个分子和一个分母，分子和分母之间以“/”间隔。

【输出格式】

输出n行，分别对应其约分形式的分数，分子和分母之间以“/”间隔。

【输入样例】

1

3/9

【输出样例】

1/3

6.1.5　素数回文数（Prime4）

【题目描述】

左右对称的自然数称为回文数，例如121、4 224、13 731等。试编程找出a～b的素数回文数，例如151既是素数又是回文数。（题目来源：HDU 1431简化版）

【输入格式】

输入两个整数a和b（$2 \leqslant a < b \leqslant 10\,000\,000$）。

【输出格式】

对每一组数据按从小到大顺序输出a～b的所有满足条件的素数回文数（包括a和b），每个数各占一行。

【输入样例】

6 500

【输出样例】

7

11

101

……

6.1.6 丑数（ugly）

【题目描述】

我们把只包含因子2、3、5的数称为丑数，例如6、8都是丑数，但14不是，因为它包含因子7。习惯上把1当作第一个丑数。试编程输出第n个丑数。

【输入格式】

输入一个正整数n（$1 \leqslant n \leqslant 10000$）。

【输出格式】

输出第n个丑数。

【输入样例】

11

【输出样例】

15

【算法分析】

逐个枚举丑数的效率很低，我们可以创建一个数组保存排好序的丑数，而每一个新的丑数都是前面的丑数乘以2、3或者5得到的。

6.1.7　三质数（prime5）

【题目描述】

一个数的约数也称为因子，例如1、2、4都是4的因子。质数的因子是1和它本身。

三质数只有3个不同的因子。例如4是三质数，因为它有1、2、4共3个因子；6不是三质数，因为6有1、2、3、6共4个因子。现在有一些数，试判断它们是不是三质数。

【输入格式】

多组测试数据（不超过1 000），每组测试数据输入一个整数n（$1 \leqslant n \leqslant 10^{12}$）。

【输出格式】

对于每组测试数据，判断是不是三质数，如果是则输出“YES”，否则输出“NO”。

【输入样例】

```
4
5
```

【输出样例】

```
YES
NO
```

【笔试测验】请手动计算程序的运行结果。

```
#include <bits/stdc++.h>
using namespace std;

int Func(int a,int b)
{
  return (a+b);
}

int main()
{
  int x=2,y=5,z=8;
  cout<<Func(Func(x,y),z);
  return 0;
}
```

C++语言的函数定义都是互相平行、独立的，也就是说在定义函数时，一个函数内不能包含另一个函数。但函数可以嵌套调用函数，也就是说，在调用一个函数的过程中又调用另一个函数，如图6.2所示。

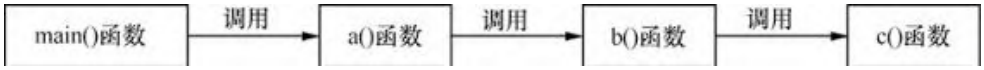


图6.2

6.2 库函数简介

前面已经接触了一些C++语言的库函数，现在通过具体实例来了解库函数的使用。

查看附录中的常用函数库，从中挑选几个有代表性的函数如表6.1所示。

表6.1

函数名	函数原型	函数功能	返回值
fabs	double fabs(double x)	返回双精度浮点数x的绝对值	计算结果
islower	int islower(int ch)	若ch是小写字母（a～z）则返回非0值，否则返回0	返回1或0
rand	int rand(void)	产生一个随机数并返回这个数	随机整数
strlen	unsigned int strlen(char *str)	返回字符串str的长度	返回字符串的长度



在使用这些函数时，必须在源文件头引入所在库的头文件。例如使用数学库函数时，需要写上 #include <cmath>；使用字符串函数时，需要写上#include <cstring>等，否则会出现编译错误。

如果引入了“万能”头文件即bits/stdc++.h，就无须考虑这些问题了。

fabs()函数的功能是求双精度浮点数x的绝对值，它与abs()函数的区别是abs()函数求的是整数x的绝对值。所以使用fabs()函数时，输入的x应该为双精度浮点数。例如：

```
double x=-34.0,result;
result=fabs(x);              //result等于34.0
```

islower()函数的功能是检查字符ch是否为小写字母，如果是则返回值为1，如果不是则返回值为0。例如：

```
char ch1='A',ch2='a';
cout<<islower(ch1);        //输出的值为0
cout<<islower(ch2);        //输出的值为1
```

rand()函数的功能是产生随机数，可以看到，该函数形参为void类型，即无输入参数。例如：

```
int r=rand();                //产生一个随机整数（0～32767）赋给int类型数r
```

strlen()函数的功能是返回字符串的长度，其返回值是正整数，因为字符串的长度为正整数。参数形式为char *str，所谓“*”，表明该变量为地址值，即字符型变量str是一个地址值。我们知道，C++语言中的数组名其实代表该数组的首地址，例如：

```
char ch[]="hello";
int num=strlen(ch);        //数组名ch为该字符型数组的首地址，num的值为5
```

6.3 常用的变量类型

局部变量是指在函数内部定义的变量，它只能在本函数范围内使用，其他函数是不能使用的。局部变量如果在定义时没有赋初始值，则其值是之前该内存空间未删除的无用数据，即未知的任意一个值。

全局变量是指在函数外部定义的变量，它能被它后面的所有函数使用。全局变量如果在定义时没有赋初始值，则其值会自动赋值为0（char类型的自动赋值也是0，即NULL，0在ASCII中是控制字符，不可见）。

请看一个具体的实例。

```
 1 //全局变量与局部变量
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int num1=5;    //全局变量，赋值为5，下面的所有函数都可使用它
 6 char c[10];    //全局数组，全部元素均自动赋值为0，下面所有函数都可使用它
 7
 8 void Fun1()
 9 {
10   int a=1;     //a为局部变量，只能在本函数中使用
11   num1+=a;     //可以使用全局变量num1
12 }
13
14 int num2=4;    //全局变量，不能被Fun1()函数使用，可被下面的Fun2()函数和
main()函数使用
15 void Fun2()
16 {
17   num2++;
18 }
19
20 int a[10];     //全局数组，全部元素均自动赋值为0，只能被后面的main()函数
使用
21 int x;         //全局变量，系统自动赋值为0，只能被后面的main()函数使用
22 int main()
23 {
24   int y;                    //局部变量，值未知，只在main()函数中有效
25   Fun1();
26   Fun2();
```

```
27    printf("%d\n",num1+num2);//可使用全局变量num1和num2，输出值为11
28    for (int i=0; i<=9; i++)
29      printf("%d ",a[i]);     //全局数组未手动赋值，输出的值均为0
30    printf("\n x=%d,y=%d\n");//全局变量x为0，局部变量y为未知的值
31    for (int i=0; i<=9; i++)
32      printf("%c ",c[i]);     //按%c格式输出为空，若按%d格式输出均为0
33    printf("\n");
34    int b[10];                //局部数组未手动赋值，输出值为任意可能值
35    for (int i=0; i<=9; i++)
36      printf("%d ",b[i]);     //输出的值未知
37    return 0;
38  }
```



所有函数都能使用它上面的全局变量，因此如果在一个函数中改变了全局变量的值，就有可能影响到其他函数对此变量的使用，这相当于各个函数间有了直接的传递通道。请谨慎使用全局变量，以免“牵一发而动全身”。

有时希望函数中的局部变量的值在函数调用结束后不消失而保留原值，即其占用的存储单元不释放，在下一次该函数调用时，该变量已有值，就是上一次函数调用结束时的值，这时就应该指定该局部变量为“局部静态变量”，用static加以说明。C++语言规定，局部静态变量数组和局部静态变量会自动赋初始值。

下面的程序运行结果为2 3 4，而不是2 2 2，这说明局部静态变量n在函数调用结束后并不释放，仍保留其值。

```
 1 //局部静态变量演示
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int Fun()
 6 {
 7   static int n=1;
 8   n++;
 9   return n;
10 }
11
12 int main()
13 {
14   for (int i=0; i<3; i++)
15     printf("%d ",Fun());                    //输出2 3 4,而不是2 2
2
16   return 0;
17 }
```

【笔试测验】请手动算出程序的结果。

```
 1 #include <bits/stdc++.h>
 2 using namespace std;
 3
 4 void Sub(int s[])
 5 {
 6   static int t=0;
 7   s[t]++;
 8   t++;
 9 }
10
11 int main()
12 {
13   int a[]= {1,2,3,4},i;
14   for (i=0; i<4; i++)
15     Sub(a);
16   for (i=0; i<4; i++)
17     cout<<a[i]<<" ";
18   return 0;
19 }
```

6.4 数组作为函数参数

C++语言可以使用数组元素作为函数实参，其用法与普通变量的用法相同。例程如下所示。

```
 1 //数组元素作为函数实参
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 void Change(int a,int b)          //形参不能写成类似于int a[i]的形式
 6 {
 7   swap(a,b);                      //a与b的值互换
 8 }
 9
10 int main()
11 {
12   int a[]= {1,2,3,4,5,6,7,8};
13   int b[]= {-1,-2,-3,-4,-5,-6,-7,-8};
14   for (int i=0; i<8; i++)
15     Change(a[i],b[i]);
16   for (int i=0; i<8; i++)
17     printf("%d %d\n",a[i],b[i]);//此时a[i]和b[i]的值还是初始值吗?
18   return 0;
19 }
```



main()函数里的a[]和b[]的元素值并未改变，这说明参数的值是单向传递的。系统只是将a[i]和b[i]值的传给了Change()函数里的a和b，Change()函数里交换的只是a[i]和b[i]值的副本，与原a[i]、b[i]无关。

数组名也可以作为函数的参数，它传递的是数组首地址。例程如下所示。

```
 1 //数组名作为参数
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 void Change(int x[],int y[])        //注意此处形参的写法
 6 {
 7   for (int i=0; i<8; i++)
 8     swap(x[i],y[i]);
 9 }
10
11 int main()
12 {
13   int a[]= {1,2,3,4,5,6,7,8};
14   int b[]= {-1,-2,-3,-4,-5,-6,-7,-8};
15   Change(a,b);                       //a、b即数组名，也是数组首地址
16   for (int i=0; i<8; i++)
17     printf("%d %d\n",a[i],b[i]);    //此时a[i]和b[i]的值还是初始值
吗?
18   return 0;
19 }
```



运行程序后可以发现，数组a与数组b的值互换了。这是因为用数组名作为参数所传递的不是它的副本，而是它的内存地址，所以Change()函数形参中的数组x、数组y的首地址与main()函数中的数组a、数组b的首地址是指向同一个位置的，因此对x[i]和y[i]的操作也是对a[i]和b[i]的操作。

再看一个二维数组名作为参数的例程。main()函数中的数组a是局部数组，只能被main()函数引用，不能被Max()函数直接引用，为了Max()函数能引用main()函数中的数组a的值，要将数组a的首地址（也就是数组名）作为参数传给Max()函数。

因为传递的是数组a的首地址，Max()函数里的数组a和main()函数里的数组a的存储位置是重叠在一起的，所以对Max()函数里的数组a进行操作，就相当于对main()函数里的数组a进行操作。

```
 1 //二维数组名作为参数求二维数组的最大元素值
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int Max(int a[][4])                              //形参数组的列数不可为空
 6 {
 7   int Max=a[0][0];
 8   for (int i=0; i<3; i++)
 9     for (int j=0; j<4; j++)
10       if (a[i][j]>Max)
11         Max=a[i][j];
12   return Max;
13 }
14
15 int main()
16 {
17   int a[][4]= {1,2,3,4,5,6,7,8,9,-1,-2,-3};//可省略第一维数值大小
18   printf("最大值为 %d\n",Max(a));            //数组名作为参数
19   return 0;
20 }
```

6.5 函数的递归调用



扫码看视频

所谓递归，是指函数能够直接或间接地调用自身。例如下面的程序段。

```
1 int Fun(x)
2 {
3    return Fun(2*x);                                    //调用自身
4 }
```

可以看到，上面的程序段是无终止的自身调用，这会导致程序运行时崩溃。正确的做法应该是在函数调用自身达到一定次数后终止调用，这可以用if语句来控制。

用一个通俗的例子来说明。

有5个人坐在一起，问第5个人多少岁，他说比第4个人大2岁；问第4个人多少岁，他说比第3个人大2岁；问第3个人多少岁，他说比第2个人大2岁；问第2个人多少岁，他说比第1个人大2岁，最后问第1个人多少岁，他说是10岁。请问第5个人多少岁？

可以用表达式表述如下：

$$Age(n) = \begin{cases} 10 & (n = 1) \\ Age(n-1) + 2 & (n>1) \end{cases}$$

求解过程可分成两个阶段。第一阶段是“递推”，即将第n个人的年龄表示为第（n-1）个人年龄的函数，而第（n-1）个人的年龄不知道，还要“递推”求出第(n-2)个人的年龄……直到“递推”获得第1个人的年龄，此时$Age(1)$已知，不必再向前推。于是开始第二阶段即“回归”，从第1个人的年龄“回归”算出第2个人的年龄，从第2个人的年龄“回归”算出第3个人的年龄……一直“回归”算出第5个人的年龄。其运算过程如图6.3所示。

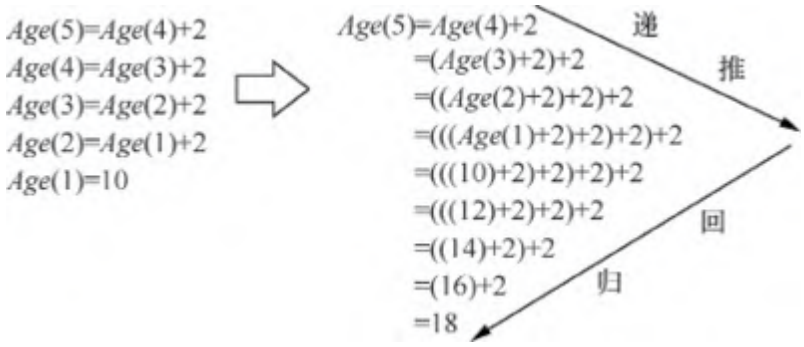


图6.3

参考程序如下所示。

```
1 //递归求年龄
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int Age(int n)
6 {
7   if (n==1)                  //if语句控制递归结束
8     return 10;
9   else
10    return Age(n-1)+2;      //调用自身，参数值每递归一次就减少1个数
11 }
12
13 int main()
14 {
15   printf("%d\n",Age(5));
```

```
16    return 0;
17 }
```

如图6.4所示，程序运行的过程是这样的：当要调用函数自身时，系统将该函数在内存中复制一份，记住当前函数的位置后在复制的程序中运行，如果再调用再复制，直到递归结束后返回到原来位置后继续运行……

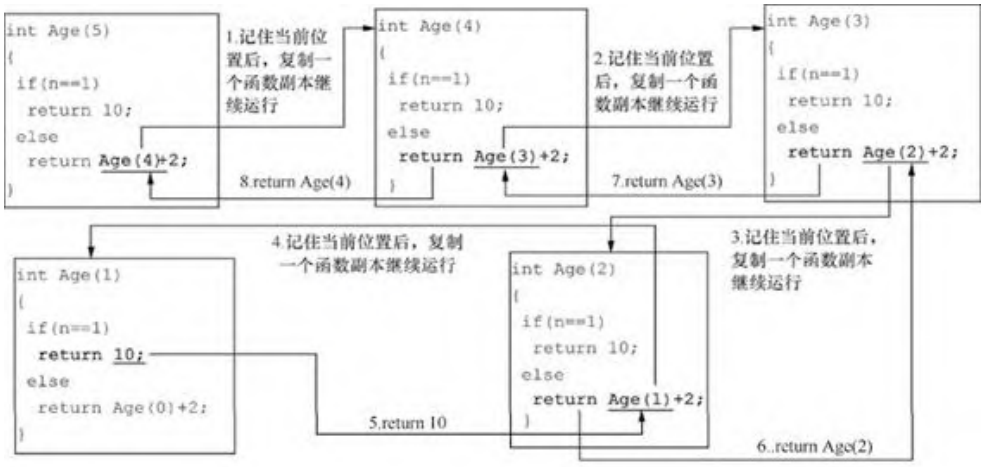


图6.4

递归算法是把一个“大问题”转化为规模较小且相似的“小问题”，当最小的一个或几个“小问题”直接推出解后，再逐步回归推出“大问题”的解。

递归必须有结束递归过程的条件。

递归程序简洁、易读，但递归过程的实现决定了递归算法的效率往往很低，费时且需要较多内存空间。在实际编程中，应优先考虑效率更高的递推法。

6.5.1 逆序字符（reverse）

【题目描述】

用getchar()函数和putchar()函数连续输入5个字符，逆序输出。

【输入格式】

输入5个字符。

【输出格式】

逆序输出5个字符。

【输入样例】

abcde

【输出样例】

edcba



输入/输出可以用getchar()和putchar()两个函数来实现，那么如何通过递归来实现呢？可以这样想：例如输入A、B、C、D、E这5个字符，可以直接用getchar()函数后就用putchar()函数，这样会直接输出ABCDE。现在假设A、B、C、D、E分别代表5个人，那么可以这样认为。

A说：让我输出可以，必须B先输出。　　⇐---　递推

B说：让我输出可以，但要C先输出。　　⇐---　递推

C说：让我输出可以，但要D先输出。　　⇐---　递推

D说：让我输出可以，但要E先输出。　　⇐---　递推

轮到E时，因为后面再没有字符了，所以输出自己！

看到E输出，则D输出。　　⇐---　回归

看到D输出，则C输出。　　⇐---　回归

看到C输出，则B输出。　　⇐---　回归

看到B输出，则A输出。　　←---　回归

所以，任何递归程序实际上包括递推和回归两部分。

请继续思考递归的结束条件是什么？用什么来控制递归的次数？

参考程序如下所示。

```
1 //逆序字符
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 void Reverse(int n)
6 {
7   char next;
8   if (n<=1)          //递归的结束条件
9   {
10    next=getchar();//从键盘获得一个字符
11    putchar(next); //因为只剩最后一个字符了，所以这里直接输出即可
12  }
13  else
14  {
15    next=getchar();//从键盘获得一个字符
16    Reverse(n-1);  //递归调用输入的下一个字符
17    putchar(next); //只有等到n-1个字符都被输出后，该字符才被输出
18  }
19 }
20
21 int main()
22 {
23   Reverse(5);
24   return 0;
25 }
```

6.5.2　逆序数（reverse）

【题目描述】

输入一个非负整数，输出这个数的逆序数（不考虑前导0的问题）。

【输入格式】

输入一个非负整数。

【输出格式】

输出这个数的逆序数。

【输入样例】

1000

【输出样例】

0001

【算法分析】

因为是整数，所以不适合使用6.5.1题中的getchar()和putchar()函数解题。

可以这样想：程序应该是先输出该数的最右边一个数字，如果余下的数字不为0，则继续输出余下数字的最右边一个数字，直至全部输出完毕。很明显，余下数字为0为递归结束条件。

参考程序如下所示。

```
//逆序数
#include <bits/stdc++.h>
using namespace std;

int Turn(int n)
{
  if (n>=10)              //如果有多位数
  {
    printf("%d",n%10);    //先输出最右边一个数字
    Turn(n/10);           //再进入递归，输出除去最右边一个数字后的数字
  }
  else
    printf("%d",n);       //如果只剩一个数字了，直接输出该数字
}

int main()
{
  int n;
  scanf("%d",&n);
  Turn(n);
  return 0;
}
```

6.5.3 求阶乘（factorial）

【题目描述】

用递归方法求$n!$，例如当$n=5$时，$n!=1\times2\times3\times4\times5=120$。

【输入格式】

输入一个整数n。

【输出格式】

输出n!的值，保证结果不超过64位整数。

【输入样例】

5

【输出样例】

5!=120

【算法分析】

如果要求5！的值，必须先求出4！的值后再乘以5；如果要求4！的值，必须先求出3！的值后再乘以4……

6.5.4　最大公约数和最小公倍数

【题目描述】

已知计算两个整数的最大公约数的递归公式是：

$$\mathrm{Gcd(m,n)} = \begin{cases} \mathrm{m} & \mathrm{n} = 0 \\ \mathrm{Gcd(n, m\%n)} & \mathrm{n} \neq 0 \end{cases}$$

两个整数的最小公倍数＝两个整数的乘积/两个整数的最大公约数。试求n个整数的最大公约数和最小公倍数。

【输入格式】

第一行输入一个整数n（$n \leqslant 12$），表示有n个正整数（不超过100）。

第二行输入n个整数。

【输出格式】

输出n个整数的最大公约数和最小公倍数，两数间以一个空格间隔。

【输入样例】

4

9 12 30 15

【输出样例】

3 180

6.5.5　复杂算式（f）

【题目描述】

已知$f(x,n)=\sqrt{n+\sqrt{(n-1)+\sqrt{(n-2)+\sqrt{\cdots+2+\sqrt{1+x}}}}}$，输入$x$和$n$的值，计算$f$（$x$，$n$)的值。

【输入格式】

输入一个双精度浮点数x和一个整数n（$1 \leqslant n < 100$）。

【输出格式】

输出f（x，n)的值，保留小数点后两位。

【输入样例】

3.6 10

【输出样例】

3.68

6.5.6　母牛数（cow）

【题目描述】

有一头母牛，它每年年初生一头小母牛。每头小母牛从第4年开始，每年年初也生一头小母牛。请编程算出在第n年的时候共有多少头母牛？（题目来源：HDU 2018）

【输入格式】

有多组测试数据，每组测试数据占一行，为一个整数n（$0<n \leqslant 60$）。

【输出格式】

每组测试数据的结果占一行，即输出在第n年的时候母牛的数量。

【输入样例】

4

5

【输出样例】

4

6

【算法分析】

设*Cow*[*n*]为第*n*年的时候母牛的数量，根据题意可推出递归表达式：

$$Cow[n]=\begin{cases} n & (n \leqslant 4) \\ Cow[n-1]+Cow[n-3] & (n>4) \end{cases}$$

但是如果由此表达式直接写递归代码，会因测试数据过多、过大而导致运行超时。以计算*Cow*[9]为例，如图6.5所示，可以发现许多值例如*Cow*[5]在递归过程中反复计算了多次，这浪费了大量时间。

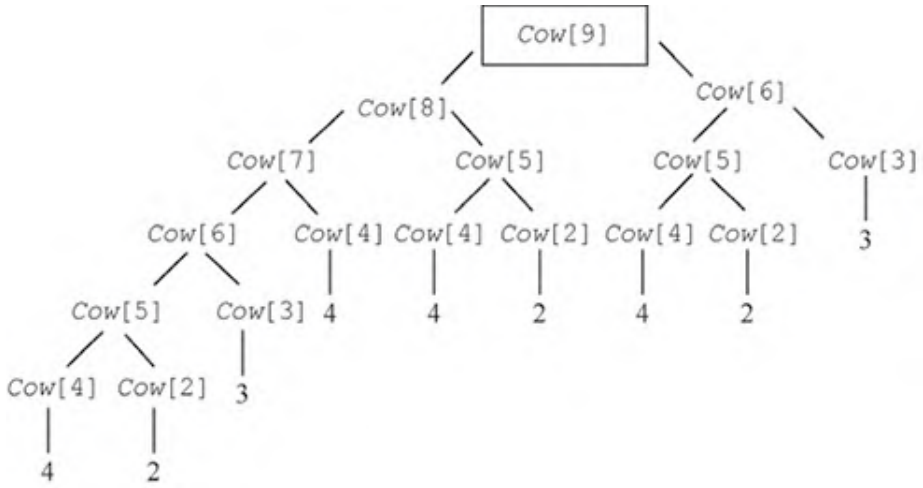


图6.5

很自然地想到：如果递归时将计算好的值保存起来，例如定义一个数组num[n]，计算出Cow(i)的值后即保存到num[i]中，当再次递归计算Cow(i)时，无须继续递归，直接取num[i]的值返回即可。

改进后的参考程序如下所示。

```
 1 //母牛数
 2 #include<bits/stdc++.h>
 3 using namespace std;
 4 typedef long long ULL;              //typedef定义long long的别名为
ULL
 5
 6 ULL num[65];                       //此处long long就可表示为ULL
 7
 8 ULL Cow(int n)
 9 {
10   if (n<=4)
11     return n;
12   if (num[n])                       //如果该值之前已经求出
13     return num[n];                  //无须递归，直接取值
14   else
15     num[n]=Cow(n-1)+Cow(n-3);       //递归算出的值先存入num[n]
16   return num[n];                    //再返回结果num[n]
17 }
18
19 int main()
20 {
21   int n;
22   while (scanf("%d",&n)==1)
23     printf("%lld\n",Cow(n));
24   return 0;
25 }
```

6.5.7 矩阵行走（walk）

【题目描述】

有一个$n\times m$的矩阵，问从矩阵左上角走到右下角有多少条不同的路径。

【输入格式】

输入一行两个正整数 n和m（$1\leqslant n\leqslant 20$，$1\leqslant m\leqslant 20$）。

【输出格式】

输出一个正整数，即路径数（同一路径不允许重复走，只能向下或向右走）。

【输入样例】

6 4

【输出样例】

210

6.5.8 计算函数值（f）

【题目描述】

输入x和n，递归计算函数$px(x, n)=x-x^2+x^3-x^4+\cdots+(-1)^{n-1}x^n$的值。

【输入格式】

输入两个整数x和n（$1\leqslant x\leqslant 10$，$1\leqslant n\leqslant 15$）。

【输出格式】

输出函数的值，保留小数点后两位。

【输入样例】

2 3

【输出样例】

6.00

【笔试测验】请手动计算程序输入2 1 3的结果。

```
#include <bits/stdc++.h>
using namespace std;

void Fun(int a, int b, int c)
{
  if (a>b)
    Fun(c,a,b);
  else
    cout<<a<<','<<b<<','<<c<<endl;
}

int main()
{
  int a, b, c;
  cin>>a>>b>>c;
  Fun(a,b,c);
  return 0;
}
```

【笔试测验】请手动计算程序运行的结果。

```
#include <bits/stdc++.h>
using namespace std;

int ACK(int M,int N)
```

```
 5 {
 6   int v;
 7   if (M==0)
 8      v=N+1;
 9   else if (N==0)
10      v=ACK(M-1,1);
11   else
12      v=ACK(M-1,ACK(M,N-1));
13   return v;
14 }
15
16 int main()
17 {
18   printf("%d\n",ACK(1,1));
19   return 0;
20 }
```

【笔试测验】请手动计算程序输入3 4的结果。

```
 1 #include <bits/stdc++.h>
 2 using namespace std;
 3
 4 int Fun(int a,int b)
 5 {
 6   if (b==1)
 7   return a;
 8   else
 9   {
10     int c=Fun(a,b/2);
11     return (b%2) ? c*c*a : c*c;
12   }
13 }
14
15 int main()
16 {
17   int X,n;
18   scanf("%d%d",&X,&n);
19   printf("%d\n",Fun(X,n));
20   return 0;
21 }
```

6.5.9 汉诺塔1（hanoi1）

【题目描述】

如图6.6所示，有3根针，其中A针上穿好了由大到小排列的64个金片，不论白天黑夜，总有一个和尚按照下面的法则移动金片：一次只移动一个，不管金片在哪根针上，小金片必须在大金片上面。当所有的金片都从A针移到C针上时就完成任务。这就是所谓汉诺塔。请编程求出将A针上所有金片移到C针上的步骤。

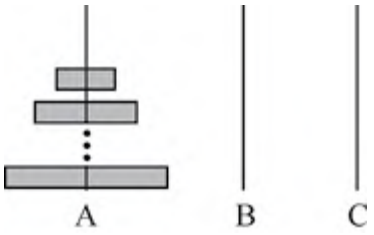


图6.6

图6.6

【输入格式】

输入一个整数*n*，表示有*n*（*n*不超过5）个金片。

【输出格式】

输出所有步骤，每一个步骤占一行。

【输入样例】

3

【输出样例】

A->C

A->B

C->B

A->C

B->A

B->C

A->C



当A针上有64个金片时，最少需要移动18 446 744 073 709 551 615次金片，即如果一秒移动一个金片，则需要移动几千亿年。所以，受客观条件的限制，*n*的值不宜过大。

假设有4个金片，4个和尚各负责一个金片，如图6.7所示。

如果直接让老和尚把4个金片从A针移到C针，老和尚显然做不到，所以老和尚说：“只要大和尚把上面3个金片从A针移到B针，我就能把金片4移到C针，然后大和尚再把刚移到B针上的3个金片移到C针就好啦。”他想象的移动过程如图6.8所示。

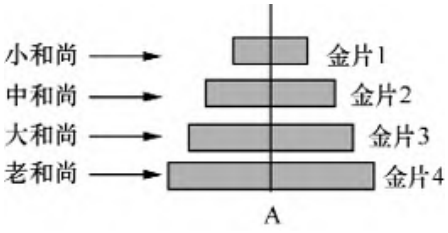


图6.7

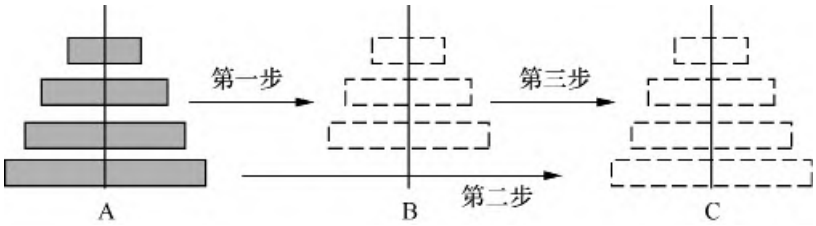


图6.8

大和尚无法直接把上面3个金片从A针移到B针，所以大和尚说：“只要中和尚把上面2个金片从A针移到C针，我就能把金片3从A针移到B针，然后中和尚再把刚移到C针上的2个金片移到B针上就好啦。”他想象的移动过程如图6.9所示。

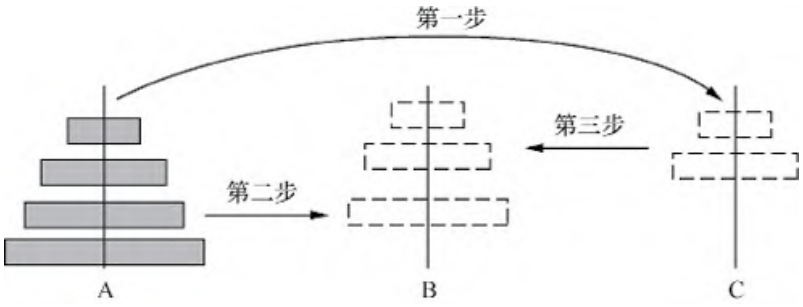


图6.9

中和尚无法直接将上面2个金片从A针移到C针，所以小和尚要先将金片1从A针移到B针，然后中和尚就可以将金片2移到C针，然后小和尚再将金片1从B针移到C针。他想象的移动过程如图6.10所示。

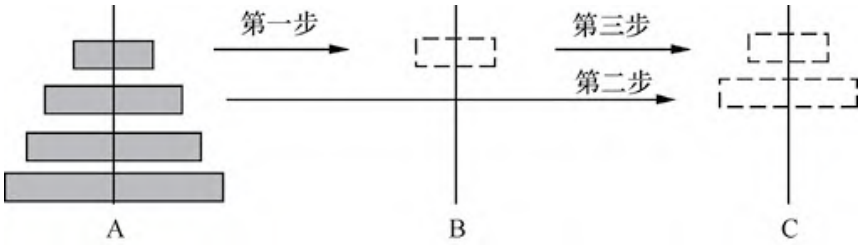


图6.10

这时大和尚就可以将金片3从A针移到B针，然后小和尚将C针的金片1移到A针，中和尚将C针的金片2移到B针……

显然这可以用从高阶向低阶转化的递归函数来实现，其递归函数的原型为void H(int n,char A,char B,char C)，其中n为金片数，A代表初始针，B代表过渡针，C代表目标针，即将n个金片从A针借助B针移到C针。

参考程序如下所示。请注意，该程序只是展示出了金片的移动次序，并没有真正移动金片。

```
1 //汉诺塔1
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 void Move(int n,char A,char B,char C)//n个金片从A针借助B针移到C针
6 {
7   if (n==1)                              //当金片只剩1个时
```

```
 8      printf("%c->%c ",A,C);              //移动金片从A针到C针
 9    else
10    {
11      Move(n-1,A,C,B);                    //递归，金片数-1，3个针换位置
12      printf("%c->%c ",A,C);              //移动金片从A针到C针
13      H(n-1,B,A,C);                       //递归，金片数-1，3个针换位置
14    }
15 }
16
17 int main()
18 {
19    int m;
20    scanf("%d",&m);                       //输入金片数
21    Move(m,'A','B','C');
22    return 0;
23 }
```

输出结果即移动过程如图6.11所示。



图6.11

图6.12所示是当金片数为4时，该递归程序的执行顺序。

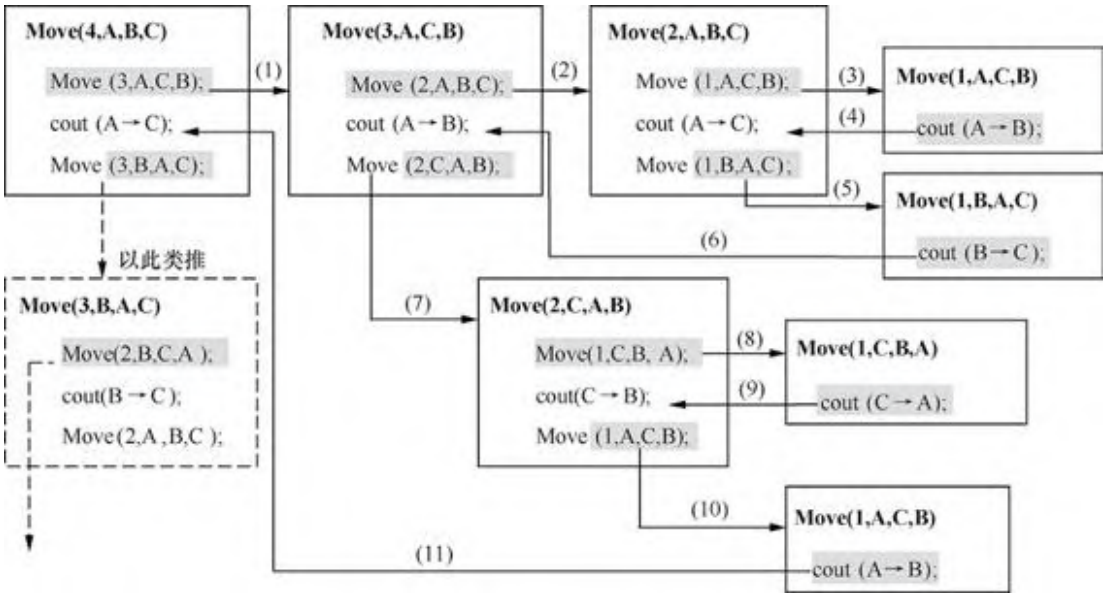


图6.12

6.5.10　汉诺塔2（hanoi2）

【题目描述】

将*N*个金片从汉诺塔的A针移到C针，最少需要移动多少次？

【输入格式】

输入一个正整数*N*。

【输出格式】

输出最少需要移动的次数。

【输入样例】

3

【输出样例】

7

第07章　阶段检测2

7.1　笔试检测

（1）请手动计算程序运行的结果。

```
#include <bits/stdc++.h>
using namespace std;

void Reverse(int a[],int n)
{
  for (int i=0; i<n/2; i++)
  {
    int t=a[i];
    a[i]=a[n-1-i];
    a[n-1-i]=t;
  }
}

int main()
{
  int b[10]={1,2,3,4,5,6,7,8,9,10};
  int i,s=0;
  Reverse(b,8);
  for (i=6; i<10; i++)
    s+=b[i];
  cout<<s<<endl;
  return 0;
}
```

（2）请手动计算程序输入3 2 7 5 17 19 11 20 22 24的结果。

```
#include <bits/stdc++.h>
using namespace std;
int a[11];

void Fun(int n)
{
  int min;
```

```
 8   for (int i=1; i<n; i++)
 9   {
10     min = i;
11     for (int j=i+1; j<=n; j++)
12       if (a[j] < a[min])
13         min = j;
14     if (min != i)
15       swap(a[i],a[min]);
16   }
17 }
18
19 int main()
20 {
21   for (int i=1; i<=10; i++)
22     scanf("%d",&a[i]);
23   Fun(10);
24   for (int i=1; i<=10; i++)
25     printf("%d ",a[i]);
26   return 0;
27 }
```

（3）请手动计算程序输入3的结果。

```
 1 #include <bits/stdc++.h>
 2 using namespace std;
 3
 4 int main()
 5 {
 6   int n,i,j,k,cnt=0;
 7   int a[4][4];
 8   cin>>n;
 9   for (k=1; k<=n; k++)
10     if (k%2)
11       for (j=1; j<=k; j++)
12       {
13         i=k+1-j;
14         a[i][j]=++cnt;
15         a[n+1-i][n+1-j]=n*n+1-cnt;
16       }
17     else
18       for (j=k; j>=1; j--)
19       {
20         i=k+1-j;
21         a[i][j]=++cnt;
22         a[n+1-i][n+1-j]=n*n+1-cnt;
23       }
24   for (i=1; i<=n; i++)
25     for (j=1; j<=n; j++)
```

```
26        cout<<a[i][j]<<' ';
27    return 0;
28 }
```

（4）请手动计算程序输入6 5 3 2 7 9 8的结果。

```
 1 #include <bits/stdc++.h>
 2 using namespace std;
 3
 4 int Data[11],n;
 5
 6 void Shell(int a[], int n)
 7 {
 8   int temp,j;
 9   for (int d=n/2; d>=1; d=d/2)
10     for (int i=d; i<n; i++)
11     {
12       temp=a[i];
13       for (j=i-d; (j>=0) && (a[j]>temp); j=j-d)
14         a[j+d]=a[j];
15       a[j+d]=temp;
16     }
17   for (int i=0; i<n; ++i)
18     printf("%d ",a[i]);
19 }
20
21 int main()
22 {
23   scanf("%d",&n);
24   for (int i=0; i<n; ++i)
25     scanf("%d",&Data[i]);
26   Shell(Data,n);
27   return 0;
28 }
```

（5）请手动计算程序输入10 5 11 12 8 6 5 3 2 9 7 15的结果。

```
 1 #include <bits/stdc++.h>
 2 using namespace std;
 3 int a[11],k,n,lc;
 4
 5 int Fun(int L,int R)
 6 {
 7   if (R>=L)
```

```
 8   {
 9     int m=(L+R)/2;
10     if (k==a[m])
11     {
12       printf("%d\n",m);
13       return 0;
14     }
15     else if (k<a[m])
16       Fun(L,m-1);
17     else
18       Fun(m+1,R);
19   }
20   else
21   {
22     printf("-1\n");
23     return 0;
24   }
25 }
26
27 int main()
28 {
29   cin>>n>>k;
30   for (int i=1; i<=n; ++i)
31     cin>>a[i];
32   int i=2;
33   while (i<n)
34   {
35     lc=n;
36     for (int j=n; j>=i; j--)
37       if (a[j]<a[j-1])
38       {
39         swap(a[j],a[j-1]);
40         lc=j;
41       }
42     i=lc;
43   }
44   Fun(1,n);
45   return 0;
46 }
```

（6）请手动计算程序输入10 10 9 -8 -7 6 5 4 3 2 1的结果。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int a[11];
5
6 void QS(int i,int j)
```

```
 7 {
 8   int m=i,n=j;
 9   int k=a[(i+j)/2];
10   while (m<=n)
11   {
12     While (a[m]<k && m<j)  m++;
13     While (a[n]>k && n>i)  n--;
14     if (m<=n)
15     {
16       swap(a[m],a[n]);
17       m++;
18       n--;
19     }
20   }
21   if (m<j)  QS(m,j);
22   if (n>i)  QS(i,n);
23 }
24
25 int main()
26 {
27   int n;
28   scanf("%d",&n);
29   for (int i=1; i<=n; i++)
30     scanf("%d",&a[i]);
31   QS(1,n);
32   for (int i=1; i<=n; i++)
33     printf("%d ",a[i]);
34   return 0;
35 }
```

（7）请手动计算程序输入10 7 1 4 3 2 5 9 8 0 6的结果。

```
 1 #include <bits/stdc++.h>
 2 using namespace std;
 3
 4 int n, d[100];
 5 bool v[100];
 6
 7 int main()
 8 {
 9   scanf("%d", &n);
10   for (int i=0; i<n; ++i)
11   {
12     scanf("%d", d+i);
13     v[i]=false;
14   }
15   int cnt=0;
16   for (int i=0; i<n; ++i)
```

```
17      if (!v[i])
18      {
19        for (int j=i; !v[j]; j=d[j])
20          v[j]=true;
21          ++cnt;
22      }
23    printf("%d\n", cnt);
24    return 0;
25  }
```

（8）请手动计算程序运行的结果。

```
 1  #include <bits/stdc++.h>
 2  using namespace std;
 3  #define N 10
 4  int S,I;
 5
 6  int Fun(int I1)
 7  {
 8    int S1=N;
 9    for (int J1=(N-1); J1>=(N-I1+1); J1--)
10      S1=S1*J1/(N-J1+1);
11    return S1;
12  }
13
14  int main()
15  {
16    S=N+1;
17    for (I=2; I<=N; I++)
18      S=S+Fun(I);
19    printf("S=%d\n",S);
20    return 0;
21  }
```

7.2 上机检测



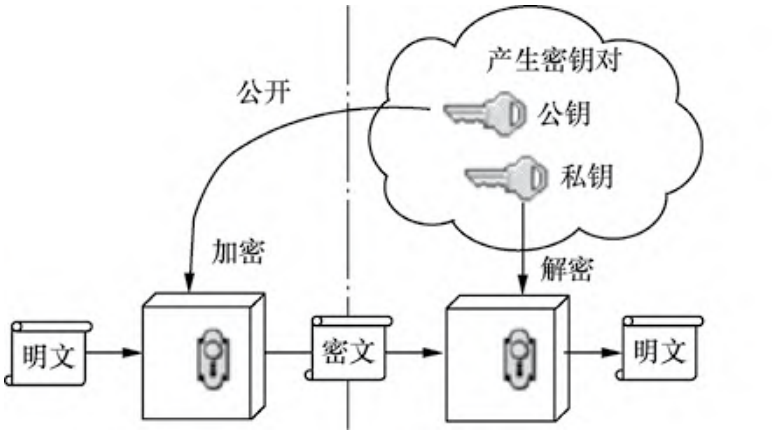
扫码看视频

7.2.1　RSA加密算法（RSA）

【题目描述】

RSA加密算法基于十分简单的数论事实：将两个大质数相乘十分容易，但想要对其乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

请编程输入一个大于1的整数，打印出它的质数分解式。



【输入格式】

输入一个大于1的整数。

【输出格式】

输出它的质数分解式。

【输入样例】

75

【输出样例】

75=3*5*5

7.2.2　单词排序（word）

【题目描述】

输入一行英文，单词与单词之间以逗号或空格间隔，结尾以“!”和“.”为标志，试将所有的单词按ASCII值由小到大排序后输出。

【输入格式】

输入一行英文，不超过100个单词，每个单词的长度不超过15个字符。

【输出格式】

第一行输出单词数，随后一行一个单词，按ASCII值由小到大排序后输出。

【输入样例】

hi,I am Jack!

【输出样例】

4

I

Jack

am

hi

7.2.3 回文日期（date）

【题目描述】

8位数字可以表示一个唯一确定的日期，例如2016年11月19日表示为20161119，2010年1月2日表示为20100102。现在，小光想知道在指定的两个日期之间（包含这两个日期自身）有多少个真实存在的日期

是回文日期（例如2001年10月2日可以表示为20011002，而20011002无论正序还是倒序都是一样的）。

【输入格式】

输入两行，每行包括一个8位数字（首位数字不为0），分别表示真实存在的起始日期和终止日期。

【输出格式】

输出一行，包含一个整数，表示在起始日期和终止日期之间有多少个日期是回文日期。

【输入样例1】

20110101

20111231

【输出样例1】

1

【输入样例2】

20000101

20101231

【输出样例2】

2

【样例说明】

对于输入样例1，符合条件的日期是20111102。

对于输入样例2，符合条件的日期是20011002和20100102。

7.2.4　多项式输出（poly）

【题目描述】

一元n次多项式可表示为$f(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$。

其中，$a_n\neq0$，a_ix^i称为i次项，a_i称为i次项的系数。给出一个一元多项式各项的次数和系数，请按照如下规定的格式要求输出该多项式。（题目来源：NOIP 2009）

（1）多项式中自变量为x，从左到右按照次数递减顺序给出多项式。

（2）多项式中只包含系数不为0的项。

（3）如果多项式n次项系数为正，则多项式开头不出现“+”；如果多项式n次项系数为负，则多项式以“-”开头。

（4）对于不是最高次的项，以“+”或者“-”连接此项与前一项，分别表示此项系数为正或者系数为负。紧跟一个正整数，表示此项系数的绝对值（如果一个高于0 次的项，其系数的绝对值为1，则无须输出1）。如果x的指数大于1，则接下来紧跟的指数部分的形式为“$x\hat{}b$”，其中b为x的指数；如果x的指数为1，则接下来紧跟的指数部分形式为“x”；如果x的指数为0，则仅需输出系数即可。

（5）多项式中，多项式的开头、结尾不含多余的空格。

【输入格式】

输入第一行为一个整数n（$1 \leqslant n \leqslant 100$），表示一元多项式的次数。

第二行有$n+1$个整数，其中第i个整数表示第$n-i+1$次项的系数，多项式各次项系数的绝对值均不超过100，每两个整数之间以空格隔开。

【输出格式】

输出共一行，按题目所述格式输出多项式。

【输入样例1】

5

100 -1 1 -3 0 10

【输出样例1】

100x^5-x^4+x^3-3x^2+10

【输入样例2】

3

-50 0 0 1

【输出样例2】

-50x^3+1

7.2.5 最大公约数和最小公倍数问题（b）

【题目描述】

输入两个正整数x_0和y_0，求出满足下列条件的P、Q的个数。

（1）P、Q是正整数。

（2）P、Q以x_0为最大公约数，以y_0为最小公倍数。

【输入格式】

输入两个正整数x_0和y_0（$2 \leqslant x_0 \leqslant 100\,000$，$2 \leqslant y_0 \leqslant 1\,000\,000$）。

【输出格式】

输出一个整数，即满足题目条件的P、Q的个数。

【输入样例】

3 60

【输出样例】

4

【样例说明】

满足条件的所有可能的P、Q的个数为4，分别为3和60、15和12、12和15、60和3。

7.2.6　子矩阵求和（matrix）

【题目描述】

有一个n行m列的矩阵，有q次询问，每次询问一个子矩阵内所有数的和，求q次询问的结果。

【输入格式】

第一行为3个整数n、m、q（$1 \leqslant n,m \leqslant 500$，$q \leqslant 1000\,000$）。

随后为n行，每行m个0～100的整数。

随后为q行，每行4个整数即x_1、y_1、x_2、y_2，表示要询问的子矩阵。

【输出格式】

输出q行结果。

【输入样例】

3 5 2

1 2 3 4 5

5 6 7 8 9

5 4 3 2 1

1 1 3 5

1 1 2 2

【输出样例】

65

14

7.2.7 石头剪刀布（rps）

【题目描述】

“石头剪刀布”是常见的猜拳游戏：石头胜剪刀，剪刀胜布，布胜石头。如果两个人手势一样，则不分胜负。现在，石头剪刀布的升级版游戏又增加了两个新手势斯波克、蜥蜴人[1]。

这5种手势的胜负关系如表7.1所示（阴影部分的胜负关系请自行推导）。

表7.1

乙 甲对乙的结果 甲	剪刀	石头	布	蜥蜴人	斯波克
剪刀	平	输	赢	赢	输
石头		平	输	赢	输
布			平	输	赢
蜥蜴人				平	赢
斯波克					平

现在，小A和小B尝试玩这种升级版的猜拳游戏。

已知他们的出拳方式都是有周期性规律的，但周期长度不一定相等。

例如，如果小A以“石头→布→石头→剪刀→蜥蜴人→斯波克”长度为6的周期出拳，那么他的出拳序列就是“石头→布→石头→剪刀→蜥蜴人→斯波克→石头→布→石头→剪刀→蜥蜴人→斯波克→……”；而如果小B以“剪刀→石头→布→斯波克→蜥蜴人”长度为5的周期出拳，那么他的出拳序列就是“剪刀→石头→布→斯波克→蜥蜴人→剪刀→石头→布→斯波克→蜥蜴人→……”。

已知小A和小B一共进行N次猜拳。

每一次赢的人得1分，输的人得0分；平局两人都得0分。

现请你统计N次猜拳结束之后两人的得分。

【输入格式】

第1行包含3个整数N、NA、NB（取值范围均为（0,200]），分别表示共进行N次猜拳、小A出拳的周期长度、小B出拳的周期长度。数与数之间以一个空格分隔。

第2行包含NA个整数，表示小A出拳的规律。

第3行包含NB个整数，表示小B出拳的规律。

其中，0表示“剪刀”，1表示“石头”，2表示“布”，3表示“蜥蜴人”，4表示“斯波克”。

【输出格式】

输出一行，包含两个整数，以一个空格分隔，分别表示小A、小B的得分。

【输入样例】

10 5 6

0 1 2 3 4

0 3 4 2 1 0

【输出样例】

6 2

7.2.8 公约数最大（com）

【题目描述】

输入n个正整数，从中任取出k个数，使这k个数的最大公约数最大。

【输入格式】

第一行两个整数，即n和k（$k \leqslant n \leqslant 50000$），随后一行有$n$个整数（均不大于100 000）。

【输出格式】

输出一个整数，即k个数的最大公约数最大的数。

【输入样例】

4 3

123 369 999 36

【输出样例】

9

7.2.9 无线网络发射器选址（wireless）

【题目描述】

假设城市的布局为由严格平行的129条东西向街道和129条南北向街道形成的网格形状，并且相邻的平行街道之间的距离都是恒定值1。东西向街道从北到南依次编号为0,1,2,…,128，南北向街道从西到东依次编号为0,1,2,…,128。

东西向街道和南北向街道相交形成路口，规定编号为x的南北向街道和编号为y的东西向街道形成的路口的坐标是(x, y)。在某些路口存在一定数量的公共场所。

现在政府要安装一个传输距离为d的大型无线网络发射器，该无线网络发射器的覆盖范围是一个以该无线网络发射器安装地点为中心，边长为$2\times d$的正方形。覆盖范围包括正方形边界。

图7.1所示是$d=1$的无线网络发射器的覆盖范围示意。

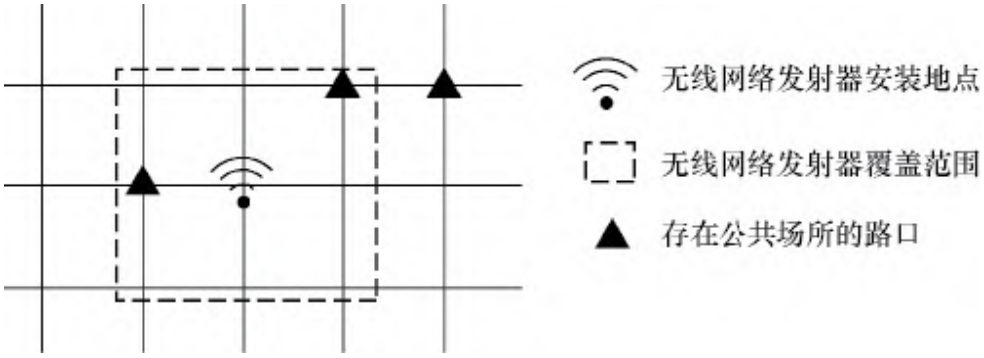


图7.1

请你帮助工作人员在城市内找到合适的安装地点，使得覆盖的公共场所最多。

【输入格式】

第一行包含一个整数d（$1 \leqslant d \leqslant 20$），表示无线网络发射器的传输距离。

第二行包含一个整数n（$1 \leqslant n \leqslant 20$），表示有公共场所的路口数目。

接下来n行，每行给出3个整数x、y、k（$0 \leqslant x \leqslant 128$，$0 \leqslant y \leqslant 128$，$0 < k \leqslant 1\,000\,000$），中间以一个空格分隔，分别代表路口的坐标$(x, y)$和该路口公共场所的数量。同一坐标只给出一次。

【输出格式】

输出一行，包含两个整数，以一个空格分隔，分别表示能覆盖最多公共场所的安装地点方案数，以及能覆盖的最多公共场所的数量。

【输入样例】

```
1

2

4 4 10

6 6 20
```

【输出样例】

```
1 30
```

7.2.10 机器翻译（translate）

【题目描述】

机器翻译软件的工作原理很简单，它只是从头到尾依次将每个英文单词用对应的中文来替换。对于每个英文单词，软件会先在内存中查找这个单词对应的中文，如果内存中有，软件就会用它进行翻译；如果内存中没有，软件就会在外存中的词典内查找，查出单词对应的中文然后翻译，并将这个单词和译义放入内存，以备后续的查找和翻

译。假设内存中有M个单元，每个单元能存放一个单词和译义。每当软件将一个新单词存入内存，如果当前内存中已存入的单词数不超过$M-1$，软件会将新单词存入一个未使用的内存单元；如果当前内存中已存入M个单词，软件会清空最早进入内存的那个单词，腾出内存单元来存放新单词。

假设一篇英语文章有N个单词，机器翻译软件需要去外存查找多少次词典？假设在翻译开始前，内存中没有任何单词。

【输入格式】

输入共两行。每行中两个数之间以一个空格分隔。

第一行为两个正整数M和N，代表内存容量和文章中单词的个数。

第二行为N个非负整数，按照文章的顺序，每个数（大小不超过1000）代表一个英文单词。文章中两个单词是同一个单词的条件是当且仅当它们对应的非负整数相同。

【输出格式】

输出共一行，包含一个整数，为机器翻译软件需要查词典的次数。

【输入样例】

3 7

1 2 1 5 4 4 1

【输出样例】

5

【数据范围】

对于10%的数据有$M=1$，$N \leqslant 5$。

对于100%的数据有$0 < M \leqslant 100$，$0 < N \leqslant 1\,000$。

7.2.11 玩具谜题（toy）

【题目描述】

玩具小人们围成了一个圈，它们有的面朝圈内，有的面朝圈外，如图7.2所示。



图7.2

第1个玩具小人singer告诉小光一个谜题：“你的眼镜藏在我左数第3个玩具小人的右数第1个玩具小人的左数第2个玩具小人那里。”

小光发现，这个谜题中玩具小人的朝向非常关键，因为面朝圈内和面朝圈外的玩具小人的左、右方向是相反的：对于面朝圈内的玩具小人，它的左边是顺时针方向，右边是逆时针方向；而对于面向圈外的玩具小人，它的左边是逆时针方向，右边是顺时针方向。

小光一边艰难地辨认着玩具小人，一边数着：

“singer面朝圈内，左数第3个是archer。

“archer面朝圈外，右数第1个是thinker。

“thinker面朝圈外，左数第2个是writer。

“所以眼镜藏在writer这里！”

为了防止下一次找不到眼镜，小光决定编写一个程序来解决类似的谜题。具体如下。

有 n 个玩具小人围成一圈，已知它们的职业和朝向。现在第1个玩具小人告诉小光一个包含m条指令的谜题，其中第i条指令形如“左数/右数第s_i个玩具小人”。你需要输出依次数完这些指令后到达的玩具小人的职业。

【输入格式】

第一行有两个正整数n和m（$1 \leqslant n$，$m \leqslant 100\,000$），表示玩具小人的个数和指令的条数。

接下来n行，每行包含一个整数和一个字符串，以逆时针为顺序给出每个玩具小人的朝向和职业。其中0表示朝向圈内，1表示朝向圈外。保证不会出现其他的数。字符串长度不超过10且仅由小写字母构成，字符串不为空，并且字符串两两不同。

接下来m行，其中的第 i 行包含两个整数a_i、s_i（$1 \leqslant s_i<n$），表示第i条指令。若a_i=0，表示向左数s_i个人；若a_i=1，表示向右数s_i个人。保证a_i不会出现其他的数。

【输出格式】

输出一个字符串，表示从第1个读入的玩具小人开始，依次数完 m 条指令后到达的玩具小人的职业。

【输入样例】

```
7 3
0 singer
0 reader
0 investigator
1 thinker
1 archer
0 writer
1 doctor
0 3
1 1
0 2
```

【输出样例】

writer

7.2.12　cantor表（cantor）

【题目描述】

现代数学的著名证明之一是格奥尔格·康托尔（Georg Cantor）证明了有理数是可枚举的。他是用图7.3来证明这一命题的。

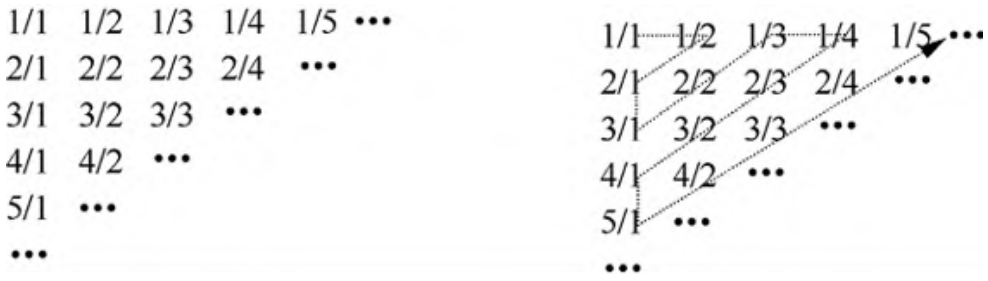


图7.3

我们以Z字形给上图的每一项编号。第一项是1/1，然后是1/2、2/1、3/1、2/2等。

【输入格式】

输入一个整数N（$1 \leqslant N \leqslant 10\,000\,000$）。

【输出格式】

输出图中的第N项。

【输入样例】

7

【输出样例】

1/4

7.2.13　计算器的改良（calc）

【题目描述】

Z1先生改良的计算器只要输入合法的一元一次方程，就可以将方程的结果（精确至小数点后3位）输出。所谓合法的一元一次方程只包含整数、小写字母及“+”“-”“=”这3个数学符号（“-”既可作减号，也可作负号）。方程中没有括号，也没有除号，方程中的字母表示未知数。例如：

$5x+2=8$

$9a-4+2=3a-2$

$12y-5=0$

无须考虑诸如$x--2x=6$或者$x+-6=8$这样的不合法方程。（题目来源：NOIP 2000）

【输入格式】

输入只有一行，即合法的一元一次方程，且一元一次方程有唯一的实数解。

【输出格式】

输出解。

【输入样例】

$6a-5+1=2-2a$

【输出样例】

$a=0.750$

7.2.14　维吉尼亚密码（vigenere）

【题目描述】

在密码学中，我们称需要加密的信息为明文，用M表示；称加密后的信息为密文，用C表示；而密钥是一种参数，是将明文转换为密文或将密文转换为明文的算法中输入的数据，记为k。在维吉尼亚密码中，

密钥$k=k_1k_2\cdots k_n$。当明文$M=m_1m_2\cdots m_n$时，得到的密文$C=c_1c_2\cdots c_n$，其中$c_i=m_i®k_i$。®运算的规则如表7.2所示。

表7.2

®	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

使用维吉尼亚密码加密在操作时需要注意以下内容。

（1）®运算忽略参与运算的字母的大小写，并保持字母在明文M中的大小写形式。

（2）当明文M 的长度大于密钥k的长度时，将密钥k重复使用。

例如，当明文M=Helloworld，密钥k=abc时，密文C=Hfnlpyosnd，如表7.3所示。

表7.3

明文	H	e	l	l	o	w	o	r	l	d
密钥	a	b	c	a	b	c	a	b	c	a
密文	H	f	n	l	p	y	o	s	n	d

【输入格式】

输出第一行为一个字符串，表示密钥 k，长度不超过100，其中仅包含大、小写字母。

第二行为经加密后的密文字符串，长度不超过1 000，其中仅包含大、小写字母。

【输出格式】

输出一个字符串，表示输入密钥和密文所对应的明文。

【输入样例】

CompleteVictory

Yvqgpxaimmklongnzfwpvxmniytm

【输出样例】

Wherethereisawillthereisaway

7.2.15　计算系数（factor）

【题目描述】

给定一个多项式$(by+ax)^k$，请求出多项式展开后$x^n \times y^m$项的系数。

【输入格式】

输入共一行，包含5个整数，分别为a、b、k、n、m，每两个整数之间以一个空格分隔。

【输出格式】

输出一个整数，表示所求的系数，这个系数可能很大，输出对10007取模后的结果。

【输入样例】

1 1 3 1 2

【输出样例】

3

【数据范围】

对于100%的数据，有$0 \leqslant k \leqslant 1\,000$，$0 \leqslant n$，$m \leqslant k$，且$n+m=k$，$0 \leqslant a$，$b \leqslant 1\,000\,000$。

7.2.16 大炮互攻（cannon）

【题目描述】

1 000×1 000的网格上有n门大炮，大炮只能倾斜45度开炮，方向不定。试计算有多少对大炮可以互相攻击。

【输入格式】

第一行输入一个整数n（$1 \leqslant n \leqslant 200\,000$），表示大炮数量。

随后n行，每行两个整数x_i和y_i（$1 \leqslant x_i$，$y_i \leqslant 1\,000$），表示大炮的坐标。

【输出格式】

输出一个整数。

【输入样例】

```
5
1 1
1 5
3 3
5 1
5 5
```

【输出样例】

```
6
```

7.2.17　解方程（equation）

【题目描述】

给出一个正整数n（$1<n \leqslant 2^{31}-1$），求当x和y都为正整数时，方程$sqrt(n) = sqrt(x) - sqrt(y)$的解中，$x$的最小值是多少？

【输入格式】

输入一个正整数n。

【输出格式】

输出一个满足条件的最小的x的解。

【输入样例】

4

【输出样例】

9

7.2.18　shlqsh数（shlqsh）

【题目描述】

我们把t_1、t_2（包括t_1、t_2）之间所有数的约数个数和n称为t_1、t_2的shlqsh数。试求出t_1、t_2的shlqsh数。

【输入格式】

输入仅一行，共有两个整数，即t_1、t_2（$1 \leqslant t_1 < t_2 \leqslant 10\,000\,000$）。

【输出格式】

输出一个整数，表示t_1、t_2的shlqsh数。

【输入样例】

2 6

【输出样例】

13

【样例说明】

2的约数有1、2 （2个）；

3的约数有1、3 （2个）；

4的约数有1、2、4 （3个）；

5的约数有1、5 （2个）；

6的约数有1、2、3、6 （4个）。

所以2、6 的shlqsh数为13。

【数据范围】

对于50 %的数据，保证有 $t_1, t_2 \leqslant 1\,000\,000$。

对于100%的数据，保证有 $t_1, t_2 \leqslant 10\,000\,000$。



学有余力的读者可以到各大题库网站找一些符合自己能力的题目来做。如果某些网站不支持“万能”头文件，那么可以按照如下形式写。

```
#include <iostream>      //用于数据流输入/输出

#include <cstdlib>       //用于定义杂项函数和内存分配函数
```

```
#include <cstdio>        //用于定义输入/输出函数
#include <cmath>        //用于定义数学函数
#include <cstring>        //用于字符串处理
#include <algorithm>      //STL通用算法
```

---

[1]斯波克、蜥蜴人均为《星际迷航》中的角色。

第08章　指针

C++语言里的指针的功能非常强大，它可以直接处理内存地址，完成许多复杂的任务，但指针的概念并不好理解。

8.1　地址和指针



扫码看视频

计算机对数据的存储和读取是通过地址来进行的，所谓地址，通俗地讲就是将存储空间（内存）的每一块都编了一个“门牌号”，这个“门牌号”就是内存的地址。所有的数据都要按照给定的“门牌号”存入相应的位置，只要知道了数据的“门牌号”，就可以找到该数据进行操作。例如，将字符串“All right”存入内存，如表8.1所示。

表8.1

地址	…	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	…
内容			A	l	l		r	i	g	h	t	

可以看到，字符“A”存在地址2003的位置上，字符“t”存在地址2011的位置上……（一个半角西文字符占一字节）。只要找到地址为2003的位置，就可以找到字符“A”。

指针是变量，专门用来存储内存的地址编号，例如2002等，这样就可以根据存储的地址编号找到该地址上存储的真实值了，如图8.1所示。



图8.1



地址是固定不变的，是内存的“门牌号”,地址的值为整数。

指针的值指向的是地址，它可以指向内存的任意位置，所以指针的值是可以改变的。可以根据指针的值找到内存的相应位置。

为了将指针变量与其他普通变量区分开，需要在变量名前加“*”。例如：

```
int *point;          //定义*point为指针变量
```

在*point前加的“int”，表示它会将该指针变量指向的位置当作int类型数据来处理。指向任何类型的指针变量均占4字节的长度，因为地址是用4字节表示的。

如图8.2所示，系统在可使用的内存中分配了一个4字节的空间（例如地址2020处）存放指针变量*point，其中point是指针，它保存的值是地址。假设当前值为2004，而地址2004处的值为100，则*point＝100。“*”可以理解为取point指向的地址上的真实值。



图8.2

8.2　指针变量的应用



扫码看视频

下面的程序是指针变量的简单应用。

```
 1 //指针变量的简单应用
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a=3,b=4;
 8   int *p1,*p2;                                    //定义两个指针变量*p1、
*p2
 9   p1=&a;                                         //把变量a的地址值赋给p1
10   p2=&b;                                         //把变量b的地址值赋给p2
11   printf("%d %d  %d %d\n",a,b,*p1,*p2);   //显示结果为3 4  3 4
12   return 0;
13 }
```



可以看到，指针变量*p1、*p2可以像普通变量一样使用。

第9行和第10行是将变量的地址值赋给指针，因为p1、p2存储的就是地址值，这与取a的地址值和取b的地址值类型匹配。

当第11行打印*p1、*p2的值时，打印的就是p1、p2指向的地址上的值，即a和b。

注意：*p1、*p2在开始定义时并未指向任何地方，只是当第9行和第10行执行语句完毕后才有了具体的指向。

例如将a、b的值通过指针的方式排序输出的程序如下所示。

```
 1 //两数排序
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int *p1,*p2,*p,a=3,b=4;
 8   p1=&a;                                    //p1指向存储变量a所在的地址
 9   p2=&b;                                    //p2指向存储变量b所在的地址
10   if (a<b)
11     p=p1, p1=p2, p2=p;                      //地址值与地址值互相交换
12   printf("%d %d\n",a,b);                    //输出3 4
13   printf("%d %d\n",*p1,*p2);                //输出4 3
14   return 0;
15 }
```



作为临时变量的p也为地址值，否则交换数据时会因为数据类型不匹配而发生意想不到的错误。

程序中a和b的值并未交换，它们仍保持原值，但p1和p2的值改变了，即交换两个指针变量的值。

前文讲到，当把一个变量作为参数传递给子函数时，无论子函数如何改变参数的值，实际原始变量值不变，因为传递给子函数的变量只是该变量的一个副本。

但下面这个指针变量作为函数参数的程序中，a和b的值竟然被改变了，为什么？

```
 1 //指针变量作为函数参数
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 void Swap(int *p1,int *p2)       //注意当指针变量作为参数时形参的形式
 6 {
 7   int temp;
 8   temp=*p1;                      //*p1是整数，所以可以赋值给整数temp
 9   *p1=*p2;
10   *p2=temp;
11 }
12
13 int main()
14 {
15   int *_point1,*_point2,a=3,b=4;
16   _point1=&a;
17   _point2=&b;                    //将a、b的地址赋给_point1、_point2
18   if (a<b)
19     Swap(_point1,_point2);       //将指针变量作为函数参数
20   printf("%d %d",a,b);
21   return 0;
22 }
```



和用数组名作为函数参数一样，该程序传递的参数是变量的地址，所以子函数里针对参数的操作是直接在该变量的地址上进行的。

不要将交换过程写成如下代码，因为temp指向的地址是不确定的，所以对*temp赋值会将未知地址上的值覆盖，这可能会破坏系统的正常运行。

```
1 int *temp;     //此时该指针变量里存储的值是未知的，指向的地址是未知的
2 *temp=*p1;     //将*temp指向的未知地址上的值改为*p1是"危险"的操作
3 *p1=*p2;
4 *p2=*temp;
```

我们也无法通过改变指针形参的值而使指针实参的值改变，例如下面的子函数代码。

```
1 void Swap(int *p1,int *p2)
2 {
3   int *p;
4   p=p1;p1=p2;p2=p;    //这样写无法改变main()函数中的a和b的值，请思考原因
5 }
```

下面的程序演示了函数在调用时的引用与取地址操作，所谓引用就是为传递的参数起一个别名。

```
1 //引用与取地址操作
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 void Swap1(int *x,int *y)          //取地址
```

```
 6 {
 7   int z=*x;
 8   *x=*y;
 9   *y=z;
10 }
11
12 void Swap2(int &x,int &y)          //引用
13 {
14   int z=x;
15   x=y;
16   y=z;
17 }
18
19 int main()
20 {
21   int a=1,b=-1;
22   Swap1(&a,&b);                    //函数要求输入为指针变量，需要取地址
符
23   printf("a=%d,b=%d\n",a,b);       //此处输出a=-1,b=1
24   Swap2(a,b);                      //引用不需要取地址符
25   printf("a=%d,b=%d\n",a,b);       //此处输出a=1,b=-1
26   return 0;
27 }
```

该程序调用Swap2的时候传了两个引用给它（x＝a，y＝b），即x只是a的一个别名，y只是b的一个别名，实际上都代表同一块内存空间。

引用相对于指针来说更高效、更简便，因为指针传参的实质还是传值调用，复制地址需要“开销”，而引用会直接被编译器优化。

【笔试测验】请手动算出程序输入5 5 3 2 6 1的结果。

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 void Swap(int &a,int &b)
5 {
6   int t;
7   t=a,a=b,b=t;
8 }
9
```

```
int main()
{
  int N,m1=2147483647,m2=2147483647,temp;
  cin>>N;
  for (int i=1; i<=N; ++i)
  {
    cin>>temp;
    if (m2>=m1 && temp<m2)
      m2=temp;
    else if (m2<m1 && temp<m1)
      m1=temp;
  }
  if (m1>m2)
   Swap(m1,m2);
   cout<<m1<<" "<<m2<<endl;
  return 0;
}
```

8.3 数组与指针



扫码看视频

定义指向数组元素的指针变量与定义指向变量的指针变量方法类似。例如：

```
int a[10], *p;
p=&a[0];
```

由于C++语言规定数组名代表数组中第一个元素的地址，因此下面两条语句等价。

```
p=&a[0];        //把数组a的首元素地址赋给p
p=a;            //把数组a的首元素地址赋给p
```

C++语言规定：如果指针变量p已指向数组中的一个元素，则p＋1指向该数组中的下一个元素（而不是简单地将p加1）。例如，数组元素是浮点数，因为每个浮点数占4字节的空间，所以
p＋1意味着使p的值加4字节，以使它指向下一个元素。如图8.3所示，p的值为2006，p＋1的值不是2006＋1，而是下一个元素的地址2010，所以p＋2＝2014、p＋5＝2026……

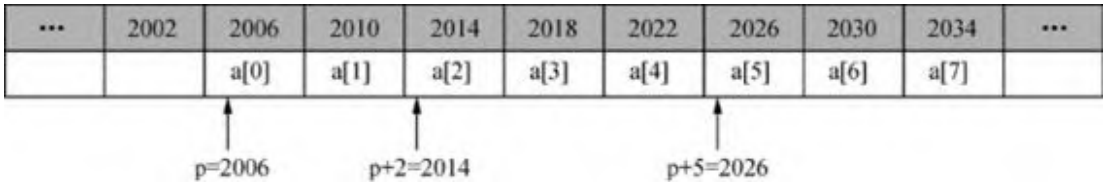


图8.3

以下程序使用了指针变量指向数组元素的方法。

```
 1 //指针变量指向数组元素的应用
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a[10]= {1,2,3,4,5,6,7,8,9,10};
 8   int *p=a;                    //可以在定义指针变量的同时赋初始值
 9   for (int i=0; i<10; i++)     //以下标方式输出
10     printf("%d",p[i]);
11
12   for (; p<(a+10); p++)        //以指针方式输出，p++表示指向下一个元素
13     printf("%d",*p);
14   return 0;
15 }
```



从第9行和第10行可以看出，指针变量p指向数组首元素后，相当于定义了数组p,可以使用下标的方式逐个输出数组a的值，因为p[i]和a[i]在内存中是重合的，对数组p的操作就是对数组a的操作。

如果第12行写成for(p=a;a<(p+10);a++)就错了，因为a为数组首地址，不是指针，指针可以改变值，而地址不可改，所以无法实现a++语句。

要注意指针变量指向的位置是否正确，例如通过指针变量输出数组a中10个元素的参考程序如下所示。

```
 1 //通过指针变量输出数组元素，这是一个错误的程序
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int *p,i,a[10];
 8   p=a;
 9   for (i=0; i<10; i++)
10   {
11     scanf("%d",p);                          //注意此处无须加取地址符"&"
12     p++;
13   }
14   printf("\n");
15   for (i=0; i<10; i++,p++)
16     printf("%d ",*p);
17   return 0;
18 }
```



这个程序运行输出的结果是错误的。因为经过第一个for循环读入数据后，p已指向了数组a的末尾，所以在执行第二个for循环时，p的起始值已经不是a而是a＋10了。

改正方法是在第二个for循环之前加一个赋值语句p=a;。

可以使用指针模拟Pascal独有的负下标数组，程序如下所示。

```
 1 //负下标数组
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main ()
 6 {
 7   int a[100],i,*p;
 8   for (i=0; i<100; i++)
 9     a[i]=i;                                    //赋初始值
10   p = &a[50];                                  //指针指向数组中间
11   for (i=-50; i<50; i++)
12     printf("%d ",p[i]);
13   return 0;
14 }
```

第09章　结构体

C++语言里提供的结构体可以使用户自定义数据类型，使之可以存储不同类型的数据作为一个整体来处理。

9.1　结构体及其应用



扫码看视频

假设有10 000个学生，每个学生的基本信息包括学号、姓名、性别、成绩等不同类型的数据，如何存储这些学生的信息并方便后续操作呢？最容易想到的是定义多个不同类型的数组来存储相应数据。

代码大致如下所示。

```
1 int ID[10000];              //定义整数数组ID[]保存每个学生的学号
2 char name[10000][50];       //定义字符串数组name[]保存每个学生的姓名
3 char sex[10000];            //定义字符数组sex[]保存每个学生的性别
4 float score[10000];         //定义浮点数数组score[]保存每个学生的成绩
```

这种存储方式将学生的完整信息分散到了不同的数组中，不仅过于烦琐，还不便于后期的管理和操作。而通过声明自定义结构体的方式，可以将每个学生的完整信息作为整体来处理。

代码大致如下所示。

```
1 struct student                    //声明一个名为student的结构体类型
2 {
3   int ID;                         //学号
4   char name[20];                  //姓名
5   char sex;                       //性别
6   float score;                     //成绩
7 };                                //声明结束，注意此处不可省略分号
```

struct是声明结构体类型时必须使用的关键字，不能省略，它声明了这是一个“结构体类型”。student是该结构体的名称，该名称是用户自行定义的。花括号内的ID、name、sex及score是该结构体中的成员。

结构体被声明后，仅仅只是一个模型，系统并不为它分配内存空间，需要在声明之后定义结构体变量或结构体数组，系统才会在内存中给结构体变量或结构体数组分配内存空间，使之可以被使用。例如下面的代码。

```
1 struct student a,b;               //定义了两个结构体变量a和b
2 struct student stu[3];            //定义了结构体数组stu[3]
```

也可以声明结构体类型后直接定义结构体变量并对其初始化，例如将表9.1所示的内容存储到结构体并输出的代码如下所示。

表9.1

ID	name	sex	score
10001	琪儿	M	95
10002	小光	F	92

```
 1 //结构体变量实例
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 struct student
 6 {
 7   int ID;
 8   char name[20];
 9   char sex;
10   float score;
11 };
12
13 int main()
14 {
15   struct student a= {10001,{"琪儿",'M',95};        //定义结构体变量并赋值
16   struct student b= {10002,"小光",'F',92};        //不能改变赋值顺序
17   struct student c;                                 //结构体c未赋初始值
18   scanf("%d%s%c%f",&c.ID,c.name,&c.sex,&c.score);//输入结构体c的各成员
19   printf("%d %s %c %f\n",a.ID,a.name,a.sex,a.score);
20   printf("%d %s %c %f\n",b.ID,b.name,b.sex,b.score);
21   printf("%d %s %c %f\n",c.ID,c.name,c.sex,c.score);
22   return 0;
23 }
```



语句scanf(“%d%s%c%f”,&c.ID,c.name,&c.sex,&c.score)中，c.name的前面没有加取地址符“&”，因为c.name本身就是数组的首地址。

可以看出，结构体内的成员变量可以像普通变量一样进行各种操作，例如：

a.score=b.score;

sum=a.score+b.score;

a.score++;

应该逐个操作结构体内的各变量，不能将各变量作为整体一次操作完。例如输入/输出写成cin>>a或者cout<<a都是错误的，应改为:

cin>>a.ID>>a.name>>a.sex>>a.score;

cout<<a.ID<<a.name<<a.sex<<a.score;

使用结构体数组的实例如下所示。

```
1 //结构体数组实例
2 #include <bits/stdc++.h>
3 using namespace std;
4
```

```
 5 struct student
 6 {
 7   int ID;
 8   char name[20];
 9   char sex;
10   float score;
11 };
12
13 struct student s[3]= {{10001,"琪儿",'M',95},
14                       {10002,"小光",'F',92}};  //定义结构体数组并对前
两个变量赋值
15 int main()
16 {
17   scanf("%d%s%c%f",&s[2].ID,s[2].name,&s[2].sex,&s[2].score);
18   for (int i=0; i<3; i++)
19     printf("%d %s %c
%f\n",s[i].ID,s[i].name,s[i].sex,s[i].score);
20   return 0;
21 }
```

9.2 结构体与指针



扫码看视频

可以定义结构体指针变量指向结构体变量，例程如下所示。

```
 1 //结构体指针变量指向结构体变量
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 struct student
 6 {
 7   int num;
 8   char name[20],sex;
 9   float score;
10 };
```

```
11
12 int main()
13 {
14   struct student stu;    //定义结构体变量stu
15   struct student *p;     //*p必须和stu一样，均为struct student类型
16   p=&stu;                //结构体指针p指向结构体stu
17   stu.num=10001;
18   strcpy(stu.name,"Neo");//无法用"="对字符数组赋值，故用strcpy()
19   stu.sex='M';
20   stu.score=90.5;
21   cout<<stu.num<<stu.name<<stu.sex<<stu.score<<endl;     //输出方
式1
22   cout<<(*p).num<<(*p).name<<(*p).sex<<(*p).score<<endl;//输出方
式2
23   return 0;
24 }
```

在C++语言中，为了使用方便和直观，可以把(*p).num改为p->num，同理，可以把(*p).name改为p->name……

下面是一个指向结构体数组的指针变量实例。

```
 1 //指向结构体数组的指针变量实例
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 struct student
 6 {
 7   int num;
 8   char name[20],sex;
 9   float score;
10 };
11 struct student stu[2]={{10001,"琪儿",'M',95},      //未结束为逗号
12                       {10002,"小光",'F',92}};      //结束为分号
13
14 int main()
15 {
16   struct student *p;
17   for (p=stu;p<stu+2;p++)               //p指针依次指向下一个结构体
元素
18     cout<<p->num<<p->name<<p->sex<<p->score<<endl;
19   return 0;
20 }
```

9.3　课后练习



扫码看视频

9.3.1　选举（kind）

【题目描述】

有*N*个候选人参加选举，请用结构体数组编程统计每个人的得票数，即每次输入一个得票的候选人的名字，要求最后输出各候选人的得票结果。

【输入格式】

输入第一行为一个整数*N*（*N*不超过100），表示候选人数。随后*N*行为各候选人的姓名，之后是一个整数*M*（*M*不超过100），表示总投票数。随后为*M*行，每行为一个姓名，表示该候选人获得一张选票，如果选票上的姓名与候选人姓名不匹配，则为废票。

【输出格式】

输出每个候选人的姓名和得票数，中间以一个空格间隔，每个候选人占一行，按输入数据的初始顺序排列。

【输入样例】

```
3
Mike
John
Smith
5
Mike
Mik
John
John
Smith
```

【输出样例】

```
Mike 1
```

John 2

Smith 1

9.3.2　统计成绩（score）

【题目描述】

输入10个学生的数据，每个学生的数据包括学号、姓名及3门课的成绩，要求打印出3门课总平均成绩和最高分的学生数据（学号、姓名及平均分数）。试用结构体数组完成。

【输入格式】

输入10行数据，分别为每个学生的学号（整数）、姓名（不超过8个字符）及3门课的成绩，以一个空格间隔。

【输出格式】

输出第一行为3门课总平均成绩，第二行为总平均成绩最高分（保证没有并列的最高成绩）的学生数据（包括学号、姓名及平均分数），以一个空格间隔，输出的浮点数保留小数点后两位（不考虑四舍五入）。

【输入样例】

1 A 100 100 100

2 B 80 80 80

3 C 80 80 80

4 D 80 80 80

5 E 80 80 80

6 F 80 80 80

7 G 80 80 80

8 H 80 80 80

9 I 80 80 70

10 J 80 80 80

【输出样例】

81.67

1 A 100.00

9.3.3 生日（birthday）

【题目描述】

试用结构体数组编程统计每个学员的生日，并按照年龄从大到小的顺序排序。

【输入格式】

第一行为学员总人数*n*，随后*n*行分别是每个学员的姓名与出生年、月、日，以一个空格间隔。

【输出格式】

输出*n*行，即*n*个按照年龄从大到小排序的学员的姓名（如果有两个学员生日相同，输入靠后的学员先输出）。

【输入样例】

```
3
Yangchu 1992 4 23
Qiujingya 1993 10 13
Luowen 1991 8 1
```

【输出样例】

Luowen

Yangchu

Qiujingya

第10章　位运算与进制

计算机程序中的所有数据在计算机内存中都是以二进制数的形式储存的。由于位运算直接对内存数据进行操作，不需要转成十进制数，因此处理速度非常快。

10.1　位运算



扫码看视频

二进制用0和1两个数码来表示数，是计算机技术中广泛采用的一种数制。二进制的基数为2，进位规则是“逢二进一”，借位规则是“借一当二”，由17～18世纪德国数理哲学大师莱布尼茨（Leibniz）发现。

（1）二进制加法运算有以下4种情况。

0＋0＝0

0＋1＝1

1＋0＝1

1＋1＝10　　//进位为1

例如计算$(1101)_2$＋$(1011)_2$ 的值。

解：
$$\begin{array}{r} 1101 \\ +\ 1011 \\ \hline 11000 \end{array}$$

（2）二进制减法运算有以下4种情况。

0－0＝0

1－0＝1

1－1＝0

10－1＝1

原码：在二进制表示的数前面增加了一位符号位（最高位为符号位），0表示正数，1表示负数，其余位表示数的大小。例如$(0010)_2$＝2，$(1010)_2$＝-2。

反码：原码在计算机中使用不方便，因此引入了反码，正数的反码与原码相同，负数的反码是将原码中除了第一位符号位不变外，其他每一位数取反。例如127＝01111111，而-127＝10000000。

补码：多数计算机的整数采用补码表示法，因为反码中0000＝0、1111＝-0＝0，有两个数表示同一个数，所以在正数和负数的计算中会

出错。正数的补码与原码相同，负数的补码最高位为1，其余位在反码基础上加1。例如-5＝11111010（反码），补码为11111011。

表10.1所示是一些整数的原码、反码及补码的表示方法。

表10.1

数据（8位）	原码	反码	补码
+7	00000111	00000111	00000111
-7	10000111	11111000	11111001
+0	00000000	00000000	00000000
-0	10000000	11111111	00000000

补码的加、减法运算十分方便，它不必判断数的正负，只要符号位参加运算便能自动得到正确的结果。假设计算机字长为8位，下列例子说明补码的加法运算。

十进制	二进制
25	00011001
+ 32	+ 00100000
57	00111001

十进制	二进制
32	00100000
+（-25）	+ 11100111
7	00000111
	↓
	1 自动丢弃

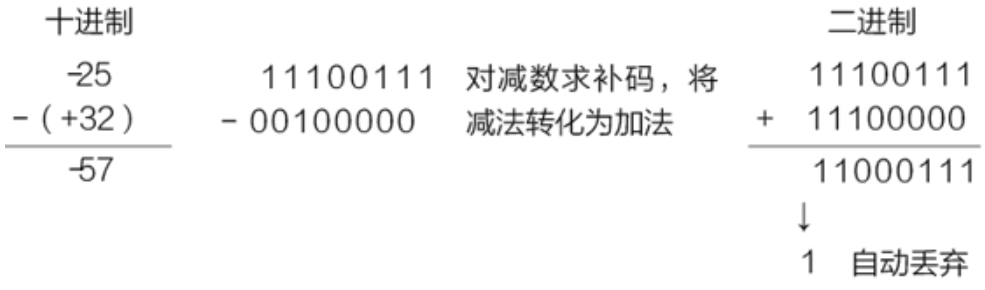
十进制	二进制
25	00011001
+（-32）	+ 11100000
-7	11111001

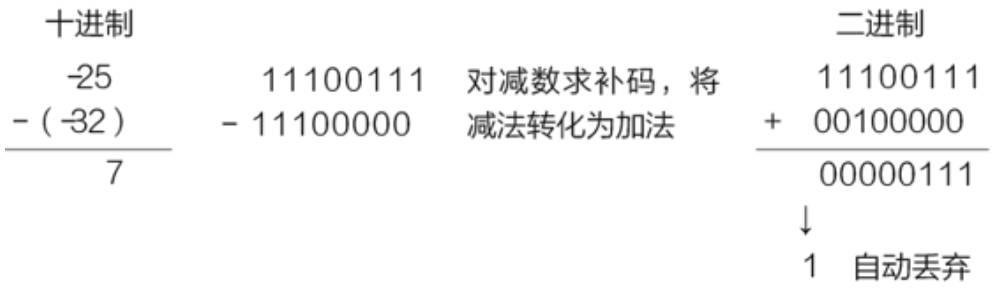
十进制	二进制
32	00100000
+（-25）	+ 11100111
7	00000111
	↓
	1 自动丢弃

补码的减法运算与加法运算同理，下面以几个例子来说明。

十进制			二进制
25	00011001	对减数求补码，将	00011001
- 32	- 00100000	减法转化为加法	+ 11100000
-7			11111001

十进制			二进制
25	00011001	对减数求补码，将	00011001
- 32	- 00100000	减法转化为加法	+ 11100000
-7			11111001





C++语言提供表10.2所列出的位运算符。

表10.2

位运算符	含义	位运算符	含义
&	与（and）	~	取反（not）
|	或（or）	<<	左移（shl）
^	异或（xor）	>>	右移（shr）

位运算符的优先级：not＞shl，shr＞and＞xor＞or。

与运算符“&”：参加运算的两个运算量，如果两个相应的位都为1，则该位的结果值为1，否则为0。即0&0＝0，0&1＝0，1&0＝0，1&1

＝1。

与运算通常用于二进制数取位操作，例如一个数 and 1的结果就是取二进制数的最末位。这可以用来判断一个整数的奇偶性，二进制数的最末位为0表示该数为偶数，最末位为1表示该数为奇数。

```
 1 //判断奇偶性
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int x;
 8   cin>>x;
 9   if ((x&1)==0)                //注意位运算的优先级很低，所以必须加圆括号
10     cout<<x<<"是偶数"<<endl;
11   else
12     cout<<x<<"是奇数"<<endl;
13   return 0;
14 }
```

如果*A*、*B*为整数，实际上可看成长度为32的二进制数各个位做上述操作。例如当*A*＝10（二进制数为1010）、*B*＝12（二进制数为1100）时，*A* & *B*＝8（1010 & 1100＝1000）。

```
1 //两个整数与运算
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 int main()
6 {
7   int a=10,b=12;
8   cout<<(a&b)<<endl;      //位运算的优先级很低，必须加圆括号，输出结果为8
9   return 0;
10 }
```

或运算符“｜”：两个相应位中只要有一个为1，该位的结果值就为1。即0|0＝0，0|1＝1，1|0＝1，1|1＝1。

如果*A*、*B*为整数，实际上可看成长度为32的二进制数各个位做上述操作。例如当*A*＝10（二进制数为1010）、*B*＝12（二进制数为1100）时，*A*|*B*＝14 （1010|1100＝1110）。

```
 1 //两个整数或运算
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a=10,b=12;
 8   cout<<(a|b)<<endl;    //位运算的优先级很低，必须加圆括号，输出结果为14
 9   return 0;
10 }
```

异或运算符“^”：参加运算的两个相应位同号则结果为0，异号则结果为1。xor的直观意思就是“是不是不一样”。即0^0＝0，0^1＝1，1^0＝1，1^1＝0。

如果*A*、*B*为整数，实际上可看成长度为32的二进制数各个位做上述操作。例如当*A*＝10（二进制数为1010）、*B*＝12（二进制数为1100）时，*A*^*B*＝6 （1010^1100＝0110）。

```
 1 //两个整数异或运算
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a=10,b=12;
 8   cout<<(a^b)<<endl;              //位运算的优先级很低，必须加圆括号，
输出结果为6
 9   return 0;
10 }
```

取反运算符“～”：用来对一个二进制数按位取反，即将0变为1、1变为0。例如对二进制数 1 取反即～ 1 ＝ 0 。

例如一个整数*a*为10，则～*a*=-11 （～0…01010＝1…10101，这里符号也取反（首位），因此变为负数）。

```
 1 //取反运算符
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a=10;
 8   cout<<~a<<endl;                                    //输出结
果-11
 9   return 0;
10 }
```

使用取反运算符要注意整数类型有没有符号，有符号的整数与无符号的整数取反的结果是不一样的。对于无符号的数，取反后的效果就是把这个数在数轴上的位置“对称翻折到另一边”，因为无符号的数是用0x00000000～0xFFFFFFFF依次表示的(0x代表十六进制)，如图10.1所示。



图10.1

而对于有符号的数，取反后最高位的变化导致了正负颠倒，又因为负数储存使用补码，所以效果就变为-*a*-1。这与上下界没有任何关系。

```
 1 //无符号数取反
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   unsigned short a=100;
 8   a=~a;
 9   cout<<a<<endl;    //输出65435，即该类型的最大值减100，而不是减101
10   return 0;
11 }
```

左移运算符“<<”：*A*<<*B*，表示*A*的所有二进制位整体向左移动*B*位，后面*B*位用0补充。对无符号数左移一位的操作如图10.2所示。

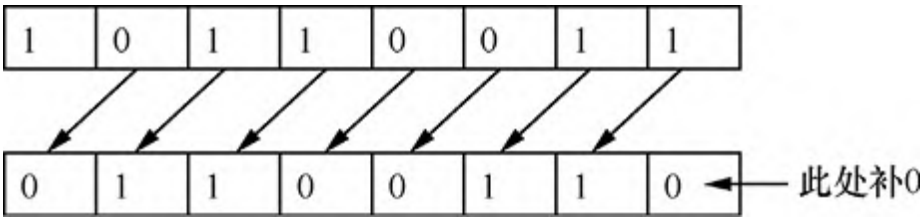


图10.2

例如100的二进制数为1100100，则100<<2＝110010000＝400。可以看出，*a*<<*b*的值实际上就是*a*乘以2的*b*次方，因为在二进制数后添一个0就相当于该数乘以2。

通常认为a << 1比a×2运算更快，因为前者是更底层一些的操作，所以程序中乘以2的操作请尽量用左移一位来代替。

```
 1 //左移运算符
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a=10;
 8   cout<<(a<<2)<<endl;                    //左移两位，即输出10*2*2=40
 9   return 0;
10 }
```

定义一些常量可能会用到左移运算符。可以方便地用(1 << 15)－1来表示32 767。很多算法和数据结构要求数据规模必须是2的幂，此时可以用左移运算符来定义Max_N等常量。

```
 1 //左移运算符
 2 #include <bits/stdc++.h >
 3 using namespace std;
 4
 5 int main()
 6 {
 7   short a;
 8   unsigned short b;
 9   a=(1<<16)-1;                           //short类型长度为16位
10   b=(1<<16)-1;
11   cout<<a<<" "<<b<<endl;                 //输出-1 65535
12   cout<<(1<<15)-1<<endl;                 //输出32767
13   return 0;
14 }
```

右移运算符“>>”：将一个数的各二进制位右移，右移时需注意符号位问题，务必确保是对非负整数进行运算，否则会出错。例如当A＝-1时，对于任何位移运算$A>>B$，结果都是-1。图10.3所示是对一个负数（有符号数）右移一位的操作。

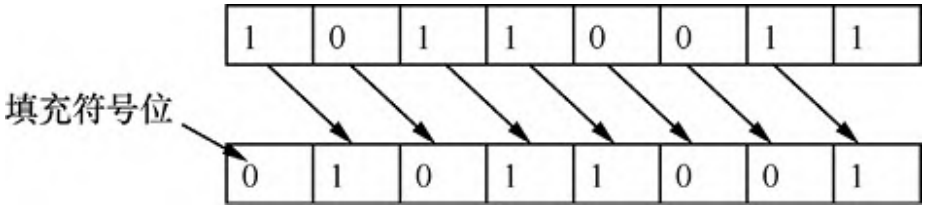


图10.3

对于无符号数右移一位的操作如图10.4所示。

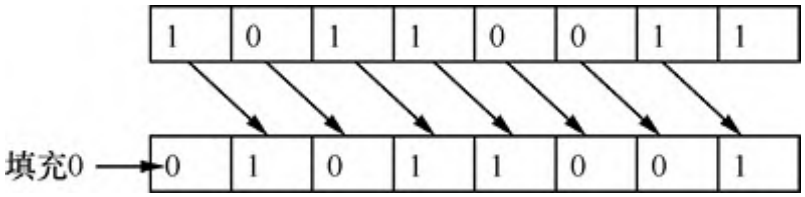


图10.4

和左移相似，*a* >> *b*表示二进制数右移*b*位（去掉末*b*位），相当于*a*除以2的*b*次方（取整）。实际编程经常用>>1来代替除以2，例如二分查找、堆的插入操作等，这可以使程序运行效率大大提高。例如最大公约数的二进制算法用除以2操作来代替运算速度缓慢的模运算（%），运行效率可以提高60%。

```
 1 //右移运算符
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a=10;
 8   cout<<(a>>2)<<endl;                    //右移两位，即10/2/2=2
 9   return 0;
10 }
```

10.1.1 整数幂（power）

【题目描述】

判断整数N是不是2的整数幂，例如$8=2^3$，8是2的整数幂，而9不是2的整数幂。

【输入格式】

第一行为一个整数T（$1\leqslant T\leqslant 1\ 000$），表示有$T$组数据。

随后为T行，每行一个正整数N（N在整数范围内）。

【输出格式】

如果N是2的整数幂，则输出“Yes”，否则输出“No”。

【输入样例】

1

8

【输出样例】

Yes

【算法分析】

最简单的方法是用这个数除以2得到商和余数，再用商除以2又得到商和余数，重复该操作直到商为0。当商为0余数也为0时，该数就是2的整数幂。

将2的幂次方写成二进制数形式后，很容易就会发现有一个特点：二进制数中只有一个1，并且1后面跟了*n*个0。例如8转化成二进制数为00001000，64转化成二进制数为01000000。

如果将这个数减去1会发现，仅有的1会变为0，而原来的*n*个0会变为1，例如64－1的二进制数为00111111，而01000000 & 00111111＝0。

实际上，x&(x－1)常用于消除二进制数最后出现的1，其余保持不变。

参考程序如下所示。

```
 1 //整数幂——位运算法
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int n,t;
 8   scanf("%d",&t);
 9   while (t--)
10   {
11     scanf("%d",&n);
12     if ((n&(n-1))==0)                                    //"=="比"&"优先级
高
13       printf("Yes\n");
14     else
15       printf("No\n");
16   }
17   return 0;
18 }
```

10.1.2　二进制半整数（bin）

【题目描述】

一个数n如果能表示成2^i+2^j，那么它就是二进制半整数。试判断一个数是不是二进制半整数。

【输入格式】

第一行输入t（$1 \leqslant t \leqslant 1\,000$），表示数据组数。

接下来t行，每行一个数n（$1 \leqslant n \leqslant 1\,000\,000\,000$）。

【输出格式】

输出t行，每行以“yes”或“no”表示是不是二进制半整数。

【输入样例】

4

4

7

5

6

【输出样例】

yes

no

yes

yes

10.2　进制转换



扫码看视频

除了十进制和二进制以外，还有其他进制，例如八进制和十六进制。

八进制采用0、1、2、3、4、5、6、7共8个数码，逢8进位。八进制的数较二进制的数书写方便，常应用在计算机的计算中。例如十进

制数32表示成八进制数就是40，八进制数32表示成十进制数就是$3\times8^1+2\times8^0=26$。

十六进制采用0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F共16个数码。其中A、B、C、D、E、F分别对应十进制数10、11、12、13、14、15。

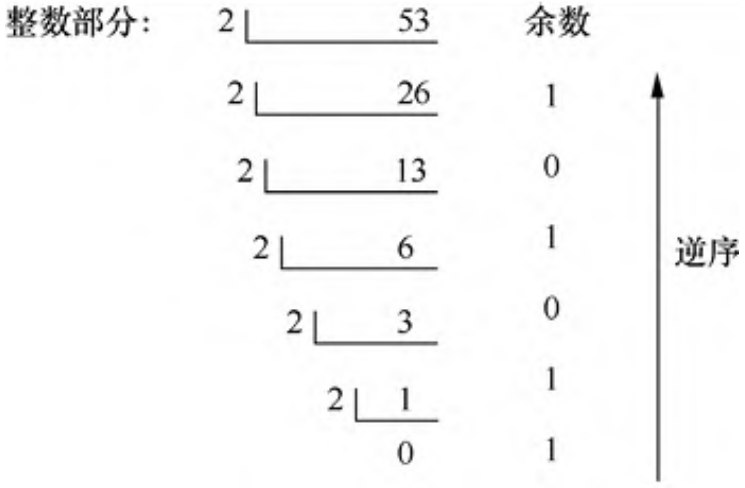
各进制数之间的转换示例如下。

二进制数转十进制数：$10110=1\times2^4+0\times2^3+1\times2^2+1\times2^1+0\times2^0=22$。

八进制数转十进制数：$125=1\times8^2+2\times8^1+5\times8^0=85$。

十六进制数转十进制数：$3A8=3\times16^2+A\times16^1+8\times16^0=936$。

十进制数转二进制数的方法如下（除2取余、逆序排列法）。



即53＝110101。

小数部分：0.312 5 × 2 ＝ 0.625 0　　整数为0

0.625 0 × 2 ＝1.25　　整数为1

0.25 ×2 ＝ 0.50　　整数为0

0.50 × 2 ＝1.0　　整数为1

即 0.312 5＝0.0101。

二进制数转八进制数： 101 011 110 ＝536。

八进制数转二进制数：357 ＝ 11 101 111 。

十六进制数转二进制数：5AB ＝ 101 1010 1011 。

二进制数转十六进制数： 1 1001 0100 1011 ＝194B。



前文讲到用输入/输出流类库中的dec(对应十进制)、hex(对应十六进制)、oct(对应八进制)的方法处理数制转换问题。

接下来介绍数制转换的通用算法。

将十进制数转换为相应的二进制数，除了模拟“除2取余、逆序排列法”的计算过程外，更快捷的方法是位运算，参考程序如下所示。

```
 1 //输入十进制数以二进制数显示
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int m,number,s[32];
 8   cin>>number;
 9   for (int i=1; i<=32; i++)          //32位的编译器
10     s[i-1]=number>>(i-1)&1;          //将每一位移到最右端与1进行与运算
11   for (int i=31; i>=0; i--)
12     cout<<s[i]<<' ';
13   return 0;
14 }
```

输入二进制数以十进制数显示的参考程序如下所示。

```
 1 //输入二进制数以十进制数显示
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int Show10(char *c)
 6 {
 7   int num=0;
 8   for (int i=0; i<=31; i++)
 9     num=(num<<1)+c[i];
10   return num;
11 }
12
13 int main()
14 {
15   int x;
16   char c[32];
17   for (int i=0; i<=31; i++)            //以字符形式输入32位的二进制数
18     cin>>c[i];
19   for (int i=0; i<=31; i++)            //字符转换为数字
20     c[i]=c[i]-'0';
21   cout<<Show10(c)<<endl;
22   return 0;
23 }
```

十进制数转换为*N*进制数的参考程序如下所示。

```
 1 //十进制数转换为N进制数 —— 采用除N反向取余的方式
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 string Fun(int x,int n)
 6 {
 7   const string a="0123456789ABCDEF";
 8   string s="";
 9   if (x==0)                                    //注意特殊数据0的处理
10     return "0";
11   for (; x>0; x/=n)
12     s=a[x%n]+s;                                //后取的余数放在前面
13   return s;
14 }
15
16 int main()
17 {
18   int x,n;
19   cin>>x>>n;
20   cout<<Fun(x,n)<<endl;
21   return 0;
22 }
```

*N*进制数转换为十进制数的参考程序如下所示。

```
 1 //N进制数转换为十进制数
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int Fun(int n,string s)
 6 {
 7   int i,t=0;
 8   for (i=0; i<=s.size(); i++)
 9   {
10     if (s[i]>='0' && s[i]<='9')
11       t=t*n+s[i]-48;
12     else if (s[i]>='A' && s[i]<='F')
13       t=t*n+s[i]-55;
14     else if (s[i]>='a' && s[i]<='f')
15       t=t*n+s[i]-87;
16   }
17   return t;
18 }
19
```

```
20 int main()
21 {
22   int n;
23   string str;
24   cin>>n>>str;
25   cout<<Fun(n,str)<<endl;
26   return 0;
27 }
```

10.2.1　N进制数加法（add）

【题目描述】

N（*N*＜37）进制数加法运算问题，即从键盘输入一个小于37的正整数*N*，再输入符合要求的两个*N*进制数，求两者之和，输出结果仍为*N*进制数。



最容易想到的算法是先将这两个数转换为十进制数，然后进行加法运算，再将十进制数转换为*N*进制数。但该算法要进行两次数制的转换，效率太低。有没有更好的方法呢？

参考程序如下所示。

```
1 //N进制数加法
2 #include <bits/stdc++.h>
3 using namespace std;
```

```
 4
 5 int main()
 6 {
 7   string a,b,w;
 8   int x[100],y[100],i,k,N;
 9   cin>>N;
10   for (i=0; i<=N-1; i++)
11   {
12     if (i<10)
13       w=w+char(i+48);          //数字的处理
14     else
15       w=w+char(55+i);          //字母的处理
16   }
17   cin>>a>>b;
18   while (a.length()<b.length())
19     a='0'+a;
20   while (b.length()<a.length())
21     b='0'+b;
22   a='0'+a;                     //前面多加一位，用于进位
23   b='0'+b;
24   for (i=a.length()-1; i>=0; i--)
25   {
26     x[i]=w.find(a[i],0);       //查找a[i]在w中的位置获得真实的数字
27     y[i]=w.find(b[i],0);       //转换到x、y数组中准备相加
28   }
29   for (i=a.length()-1; i>=0; i--) //进位加法
30   {
31     x[i]=x[i]+y[i];
32     if (x[i]>=N)
33     {
34       k=i;
35       while (x[k]>=N)
36       {
37         x[k]=x[k]-N;
38         x[k-1]++;
39         k--;
40       }
41     }
42   }
43   for (i=a.length()-1;i>=0;i--)
44     a[i]=w[x[i]];              //转换为N进制数
45   while (a[0]=='0')            //此处没有考虑输入为0的特殊情况
46      a.erase(0,1);             //删除前导0
47   cout<<a<<endl;
48   return 0;
49 }
```



如果是一些特殊的进制数之间进行转换，可以设计特殊的算法。例如二进制数与八进制数、二进制数与十六进制数等有简单幂次关系的进制数之间相互转换，就可以简单用1位对应于3位和1位对应于4位的方法来设计算法。而像八进制数与十六进制数之间进行转换，则需借助十进制数作为桥梁来间接地转换。

10.2.2　二进制数分类（classify）

【题目描述】

若将一个正整数转换为二进制数，则在此类二进制数中，我们将数字1的个数多于数字0的个数的这类二进制数称为A类数，否则就称其为B类数。例如：

$(13)_{10}=(1101)_2$，其中1的个数为3，0的个数为1，则称此数为A类数；

$(10)_{10}=(1010)_2$，其中1的个数为2，0的个数也为2，则称此数为B类数；

$(24)_{10}=(11000)_2$，其中1的个数为2，0的个数为3，则称此数为B类数。

程序要求：求出a～b的全部A、B两类数的个数。

【输入格式】

输入两个整数，即a和b（$1 \leqslant a$，$b \leqslant 1\,000$）。

【输出格式】

输出两个整数，分别是A类数和B类数的个数，中间以一个空格间隔。

【输入样例】

1 1000

【输出样例】

538 462

10.2.3 确定进制（num）

【题目描述】

6×9＝42 对十进制来说是错误的，但是对十三进制来说却是正确的。即$6_{(13)} \times 9_{(13)} = 42_{(13)}$，而$42_{(13)} = 4 \times 13 + 2 \times 1 =$

$54_{(10)}$。

试编程输入3个整数p、q、r，然后确定进制B（$2 \leqslant B \leqslant 16$），使得$p \times q = r$。如果$B$有很多选择，则输出最小的那个进制；如果没有合适的进制，则输出0。

【输入格式】

输入3个整数p，q，r（$1 \leqslant p,q,r \leqslant 1\,000\,000$）。

【输出格式】

输出一个整数，即令$p \times q = r$成立的最小的B。

【输入样例】

11 11 121

【输出样例】

3

10.2.4 *K*进制数转*L*进制数

【题目描述】

输入K进制的正整数N，将之转换为L进制数后输出。

【输入格式】

输入多组数据，每组数据有3个正整数K，N，L（$N<1\,000\,000$；$L, K \leqslant 16$）。

【输出格式】

每行输出一个数，即N的L进制数。

【输入样例】

8 10 2

10 10 16

【输出样例】

1000

A

第11章　STL编程

标准模板库（Standard Template Library，STL）从广义上讲分为算法（Algorithm）、容器（Container）及迭代器（Iterator）3类，包含诸多常用的基本数据结构和基本算法。

标准C++语言中，STL被组织为下面的13个头文件：<algorithm>、<deque>、<functional>、<iterator>、<vector>、<list>、<map>、<memory>、<numeric>、<queue>、<set>、<stack>及<utility>。

11.1　sort排序算法



扫码看视频

STL里的sort排序算法在头文件<algorithm>中声明，采用的是成熟的“快速排序算法”，可以保证很好的平均性能，其时间复杂度为$n\log(n)$，比标准C语言的qsort要好。

对无序整型数组排序的参考程序如下所示。

```
 1 //sort对无序整型数组排序
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 void Print(int a[],int n)
 6 {
 7   for (int i=0; i<n; i++)
 8     cout<<a[i]<<' ';
 9   cout<<endl;
10 }
11
12 int main()
13 {
14   int a[]= {-1,9,-34,100,45,2,98,32};                    //无序整型数
组元素
15   int len=sizeof(a)/sizeof(int);
16   sort(a,a+len);                                          //由小到大排
序
17   Print(a,len);
18   sort(a,a+len,greater<int>());                           //由大到小排
序
19   Print(a,len);
20   return 0;
21 }
```

对字符排序也一样简单，参考程序如下所示。

```
 1 //sort对字符排序
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   char a[11]="asdfghjklk";                    //无序字符元素
 8   for (int i=0; i<10; i++)
 9     cout<<a[i];
10   cout<<endl;
11   sort(a,a+10,greater<char>());               //按ASCII值由大到小排
序
12   for (int i=0; i<10; i++)
13     cout<<a[i];
13   return 0;
14 }
```

对结构体排序的参考程序如下所示。可以看出，根据题目要求可以自定义比较函数。

```
//sort对结构体排序
#include <bits/stdc++.h>
using namespace std;

struct Node
{
  int x,y;
} p[1001];

int Cmp(Node a,Node b)
{
  if (a.x != b.x)
    return a.x < b.x;            //如果a.x不等于b.x，就按x从小到大排序
  return a.y < b.y;              //如果x相等则按y从小到大排序
}

int main()
{
  int n;
  scanf("%d",&n);
  for (int i = 1; i <= n; i++)
    scanf("%d%d",&p[i].x,&p[i].y);
  sort(p+1,p+n+1,Cmp);           //排序，比较函数为Cmp()
  for (int i = 1; i <=n; i++)
    printf("%d %d\n",p[i].x,p[i].y);
  return 0;
}
```

11.1.1 单词排序(WordSort)

【题目描述】

输入一行单词，相邻单词之间由一个或多个空格间隔，请按照字典序输出这些单词，要求重复的单词只输出一次（区分大小写）。

【输入格式】

第一行为一个整数n，表示有n（$1 \leqslant n \leqslant 100$）个单词。随后一行为$n$个单词，每个单词长度不超过50，单词之间用至少一个空格间隔。数据不含除字母、空格外的其他字符。

【输出格式】

按字典序输出这些单词，重复的单词只输出一次。

【输入样例】

```
3

Keep on going
```

【输出样例】

```
Keep

going

on
```

11.1.2　志愿者选拔（voluntary）

【题目描述】

学院选拔志愿者，面试分数线根据计划录取人数的150%划定，即如果计划录取m名志愿者，则面试分数线为排名第m×150%（向下取整）名的选手的分数，而最终进入面试的选手为笔试成绩不低于面试分数线的所有选手。

请编写程序划定面试分数线，并输出所有进入面试的选手的报名号和笔试成绩。

【输入格式】

第一行为两个整数 n和m （$5 \leqslant n \leqslant 5000$，$3 \leqslant m \leqslant n$），其中$n$表示报名参加笔试的选手总数，$m$表示计划录取的志愿者人数。输入数据保证$m$×150%向下取整后小于等于$n$。

第二行到第n+1行，每行两个整数，分别是选手的报名号k（$1000 \leqslant k \leqslant 9999$）和该选手的笔试成绩$s$（$1 \leqslant s \leqslant 100$）。输入数据保证选手的报名号各不相同。

【输出格式】

第一行为两个整数，分别表示面试分数线和进入面试的选手的实际人数。

从第二行开始，每行两个整数，分别表示进入面试的选手的报名号和笔试成绩，按照笔试成绩从高到低输出，如果成绩相同则按报名号由小到大的顺序输出。

【输入样例】

6 3

1000 90

3239 88

2390 95

7231 84

1005 95

1001 88

【输出样例】

88 5

1005 95

2390 95

1000 90

1001 88

3239 88

【样例说明】

m×150%＝3×150%＝4.5，向下取整后为4。保证4个人进入面试的分数线为88，但因为88有重分，所以所有成绩大于等于88的选手都可以进入面试，故进入面试的选手的实际人数为5。

11.1.3 奖学金（scholar）

【题目描述】

学校打算为学习成绩优秀的前5名学生发奖学金。每个学生都有3门课的成绩：语文、数学及英语。先按总分从高到低排序，如果两个学生总分相同，再按语文成绩从高到低排序；如果两个学生总分和语文成绩都相同，那么规定学号小的学生排在前面。这样每个学生的排序是唯一确定的，试按排名顺序输出前5名学生的学号和总分。



使用“内容市场”
APP扫描看视频

【输入格式】

第一行为一个正整数n（$6 \leqslant n \leqslant 300$），表示该校参加评选的学生人数。

第二行到第$n+1$行，每行有3个以空格间隔的数字，每个数字都为0～100。第j行的3个数字依次表示学号为$j-1$的学生的语文、数学及英语的成绩。每个学生的学号按照输入顺序编号为$1 \sim n$（恰好是输入数据的行号减1）。

【输出格式】

输出共有5行，每行两个正整数（以空格间隔），依次表示前5名学生的学号和总分。

【输入样例】

6

90 67 80

87 66 91

78 89 91

88 99 77

67 89 64

78 89 98

【输出样例】

6 265

4 264

3 258

2 244

1 237

11.1.4 导弹拦截（missile）

【题目描述】

有一种新的导弹拦截系统，凡是与它的距离不超过其工作半径的导弹都能够被它成功拦截。当工作半径为0时，则能够拦截与它位置恰好相同的导弹。但每套导弹拦截系统每天只能设定一次工作半径，而当天的使用代价，就是所有系统工作半径的平方和。

某天，雷达捕捉到敌国的导弹来袭。由于该系统尚处于试验阶段，因此只有两套系统投入工作。如果现在的要求是拦截所有的导弹，请计算这一天的最小使用代价。

【输入格式】

第一行包含4个整数x_1、y_1、x_2、y_2，每两个整数之间以一个空格间隔，表示这两套导弹拦截系统的坐标分别为(x_1, y_1)、(x_2, y_2)。

第二行包含一个整数N，表示有N颗导弹。接下来为N行，每行有两个整数x、y，中间以一个空格间隔，表示一颗导弹的坐标(x, y)，不同导弹的坐标可能相同。

【输出格式】

输出只有一行，包含一个整数，即当天的最小使用代价。

【算法提示】

两个点(x_1, y_1)、(x_2, y_2)之间距离的平方是$(x_1-x_2)^2+(y_1-y_2)^2$。

两套系统工作半径r_1、r_2的平方和是指r_1、r_2 分别取平方后再求和，即$r_1^2+r_2^2$。

【输入样例1】

```
0 0 10 0
2
-3 3
10 0
```

【输出样例1】

```
18
```

【样例说明1】

样例1中要拦截所有导弹，在满足最小使用代价的前提下，两套系统工作半径的平方分别为18和0。

【输入样例2】

0 0 6 0

5

-4 -2

-2 3

4 0

6 -2

9 1

【输出样例2】

30

【样例说明2】

样例2中的导弹拦截系统和导弹所在的位置如图11.1所示。要拦截所有导弹，在满足最小使用代价的前提下，两套系统工作半径的平方分别为20和10，即最小使用代价为30。

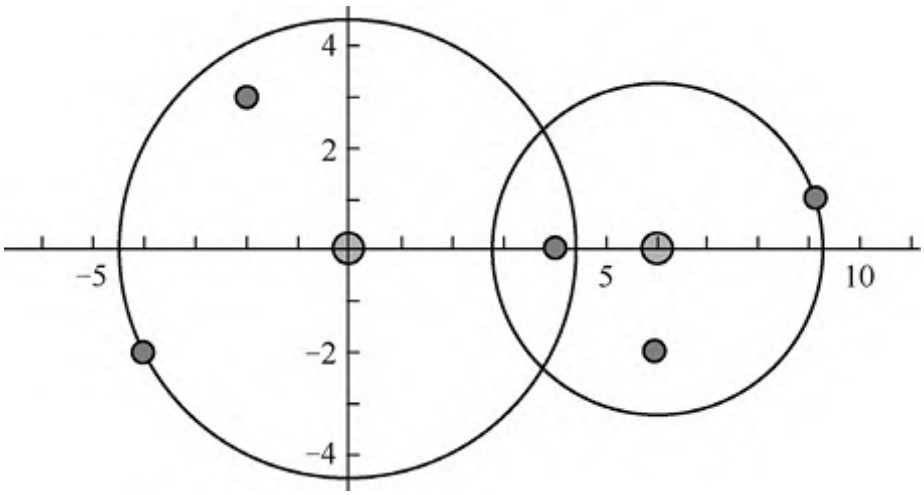


图11.1

【数据范围】

对于100%的数据，$1 \leqslant N \leqslant 100\,000$，且所有坐标分量的绝对值都不超过1 000。

【算法分析】

如图11.2所示，设导弹拦截系统为a和b，计算出所有导弹到a和b的距离，并按照到a的距离从大到小进行排序。假如选择某一个点例如k点到a的距离作为a的半径，那么k点之后的点都能被a击落，而k点之前的点只能由b击落，则b的半径即前 $k-1$个点到 b 的最大半径。

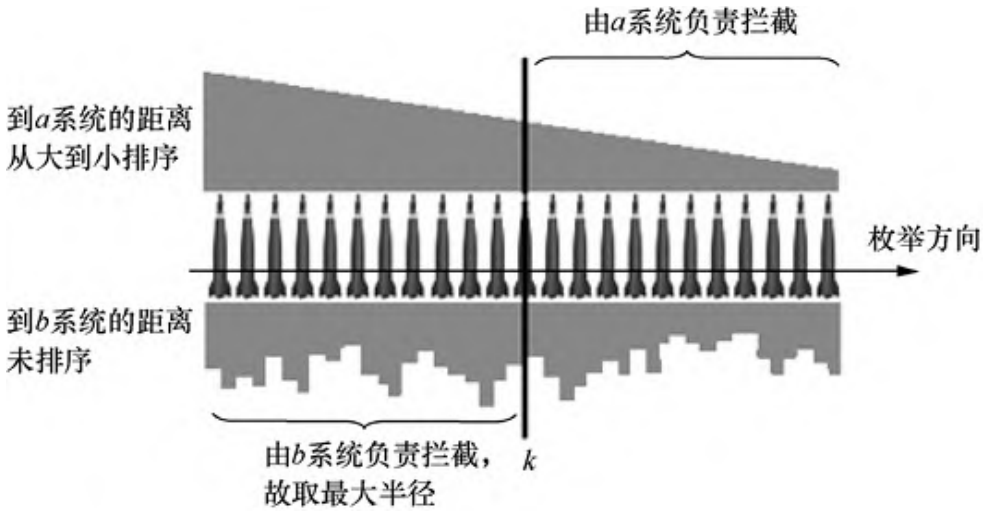


图11.2

11.2 lower_bound/upper_bound

lower_bound(起始地址first,结束地址last,要查找的数值val)：在first和last中的前闭后开区间进行二分查找，返回大于或等于val的第一个元素地址。如果区间内所有元素都小于val，则返回last的地址，且last的地址是越界的。

upper_bound(起始地址first,结束地址last,要查找的数值val)：在first和last中的前闭后开区间进行二分查找，返回大于val的第一个元素地址。如果val大于区间内所有元素，则返回last的地址，且last的地址是越界的。

特别注意：lower_bound/upper_bound二分查找的区间必须为有序序列，如图11.3所示。

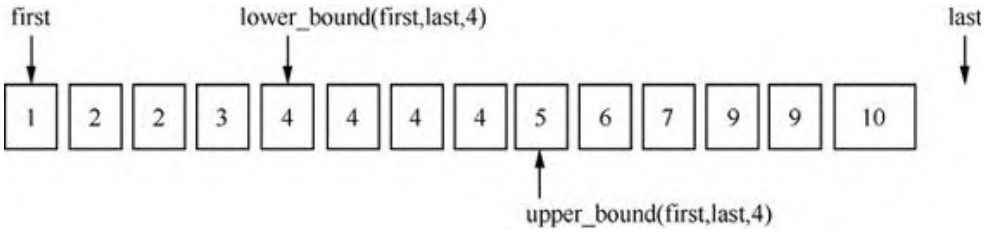


图11.3

升序数组使用lower_bound/upper_bound的示例程序如下所示。

```
 1 //升序数组使用lower_bound/upper_bound的示例
 2 #include<bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a[] = {1, 1, 2, 2, 3, 3, 3, 4, 4, 4};
 8   cout<< lower_bound(a, a+10, 0)-a <<endl;            //输出下标
0
 9   cout<< lower_bound(a, a+10, 1)-a <<endl;            //输出下标
0
10   cout<< lower_bound(a, a+10, 3)-a <<endl;            //输出下标
4
11   cout<< lower_bound(a, a+10, 4)-a <<endl;            //输出下标
7
12   cout<< lower_bound(a, a+10, 5)-a <<endl;            //输出下标
10
13
14   cout<< upper_bound(a, a+10, 0)-a <<endl;            //输出下标
0
15   cout<< upper_bound(a, a+10, 1)-a <<endl;            //输出下标
2
16   cout<< upper_bound(a, a+10, 3)-a <<endl;            //输出下标
7
17   cout<< upper_bound(a, a+10, 4)-a <<endl;            //输出下标
10
18   cout<< upper_bound(a, a+10, 5)-a <<endl;            //输出下标
10
19   return 0;
20 }
```



因为lower_bound和upper_bound返回的是数组元素的地址值，所以再减去数组首地址的值即该数组元素的下标。

降序数组直接使用lower_bound/upper_bound二分查找的结果是错误的，示例程序如下所示。

```
 1 //降序数组使用lower_bound/upper_bound的错误示例
 2 #include<bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a[] = {4, 4, 3, 3, 2, 2,  1, 1};                  //降序数组
 8   cout<< lower_bound(a, a+8, 4)-a <<endl;               //输出下标
8
 9   cout<< upper_bound(a, a+8, 4)-a <<endl;               //输出下标
8
10   cout<< lower_bound(a, a+8, 1)-a <<endl;               //输出下标
0
11   cout<< upper_bound(a, a+8, 1)-a <<endl;               //输出下标
0
12   cout<< lower_bound(a, a+8, 3)-a <<endl;               //输出下标
8
13   cout<< upper_bound(a, a+8, 3)-a <<endl;               //输出下标
8
14   return 0;
15 }
```



这是因为lower_bound/upper_bound默认二分查找的区间是升序序列。以查找数值4为例，第一步从中间开始，取中间值 a[(0+8)/2] = a[4] =2，比4小，于是继续向更大的值靠近，向哪边靠近呢，右边，因为它以为序列是升序的，这显然是错误的。

所以，在降序序列要注意以下两点。

（1）lower_bound的正确写法为lower_bound(first, last, val, greater<int>())，或类似于sort排序，使用自定义比较函数。若val在序列中，则返回val第一次出现的位置，否则返回第一个插入val不影响原序列顺序的位置。

（2）upper_bound的正确写法为upper_bound(first, last, val, greater<int>())，或类似于sort排序，使用自定义比较函数。若val在序列中，则返回第一个小于val的位置，否则返回第一个插入val不影响原序列顺序的位置。

参考程序如下所示。

```
 1 //降序数组使用lower_bound/upper_bound的正确示例
 2 #include<bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int a[] = {4, 4, 3, 3, 2, 2,  1, 1};
 8   cout<< lower_bound(a, a+8, 0,greater<int>())-a <<endl;  //输出
8
9   cout<< lower_bound(a, a+8, 4,greater<int>())-a <<endl;  //输出
0
```

```
10    cout<< lower_bound(a, a+8, 1,greater<int>())-a <<endl;   //输出
6
11    cout<< lower_bound(a, a+8, 3,greater<int>())-a <<endl;   //输出
2
12    cout<< lower_bound(a, a+8, 5,greater<int>())-a <<endl;   //输出
0
13
14    cout<< upper_bound(a, a+8, 0,greater<int>())-a <<endl;   //输出
8
15    cout<< upper_bound(a, a+8, 4,greater<int>())-a <<endl;   //输出
2
16    cout<< upper_bound(a, a+8, 1,greater<int>())-a <<endl;   //输出
8
17    cout<< upper_bound(a, a+8, 3,greater<int>())-a <<endl;   //输出
4
18    cout<< upper_bound(a, a+8, 5,greater<int>())-a <<endl;   //输出
0
19    return 0;
20 }
```

顺便提一下，STL里还有一个二分查找函数binary_search(first,last,val)，用法类似于lower_bound/upper_bound，但它只能判断val是否在first和last的有序区间里存在，如果val存在则返回true，否则返回false。

11.3 vector向量容器



扫码看视频

vector在头文件<vector>中声明，它通过一个连续的长度可变的数组存放任意类型的数据。新增数据时，如果vector空间已满，则扩展vector空间（GNU编译器套件（GNU Compiler Collection,GCC）规定为两倍），将原来的数据复制过来，释放之前的空间再插入新增的数据；如果存放的数据不超过vector空间的一半，则释放一半的vector空间（类似于倍增算法）。

vector在尾端插入和删除元素的时间复杂度为$O(1)$，其他元素插入和删除的时间复杂度为$O(n)$。

用数组方式访问vector元素的参考程序如下所示。

```
 1 //数组方式访问vector元素
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   vector<int> v;        //定义了一个存放整数的vector容器
 8   v.reserve(30);        //调整数据空间大小
 9   v.push_back(20);      //尾端插入新元素
10   v.push_back(26);
11   v.push_back(12);
12   v.push_back(52);
13   v.insert(v.begin(),2);//begin()为指针，指向v的头部,在此处插入2
14   v.insert(v.end(),43); //end()为指针，指向v最后的元素的后面，插入43
15   v.insert(v.begin()+2,15);                    //在第2个元素前插入15
16   v.erase(v.begin()+1);                        //删除第2个元素
17   v.erase(v.begin(),v.begin()+2);              //删除前3个元素
18   v.pop_back();                                //删除末尾的一个元素
19   for (int i=0; i<v.size(); ++i)               //size()为v中元素的
个数
20     cout<<v[i]<<' ';
21   cout<<"\n 首元素为:"<<v.front()<<'\n';      //首元素引用
22   cout<<"末元素为:"<<v.back()<<'\n';          //末元素引用
23   reverse(v.begin(),v.end());                  //反转整个vector元素
24   for (int i=0; i<v.size(); ++i)
25       cout<<v[i]<<' ';
```

```
26    v.clear();                                          //清空全部元素
27    cout<<"\n v是否为空:"<<v.empty()<<'\n';      //判断是否为空
28    return 0;
29 }
```

结构体容器的vector的参考程序如下所示。

```
 1 //结构体容器的vector
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 struct stu
 6 {
 7   int x,y;
 8 };
 9
10 int main()
11 {
12   int j;
13   vector<stu> v1;                                    //结构体容器
14   vector<stu> v2;
15   struct stu a= {1,2};
16   struct stu b= {2,3};
17   struct stu c= {4,5};
18   v1.push_back(a);
19   v1.push_back(b);
20   v1.push_back(c);
21   v2.push_back(c);
22   v2.push_back(b);
23   v2.push_back(a);
24   swap(v1,v2);                                       //两结构体元素交
换
25   for (int i=0; i<v1.size(); i++)                    //输出v1所有元素
26     cout<<v1[i].x<<" "<<v1[i].y<<endl;
27   cout<<"\n";
28   for (int i=0; i<v2.size(); i++)                    //输出v2所有元素
29     cout<<v2[i].x<<" "<<v2[i].y<<endl;
30   return 0;
31 }
```

11.3.1　电话列表(telephone)

【题目描述】

给定一些电话号码，看看其是否符合没有号码是其他号码的前缀的要求。例如有下面这些电话号码：

紧急呼叫 911

琪儿 97625999

琳琳 91125426

在这个案例里，就不可能打电话给琳琳，因为当你拨了琳琳电话号码前3位数字的时候，电话服务中心将拨通紧急呼叫电话。所以这个电话列表不符合要求。

【输入格式】

输入数据的第一行给出了一个整数t（$1 \leqslant t \leqslant 40$），表示整个测试案例的个数。每个测试案例的开头一行是n（$1 \leqslant n \leqslant 10000$），表示电话号码的个数。接下来的$n$行中，每行都是一个不相同的电话号码，每个电话号码都是一个至多10个数字的序列。

【输出格式】

对于每个测试案例，如果它是符合要求的，则输出“YES”，否则输出“NO”。

【输入样例】

1

5

113

12340

123440

12345

98346

【输出样例】

YES

【算法分析】

此题的标准解法是使用数据结构中的字典树，但实际上只要将输入的字符串由小到大排列，然后到下一个字符串中查找上一个字符串即可，如果下一个字符串的最前面的子串是上一个字符串，就终止。

参考程序如下所示。

```
1 //电话列表
2 #include <bits/stdc++.h>
3 using namespace std;
```

```
 4 vector <string> v;
 5 char ch[10];
 6
 7 int Check(int n)
 8 {
 9   for (int i=0; i<n-1; i++)
10     if (v[i+1].find(v[i])==0)//find返回某字符串在另一个字符串中首次出
现的位置
11       return 0;
12   return 1;
13 }
14
15 int main()
16 {
17   int n;
18   scanf("%d",&n);
19   while (~scanf("%d",&n))    //scanf读取数据失败会返回-1，取反即0
20   {
21     v.clear();
22     for (int i=0; i<n; i++)
23     {
24       scanf("%s",&ch);       //scanf比cin运行速度快，读入字符串不会超
时
25       v.push_back(string(ch));
26     }
27     sort(v.begin(),v.end()); //对vector从小到大排序
28     printf("%s\n",Check(n)==0?"NO":"YES");
29   }
30   return 0;
31 }
```

11.3.2　普通平衡树（BalanceTree）

【题目描述】

你需要写一种数据结构（可参考题目标题）来维护一些数，可提供以下操作：

（1）插入x数；

（2）删除x数（若有多个相同的数，应只删除一个）；

（3）查询x数的排名（若有多个相同的数，应输出最小的排名）；

（4）查询排名为x的数；

（5）求x的前驱（前驱定义为小于x，且最大的数）；

（6）求x的后继（后继定义为大于x，且最小的数）。

【输入格式】

第一行为n（n＜180 000），表示操作的个数，下面n行每行有两个数opt和x，opt表示操作的序号（$1 \leqslant opt \leqslant 6$）。

【输出格式】

对于操作3,4,5,6每行输出一个数，表示对应答案。

【输入样例】

10

1 106465

4 1

1 317721

1 460929

1 644985

1 84185

1 89851

6 81968

1 492737

5 493598

【输出样例】

106465

84185

492737

【算法分析】

对于数据结构中的平衡树，代码实现是很复杂的，但幸运的是，题中的操作完全可以使用STL模板中的vector来实现，参考程序如下所

示。

```
 1 //普通平衡树
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 inline int Read()     //inline把函数指定为运行速度更快的内联函数
 6 {
 7   int x=0,f=1;
 8   char ch=getchar();  //使用getchar()比使用scanf()运行速度大约快10
倍
 9   for (; ch<'0' || ch>'9'; ch=getchar())
10     if (ch=='-')
11       f=-1;
12   for (; ch>='0' && ch<='9'; ch=getchar())
13     x=(x<<3)+(x<<1)+ch-'0';//(x<<3)+(x<<1)相当于x*10
14   return x*f;
15 }
16
17 int main()
18 {
19   int n=Read();         //Read()用于快速读取整数
20   vector<int> v;
21   v.reserve(200000);    //预留v的空间大小为200000以免扩展空间浪费时间
22   int f,x;
23   for (int i=1; i<=n; i++)
24   {
25     f=Read(),x=Read();
26     switch (f)
27     {
28         //upper_bound用于找出有序数列中首个大于某值的元素
29       case
1:v.insert(upper_bound(v.begin(),v.end(),x),x);break;
30         //lower_bound用于找出有序数列中首个大于等于某值的元素
31       case 2:v.erase(lower_bound(v.begin(),v.end(),x));break;
32       case 3:cout<<lower_bound(v.begin(),v.end(),x)-
v.begin()+1<<endl;
33             break;
34       case 4:cout<<v[x-1]<<endl;break;
35       case 5:cout<<*--lower_bound(v.begin(),v.end(),x)
<<endl;break;
36       case 6:cout<<*upper_bound(v.begin(),v.end(),x)
<<endl;break;
37     }
38   }
39   return 0;
40 }
```

11.4 pair容器



扫码看视频

pair容器在<utility>头文件中定义，它的作用是将一对值组合成一个值。与之前介绍的容器不同，在创建pair对象时，必须提供两个类型名，两个对应的类型名的类型不必相同。

例如：

```
pair<int, double> p1;          //定义p1为pair类型，包含一个整数和一个浮点数
pair<int, double> p[10];       //定义pair类型的数组p[]
pair<string,string>author("James","Joy");//定义author是pair类型且初始化
```

可以通过first和second访问pair中的两个元素。常用操作的参考程序如下所示。

```
 1 //pair例程1
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   pair<int, double> p1;
 8   p1.first = 10;
 9   p1.second = 12.5;
10   cout<<p1.first<<' '<< p1.second<<endl;
```

```
11    return 0;
12 }
```

还可以使用make_pair来生成需要的pair，参考程序如下所示。

```
 1 //pair例程2
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4 typedef pair<string, double> Record;//typedef简化pair的声明为
Record
 5
 6 int main()
 7 {
 8   Record p1 = make_pair("zxh", 100);
 9   Record p2 = p1;
10   cout<<p2.first<<' '<< p2.second<<endl;
11   return 0;
12 }
```

pair的比较是按照字典序比较的，比较时先按first的大小比较，如果相等，再按second的大小比较，参考程序如下所示。

```
 1 //pair的比较
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main ()
 6 {
 7   pair<int,char> A (10,'z');
 8   pair<int,char> B (90,'a');
 9   if (A==B) cout << "相等\n";
10   if (A!=B) cout << "不相等\n";
11   if (A<B)  cout << "A<B\n";
12   if (A>B)  cout << "A>B\n";
13   if (A<=B) cout << "A<=B\n";
14   if (A>=B) cout << "A>=B\n";
15   return 0;
16 }
```

11.4.1　奇怪的排序（OddSort）

【题目描述】

有一台故障机器人从右往左读取自然数，例如它看到123时会理解成321。让它比较23与15哪一个大，它会说15大，原因是它会以为是32与51在进行比较。

输入A和B（$1 \leqslant A \leqslant B \leqslant 200\,000$），让该故障机器人将$[A, B]$区间中的所有数按从小到大的顺序排序。

【输入格式】

第一行为一个整数N（$1 \leqslant N \leqslant 5$），表示有多少组测试数据。

随后为N行，每一行有两个正整数A和B（$B - A \leqslant 50$），表示待排序元素的区间范围。

【输出格式】

对于每一行测试数据，输出一行，为所有排好序的元素，元素之间以一个空格间隔。

【输入样例】

1

8 15

【输出样例】

10 8 9 11 12 13 14 15

11.5 set集合容器



扫码看视频

set集合容器在头文件<set>中声明，使用类似于树的结构（基于红黑树的平衡二叉检索树）（如图11.4所示）存储数据并自动将数据（无重复值）由小到大排列。构造set集合容器的主要目的是快速检索（时间复杂度为$O(\log N)$），检索效率高于vector、deque及list等容器。

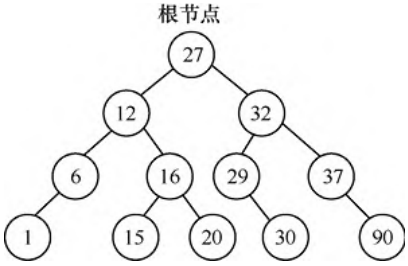


图11.4

访问set集合容器中的元素，需要通过迭代器进行。迭代器类似于指针，可以通过它指向容器中的某个元素的地址。例如set<int>::iterator ii;定义了一个set<int>类型的迭代器为ii。

对set集合容器的部分操作例程如下所示。

```
 1 //set例程
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   set<int> s;
 8   for (int i=10; i>0; --i)                          //此处由大到小赋值
 9     s.insert(i);                                    //插入元素
10   set<int> s2(s);                                   //复制s生成s2
11   s.erase(s.begin());                               //删除操作
12   s.erase(6);
13   s.insert(5);                                      //不会重复插入
14   set<int>::iterator ii;                            //ii为正向迭代器
15   for (ii=s.begin(); ii!=s.end(); ii++)//ii从首地址到末元素地址遍历
16     cout<<*ii<<' ';              //ii为地址，所以取地址上的值前面要加
*
17   cout<<"\n元素个数为"<<s.size();//统计set中元素个数，时间复杂度为
O(1)
18   ii=s.find(10);     //查找元素值，并返回指向该元素的迭代器
19   if (ii!=s.end())   //如果容器中不存在该元素，返回值等于s.end()
20     cout<<"\n查找="<<*ii;
21   if (s.count(5))    //count返回s中值为5的元素个数，时间复杂度为
O(logn)
22     cout<<"\n存在元素5";
23   s.clear();         //清空所有元素
24   cout<<"\n元素是否为空:"<<s.empty();
25   return 0;
26 }
```



迭代器分为正向迭代器和反向迭代器，顾名思义，正向迭代器可以正向遍历容器元素，反向迭代器可以反向遍历容器元素。考虑到反向迭代器的功能可以通过正向迭代器间接实现（例如翻转或自定义排序等方式），所以本书暂不对反向迭代器做更多介绍，感兴趣的读者请自行查询相关资料学习。

使用insert()将元素插入set集合容器的时候，集合容器默认按由小到大的顺序插入元素，然而在很多情况下（例如需要由大到小排序）需要自行编写比较函数。

当元素不是结构体时，编写比较函数的参考程序如下所示。

```
 1 //set的比较函数
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 struct Comp
 6 {
 7   bool operator() (const int &a,const int &b) const  //重载操作符"（）"
 8   {
 9     return a>b;                                  //由大到小排序
10     //return a<b;                                //由小到大排序
11   }
12 };                                               //需在结构体内定义比较函数
13
14 int main()
15 {
16   set<int,Comp> s;                               //set调用的比较函数为Comp（）
17   //set<int,greater<int> >s;                     //其实这样写最简单
18   for (int i=1; i<=10; ++i)                      //此处由小到大赋值
19     s.insert(i);
20   set<int>::iterator ii;
21   for (ii=s.begin(); ii!=s.end(); ii++)  //遍历
```

```
22      cout<<*ii<<' ';                                  //输出是由大到小排序
23    return 0;
24 }
```

重载操作符可以使得自定义类型像基本数据类型一样支持加、减、乘、除、自加、自减等各种操作。operator是C++的关键字，它和操作符（例如“（）”）一起使用，表示运算符重载函数，在理解时可将operator和运算符（例如“operator（）”）视为函数名。

参数定义为const int &a和const int &b这种形式，一是防止a和b被修改，二是通过引用变量的方式减少系统开销。末尾的const也是为了防止被修改。

实际上，很多C++操作符已经被重载了，例如“*”用于两个数字之间时，得到的是它们的乘积；用于一个地址之前时，得到的是这个地址上存储的值。cin>>x和cout<<x中的“>>”和“<<”运算符也被重载了。

例如自定义一个结构体Time类型，并想实现两个Time类型相加的操作（即小时与小时相加，分钟与分钟相加，秒与秒相加），其程序可能如下所示。

```
 1 //Time+Time的操作
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 struct Time
 6 {
 7   int H,M,S;
 8 } T1= {3,2,4},T2= {5,20,30},T3;
 9
10 int main()
11 {
```

```
12   T3=T1+T2;                                          //此行无法通过编译
13   cout<<T3.H<<":"<<T3.M<<":"<<T3.S<<endl;
14   return 0;
15 }
```

显然，程序的第12行是无法通过编译的，因为C++并不知道怎么使用运算符“+”对两个Time类型进行操作，所以需要对运算符“+”进行重载。完整程序如下所示。

```
 1 //Time+Time的操作
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 struct Time
 6 {
 7   int H,M,S;                                         //时，分，秒
 8   Time operator+ (const Time &b)const                //重载运算符"+"
 9   {
10     return Time {H+b.H,M+b.M,S+b.S};                 //仅为演示，不考虑进位
11   }
12 } T1= {3,2,4},T2= {5,20,30},T3;
13
14 int main()
15 {
16   T3=T1+T2;
17   cout<<T3.H<<":"<<T3.M<<":"<<T3.S<<endl;
18   return 0;
19 }
```

C++只能重载已有的操作符。表11.1列出了可以被重载的操作符。

表11.1

+	-	*	/	%	^	&	|	~
!	,	=	<	>	<=	>=	++	--
<<	>>	==	!=	&&	||	+=	-=	/=
%=	^=	&=	|=	*=	<<=?	>>=	[]	()
->	->*	new	new[]	delete	delete[]			

当元素是结构体时，必须要重载运算符“<”，参考程序如下所示。

```
 1 //set的结构体排序
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 struct Info
 6 {
 7   string name;
 8   double score;
 9   bool operator < (const Info &a) const     //必须重载操作符"<"
10   {
11     return a.score <score; //从大到小排序（若改为">"则为从小到大排序）
12   }
13 } info;
14
15 int main()
16 {
17   set<Info> s;
18   info= {"A",90.0};
19   s.insert(info);
20   info= {"B",92.0};
21   s.insert(info);
22   info= {"C",96.0};
23   s.insert(info);
24   set<Info>::iterator ii;
25   for (ii=s.begin(); ii!=s.end(); ii++)                    //遍历
26     cout<<(*ii).name<<' '<<(*ii).score<<endl;
27   return 0;
28 }
```

11.5.1　问卷调查（random）

【题目描述】

小光用计算机生成了N个1～1 000的随机整数，对于其中重复的数，只保留一个，把其余相同的数去掉，然后把这些数从小到大排

序，按照排好的顺序查找对应的学员做调查。请协助小光完成“去重”与“排序”的工作。

【输入格式】

输入有两行，第一行为一个正整数，表示所生成的随机数的个数N（$N \leqslant 100$）。第二行有N个以空格隔开的正整数，表示所生成的随机数。

【输出格式】

输出有两行，第一行为一个正整数M，表示不相同的随机数的个数。第二行为M个以空格隔开的正整数，表示从小到大排好序的不相同的随机数。

【输入样例】

10

20 40 32 67 40 20 89 300 400 15

【输出样例】

8

15 20 32 40 67 89 300 400

11.5.2 两倍（Double）

【题目描述】

有一组随机产生的包含2～15个不重复的正整数的列表，要求说出这个列表中有多少对数字，其中一个数字是该列表中的其他数字的两倍。例如列表为1 4 3 2 9 7 18 22，由于2是1的两倍、4是2的两倍、18是9的两倍，因此答案为3。

【输入格式】

每组测试数据为一行，每行包含2～15个不重复的正整数（均不大于99），每行末尾的0仅作为结束标志使用，一行上只有一个整数-1表示文件的结束。

【输出格式】

每组测试数据应当输出一行，打印出测试案例中具有两倍关系的数的个数。

【输入样例】

1 4 3 2 9 7 18 22 0

2 4 8 10 0

7 5 11 13 1 3 0

-1

【输出样例】

3

2

0

【算法分析】

数据输入set集合容器中，会自动由小到大排序。再在集合中使用find()查找每个元素的两倍的元素是否存在。由于集合是平衡二叉检索树，因此查询速度是极快的。

11.6 multiset多重集合容器



扫码看视频

multiset多重集合容器在头文件<set>中定义，它可以看成序列，序列中可以存在重复的数。multiset能时刻保证序列中的数是有序的（默认从小到大排序），插入一个数或删除一个数都能够在$O(\log n)$的时间内完成。

multiset多重集合容器的一些简单操作的参考程序如下所示。

```
 1 //multiset例程
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   multiset<int> m;
 8   m.insert(11);                                          //插入数据
 9   m.insert(21);
10   m.insert(10);
11   m.insert(12);
12   m.insert(12);
13   m.insert(11);
14   m.insert(11);
15   m.insert(11);
16   m.insert(9);
17   cout<<"11的个数有"<<m.count(11)<<endl;//count返回m中11的个数
18   cout<<"第一个大于等于10的元素为："<<*m.lower_bound(10)<<endl;
19   cout<<"第一个大于11的元素为:"<<*m.upper_bound(11)<<endl;
20   multiset<int>::iterator it;
21   for (it=m.begin(); it!=m.end(); it++)
22     cout<<*it<<endl;                                     //从小到大输出
23   cout<<"删除12,有"<<m.erase(12)<<"个"<<endl;      //删除等于12的元
素
24   cout<<"查找9\n";
25   multiset<int>::iterator i=m.find(9);//查找v，返回该元素的迭代器位
置
26   if (i!=m.end())                  //找到则输出，否则i为end()迭代器位
置
27     cout<<*i<<endl;
28   int v=11;                                  //查找所有相同元素
29   pair<multiset<int>::iterator,multiset<int>::iterator>p;
```

```
30    //equal_range:有序容器中表示一个值第一次出现与最后一次出现的后一位
31    p=m.equal_range(v);
32    cout<<"大于等于"<<v<<"的第一个元素为"<<*p.first<<endl;
33    cout<<"大于"<<v<<"的第一个元素为"<<*p.second<<endl;
34    cout<<"键值为"<<v<<"的全部元素为";
35    for (it=p.first; it!=p.second; it++) //打印重复键值元素11
36      cout<<*it<<" ";
37    m.clear();                                  //清空所有元素
38    return 0;
39  }
```

11.6.1　12!配对(12)

【题目描述】

找出输入数据中有多少对数两两相乘的积为12!。

【输入格式】

输入数据中第一行为个数n，随后是n个整数m（$1 \leqslant m<2^{32}$）。

【输出格式】

输出有多少对数两两相乘的积为12!。

【输入样例】

8

1 10000 159667200 9696 38373635 1000000 479001600 3

【输出样例】

2

【算法分析】

输入的整数如果是12!的约数，则从多重集合容器中查找到该数对应的因子，答案个数加1，然后从多重集合容器中删除该因子。若找不到该数对应的因子，则先将该数插入多重集合容器中，以供后面输入的数查找对应因子。

参考程序如下所示。

```
 1 //12!配对
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int N,n,num=0,f12=479001600;
 8   multiset<unsigned int>s;                          //多重集合容器，允许
值重复
 9   cin>>N;
10   for (int i=1; i<=N; i++)
11   {
12     cin>>n;
13     if (f12%n==0)                                   //如果n是f12的约数
14     {
15       multiset<unsigned int>::iterator it=s.find(f12/n);//找n的
因子
16       if (it!=s.end())
17       {
18         ++num;
19         s.erase(it);                                //从多重集合容器中删
除该因子
20       }
21       else
```

```
22          s.insert(n);                                    //将n插入多重集合容器
中
23      }
24    }
25    printf("%d\n",num);
26    return 0;
27  }
```

11.6.2　01串排序(Sort01)

【题目描述】

将01串首先按长度由小到大排序，长度相同时按1的个数由少到多排序，1的个数相同时再按ASCII值排序。

【输入格式】

输入数据中第一行为整数*n*，表示有*n*个01串，随后是*n*个01串，01串的长度不大于256个字符。

【输出格式】

重新排列01串的顺序，使得01串按题目描述的方式排序。

【输入样例】

6

10011111

00001101

1010101

1

0

1100

【输出样例】

0

1

1100

1010101

00001101

10011111

参考程序如下所示。

```
1 //01串排序
2 #include <bits/stdc++.h>
```

```
 3 using namespace std;
 4
 5 struct Comp
 6 {
 7   bool operator()(const string &s1,const string &s2)const//重载
操作符"（）"
 8   {
 9     if (s1.length()!=s2.length())                    //先按长度排
序
10       return s1.length()<s2.length();
11     int c1=count(s1.begin(),s1.end(),'1');//count返回s1中1的个数
12     int c2=count(s2.begin(),s2.end(),'1');
13     return c1!=c2?c1<c2:s1<s2; //按1的个数排序，若个数相同则按ASCII
值排序
14   }
15 };
16
17 int main()
18 {
19   multiset<string,Comp>ms;
20   int n;
21   cin>>n;
22   string s;
23   for (int i=1; i<=n; i++)
24   {
25     cin>>s;
26     ms.insert(s) ;
27   }
28   multiset<string,Comp>::iterator it;
29   for (it=ms.begin(); it!=ms.end(); it++)
30     cout<<*it<<endl;
31   return 0;
32 }
```

11.6.3　卡片游戏（CardGame）

【题目描述】

琪儿和琳琳分别有不同的矩形卡片，已知A卡片可以覆盖B卡片的条件是A卡片的高度不小于B卡片的高度且A卡片的宽度不小于B卡片的

宽度，每张卡片只能使用一次，而且卡片不能旋转。试计算琪儿的卡片可以覆盖琳琳的卡片的最大张数。（题目来源：HDU 4268）

【输入格式】

输入的第一行是t（$t \leqslant 40$），表示测试用例的数量。

对于每一种情况，第一行是n，表示琪儿和琳琳分别拥有的卡片数。下面n（$n \leqslant 100\,000$）行中的每一行都包含两个整数h（$h \leqslant 1000\,000\,000$）和$w$（$w \leqslant 1\,000\,000\,000$），分别表示琪儿的卡片的高度和宽度，再下面的$n$行表示琳琳的卡片的高度和宽度。

【输出格式】

对于每个测试用例，使用一行包含一个数字的方法输出结果。

【输入样例】

1

3

2 3

5 7

6 8

4 1

2 5

3 4

【输出样例】

2

11.7　deque双端队列容器



扫码看视频

deque在头文件<deque>中声明，它是一种双向开口的连续线性空间，可以高效地在头、尾两端插入和删除元素。它的时间复杂度为$O(1)$，考虑容器元素的内存分配策略和操作性能时，deque比vector有优势。

deque的部分操作参考程序如下所示。

```
 1 //deque例程
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   deque<string>d;                          //定义一个包含string的deque
 8   d.push_back("A");                        //尾部插入元素
 9   d.push_back("B");
10   d.push_front("X");                       //头部插入元素
11   d.push_front("Y");
12   //d.pop_front();                         //删除首元素
13   //d.pop_back();                          //删除尾元素
14   //d.erase(d.begin()+1);                  //删除指定位置的元素
15   //d.clear();                             //删除所有元素
16   d.insert(d.end()-2,"O");                 //指定位置插入
17   reverse(d.begin(),d.end());              //反转元素顺序
18   for (int i=0; i<d.size(); i++)           //数组方式访问
19     cout<<d[i]<<" ";
20   cout<<endl;
21   swap(d[1],d[2]);                         //两个元素交换
22   deque<string>::iterator i;               //迭代器访问
23   for (i=d.begin(); i!=d.end(); i++)//正向遍历
24     cout<<*i<<" ";
25   cout<<endl;
26   cout<<"\ndeque是否为空 "<<d.empty();
27   cout<<"\ndeque元素个数为 "<<d.size();
28   cout<<"\ndeque的首元素为 "<<d.front();
29   cout<<"\ndeque的末元素为 "<<d.back();
30   return 0;
31 }
```

11.7.1　鸡蛋队列(egg)

【题目描述】

如图11.5所示，将两根筷子平行地放在一起，就可构成一个队列。将带有编号的鸡蛋放到两根筷子之间叫作入队（push），将筷子之间的鸡蛋拿出来叫作出队（pop）。但这两种方式有特殊的定义，对

于入队，只能将鸡蛋从队列的尾部向里放入；对于出队，只能将鸡蛋从队列的头部向外拿出。



图11.5

【输入格式】

第一行输入一个数T，表示有T组数据。

第二行输入一个数N，表示有N（$N \leqslant 10$）种操作。

接下来输入N行，每行一种操作，push表示将编号为x的鸡蛋放入队列中，pop表示拿走队列头部的一个鸡蛋，保证队列中没有鸡蛋时不会有出队操作。

【输出格式】

每组数据输出占一行，输出N种操作完成之后队列中鸡蛋的编号，如果没有鸡蛋就输出“no eggs!”(不包括引号）。

【输入样例】

2

3

push 1

push 2

push 3

2

push 1

pop

【输出样例】

1 2 3

no eggs!

11.8 list双向链表容器



扫码看视频

list双向链表容器在头文件<list>中声明，它对任一位置元素的查找、插入及删除都具有高效的常数阶算法时间复杂度。它的结构示意如图11.6所示。

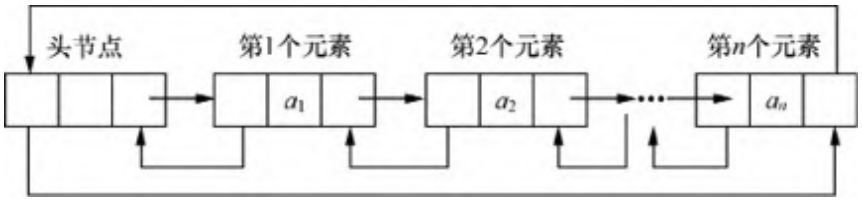


图11.6

参考程序如下所示。

```
 1 //list例程
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   list<int>l;
 8   l.push_back(2);                          //尾部插入新元素，链表自动
扩张
 9   l.push_back(2);
10   l.push_back(9);
11   l.push_back(12);
12   l.push_back(12);
13   l.push_back(4);
14   //l.clear();                             //清空链表
15   l.push_front(9);                         //头部插入新元素，链表自动
扩张
16   list<int>::iterator it;
17   it=l.begin();
18   it++;  //链表迭代器只能++或--操作，不能+n操作，因为list节点非连续内存
19   l.insert(it,20);                         //当前位置插入新元素
20   it++;
21   l.erase(it);                             //删除迭代器位置上的元素
22   for (it=l.begin(); it!=l.end(); it++) //正向遍历
23     cout<<*it<<" ";
```

```
24   cout<<endl;
25   l.remove(12);                                  //删除所有值为12的元素
26   l.pop_front();                                 //删除链表首元素
27   l.pop_back();                                  //删除链表尾元素
28   it=find(l.begin(),l.end(),4);                  //查找值为4的元素
29   if (it!=l.end())
30     cout<<”find "<<*it<<endl;
31   l.sort();                                      //升序排列
32   l.unique();                                    //删除连续重复元素(只保留一
个)
33   for (it=l.begin(); it!=l.end(); it++) //正向遍历
34     cout<<*it<<" ";
35   cout<<endl;
36   return 0;
37 }
```

结构体的参考程序如下所示。

```
 1 //结构体list例程
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 struct student
 6 {
 7   char *name;
 8   int age;
 9   char *city;
10 };
11
12 int main()
13 {
14   student s[]=
15            {{"张三",18,"浙江"},{"李四",19,"北京"},{"王二",18,"上
海"}};
16   list<student>l;
17   l.push_back(s[0]);                             //插入元素
18   l.push_back(s[1]);
19   l.push_back(s[2]);
20   student x= {"刘四",19,"四川"};
21   l.push_front(x);                               //插入首位，时间复杂度为
O(1)
22   l.insert(l.begin(),x);                         //插入任意位置，时间复杂度
为O(1)
23   //l.pop_front();                               //删除首元素
24   //l.pop_back();                                //删除尾元素
25   l.erase(l.begin());
```

```
26    //l.erase(l.begin(),l.end());           //删除区间的元素
27    for (list<student>::iterator i=l.begin(); i!=l.end(); i++)
28      cout<<(*i).name<<"  "<<(*i).age<<"  "<<(*i).city<<"\n";
29    return 0;
30  }
```

11.8.1　队列训练（train）

【题目描述】

将学生按顺序依次编号后使其排成一行横队进行队列训练，训练的规则如下：从头开始进行1～2报数，凡报到2的学生出列，剩下的向小序号方向靠拢；再从头开始进行1～3报数，凡报到3的学生出列，剩下的向小序号方向靠拢；继续从头开始进行1～2报数……以后从头开始轮流进行1～2报数、1～3报数，直到剩下的人数不超过3人为止。（题目来源：HDU 1276）

【输入格式】

本题有多个测试数据组，第一行为组数*N*，接着为*N*行，表示学生人数，人数不超过5 000。

【输出格式】

输出共*N*行，每行输出剩下的学生最初的编号，编号之间以一个空格间隔。

【输入样例】

2

20

40

【输出样例】

1 7 19

1 19 37

11.9 map映照容器



扫码看视频

map映照容器在头文件<map>中定义，它的元素数据是由键值和映照数据组成的，键值与映照数据之间具有一一对应的关系，如图11.7所示。

键值	映照数据
Name	Score
Alice	98
Mary	96
Jack	94

图11.7

map映照容器和set集合容器一样都是采用红黑树来实现的，插入键值的元素不允许重复，元素默认是按键值由小到大排序的。如果定义比较函数，比较函数也只能对元素的键值进行比较，元素的各项数据可通过键值检索出来。

map与set的区别是：map是处理带有键值的记录型元素数据的快速插入、删除及检索，而set是对单一数据的处理。

简单的参考程序如下所示。

```
 1 //map例程1
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   map<char*,float> m;
 8   m["apple"]=3.4;
 9   m["orange"]=1.2;
10   m["pear"]=3.5;
11   cout<<m["apple"]<<endl;
12   cout<<m["orange"]<<endl;
13   cout<<m["pear"]<<endl;
14   m.clear();
15   return 0;
16 }
```

又一个使用map映照容器的参考程序如下所示。

```
 1 //map例程2
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   map<int, string> ms;
 8   ms[1] = "student_one";
 9   ms[1] = "student_two";                    //id相同，则覆盖
10   ms[2] = "student_three";
11   map<int, string>::iterator iter;
12   ms.insert(make_pair(3,"student_four")); //插入新元素，用
make_pair
13   for (iter = ms.begin(); iter != ms.end(); iter++)
14     cout<<iter->first<<" "<<iter->second<<endl;
15   cout<<endl;
16   iter=ms.lower_bound(1);                   //首个大于等于1的元素
17   cout<<iter->second<<endl;
18   iter=ms.upper_bound(1);                   //首个大于1的元素
19   cout<<iter->second<<endl;
20   iter = ms.find(1);                        //查找键值为1的元素的位置
21   ms.erase(iter);                           //删除键值为1的元素
22   for (iter = ms.begin(); iter != ms.end(); iter++)
23     cout<<iter->first<<" "<<iter->second<<endl;
24   ms.erase(ms.begin(),ms.end());            //删除全部元素
25   cout<<ms.size()<<endl;
26   cout<<ms.empty()<<endl;                   //empty()判断map是否为
空
27   return 0;
28 }
```

结构体参考程序如下所示。

```
 1 //结构体map例程
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 struct Info                                 //学生信息结构体
 6 {
 7   char *xm;                                 //姓名
 8   int y;                                    //年份
 9   char *d;                                  //地址
10 };
11
12 struct Record                               //学生记录结构体
13 {
```

```
14    int id;                                          //学号作为键值
15    Info sf;                                         //学生信息作为映照数据
16 };
17
18 int main()
19 {
20    Record srArray[]={{4,"Li",21,"beijing"},
21                      {2,"wang",29,"shanghai"},
22                      {3,"zhang",30,"shenzhen"}};
23    map<int,Info, greater<int> >m;                   //按键值由大到小排序
24    for (int j=0; j<3; j++)                          //装入3个学生的信息
25      m[srArray[j].id]=srArray[j].sf;
26    Record s1= {5,"Ling",23,"SICHUAN"};
27    m.insert(make_pair(s1.id,s1.sf));                //插入新生信息
28    map<int,Info>::iterator i;
29    for (i=m.begin(); i!=m.end(); i++)               //正向遍历
30      cout<<(*i).first<<' '<<(*i).second.xm<<'
'<<(*i).second.d<<'\n';
31    i=m.find(2);                                     //查找键值为2的记录并输出
32    cout<<"键值2: "<<(*i).second.xm<<' '<<(*i).second.d;
33    return 0;
34 }
```

11.9.1 射箭（toxophily）

【题目描述】

小光喜欢射箭运动，他一次可以射下一串排成一行的气球，例如有如下4个射箭位置。

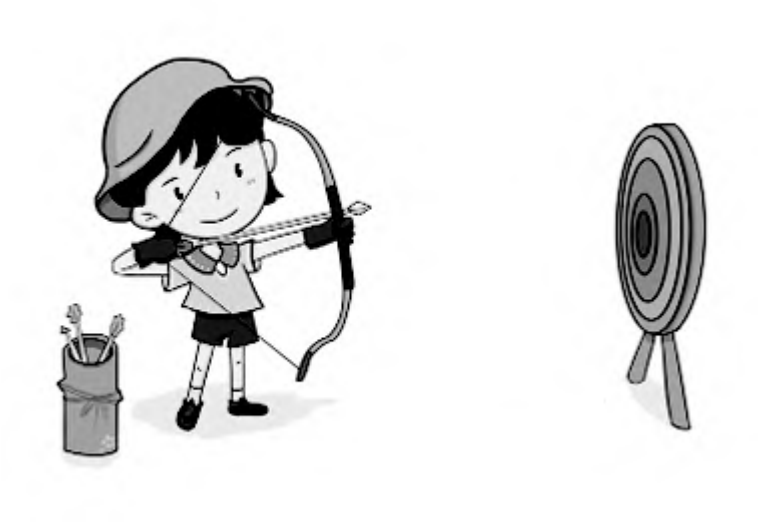
1 → ○○○○

2 → ○

3 → ○○○

4 → ○○

如果小光想一箭射下3个气球，他就站在3号位置；如果小光想一箭射下4个气球，他就站在1号位置。现在他想射下m个气球，请问他应该站在几号位置？



使用“内容市场”APP扫描看视频

【输入格式】

第1行为一个正整数n（$1 \leqslant n \leqslant 100\,000$），表示射箭位置数。

第2行包含 n 个正整数，第 i 个数表示第 i 个位置的气球数 a_i（$1 \leqslant a_i \leqslant 1\,000\,000\,000$），各个位置的气球数不同。

第3行包含一个正整数V（$1 \leqslant V \leqslant 100\,000$），表示射箭的次数。

随后为V行，每行一个正整数m（$1 \leqslant m \leqslant 1\,000\,000\,000$），表示需要射下的气球数。

【输出格式】

输出V行，每行包含一个正整数，表示每次的射箭位置。若无法完成则输出0。

【输入样例】

5

2 1 4 3 5

2

5

6

【输出样例】

5

0

【算法分析】

考虑到数据规模，一般的算法是对气球数排序后再进行二分查找，但实际使用map映照容器更为简捷。

参考程序如下所示。

```
//射箭
#include <bits/stdc++.h>
using namespace std;

map <int,int> Map;

int main()
{
  int t,n,v,m;
  scanf("%d",&n);
  for (int i=1; i<=n; i++)
  {
    scanf("%d",&t);
    Map[t]=i;
  }
  cin>>v;
  for (int i=1; i<=v; i++)
  {
    scanf("%d",&m);
    printf("%d\n",Map[m]);
  }
  return 0;
}
```

11.9.2 漂亮数字(pretty)

【题目描述】

所有能被3或5整除的正整数都是漂亮数字。试找到第N个漂亮数字。

【输入格式】

输入包含多个测试案例。每个测试案例是一个整数N（$1 \leqslant N \leqslant 100\,000$），占一行。

【输出格式】

对于输入数据中的每个测试案例，在单独一行上输出结果。

【输入样例】

1

2

3

4

【输出样例】

3

5

6

9

【算法分析】

建立1～100 000个漂亮数字的数字表到map映照容器中，然后直接查表就行，因为基于红黑树的map映照容器的检索速度很快。

参考程序如下所示。

```
//漂亮数字
#include <bits/stdc++.h>
using namespace std;

map<int,int>m;

int main()
{
  int n,i=0,p=0;
  while (1)
  {
    i++;
    if (i%3==0 || i%5==0)
    {
      ++p;
      if (p>100000)
        break;
      m[p]=i;
    }
  }
  map<int,int>::iterator it;
  while (cin>>n)
    cout<<m[n]<<endl;
  return 0;
}
```

11.9.3　彩色石头(stone)

【题目描述】

小光手里有许多五颜六色的石头，试统计哪种颜色的石头数量最多。

【输入格式】

输入数据包含多组测试案例，每组测试案例由N（$0<N<1\ 000$）开始，N表示石头数量。接下来的N行中每行包含一种颜色，颜色以最多达15个小写字母的单词表示。

当一组测试案例的$N=0$时，表示输入结束。

【输出格式】

对于每组测试案例，把数量最多的颜色在单独一行上输出。每组测试案例都仅有一种颜色是最多的。

【输入样例】

```
5
green
red
blue
```

red

red

0

【输出样例】

red

11.9.4　水果（fruit）

【题目描述】

请你帮水果店老板制作一份水果销售情况明细表。（题目来源：HDU 1263）

【输入格式】

第一行为正整数N（$0<N \leqslant 10$），表示有N组测试数据。

每组测试数据的第一行是一个整数M（$0<M \leqslant 100$），表示已有M次成功的交易。其后有M行数据，每行表示一次交易，由水果名称（小写字母组成，长度不超过80）、水果产地（小写字母组成，长度不超过80）及销售的水果数目（正整数，不超过100）组成。

【输出格式】

对于每一组测试数据，请你输出一份排版格式正确的水果销售情况明细表（请分析样本输出）。这份明细表包括所有水果的产地、名称及销售数目的信息。水果先按产地分类，产地按字母顺序排序；同一产地的水果按照名称排序，名称按字母顺序排序。

如果存在多组测试数据，则两组测试数据之间有一个空行，最后一组测试数据之后没有空行。

【输入样例】

```
1

5

apple shandong 3

pineapple guangdong 1

sugarcane guangdong 1

pineapple guangdong 3

pineapple guangdong 1
```

【输出样例】

guangdong

|----pineapple(5)

|----sugarcane(1)

shandong

|----apple(3)

【算法分析】

本题需要通过产地找到水果名称，通过水果名称找到它的销售数目。所以这里定义一个双重map结构即map<string,map<string,int> >，其中的3个变量分别是产地、水果名称及销售数目。

参考程序如下所示。

```
 1 //水果
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   int T,n,Count;
 8   cin>>T;
 9   while (T--)
10   {
11     cin>>n;
12     char name[100],place[100];
13     map<string,map<string,int> > list;         //注意">"之间要有空
格
14     while (n--)
15     {
16       cin>>name>>place>>Count;
```

```
17        list[place][name]+=Count;                    //保存产地、水果名称
及销售数目
18      }
19      map<string,map<string,int>>::iterator it;
20      map<string,int>::iterator it1;
21      for (it=list.begin(); it!=list.end(); it++)
22      {
23        cout<<it->first<<endl;
24        for (it1=(it->second).begin(); it1!=(it->second).end();
it1++)
25        {
26          cout<<"   |----";
27          cout<<it1->first<<'('<<it1->second<<')'<<endl;
28        }
29      }
30      list.clear();
31      if (T!=0)
32        cout<<endl;
33    }
34    return 0;
35  }
```

11.9.5　A—B(sub)

【题目描述】

给出一串数和一个数C，要求计算出所有$A-B=C$的数对的个数（不同位置的数一样的数对算不同的数对）。

【输入格式】

第一行包括两个非负整数N（$N \leqslant 200\,000$）和C，中间以空格间隔。

第二行有N个整数，均在整数范围内。

【输出格式】

输出一行，表示该串数中包含的所有满足$A-B=C$的数对的个数。

【输入样例】

5 2

1 2 2 3 4

【输出样例】

3

11.10　排列组合关系算法



扫码看视频

STL提供了两个用来分析排列组合关系的算法，分别是next_permutation()和prev_permutation()。例如有3个字符所组成的序列{a，b，c}，这个序列有6个可能的排列，即abc、acb、bac、bca、cab、cba，使用next_permutation()或prev_permutation()可以很方便地生成该序列的全部可能排列。其中：

（1）next_permutation()按照字典序产生排列，并且是从数组中当前的字典序开始依次增大直至到最大字典序；

（2）prev_permutation()按照字典序产生排列，并且是从数组中当前的字典序开始依次减小直至到最小字典序。

参考程序如下所示。

```
 1 //全排列例程
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 void Print(int a[])
 6 {
 7   for (int i=0; i<5; i++)
 8     cout<<a[i]<<" ";
 9   cout<<endl;
10 }
11
12 int main()
13 {
14   int a[]= {3,5,6,7,9};    //随机赋值，如果要全排列，值必须由小到大
15   while (next_permutation(a,a+5)) //产生下一排列,速度较慢,时间复杂度为O(n!)
16     Print(a);
17   int b[]= {5,4,3,2,1};            //随机赋值
18   while (prev_permutation(b,b+5)) //产生上一排列
19     Print(b);
20   return 0;
21 }
```

11.10.1　进制位（system）

【题目描述】

给出如下的一张加法表，表中的字母代表数字。

+	L	K	V	E
L	L	K	V	E
K	K	V	E	KL
V	V	E	KL	KK
E	E	KL	KK	KV

其中L+L=L，L+K=K，L+V=V，L+E=E，K+L=K，K+K=V，K+V=E，K+E=KL，…，
E+E=KV

根据这些规则可推导出：L＝0，K＝1，V＝2，E＝3。同时可以确定该表为四进制加法表。

【输入格式】

第一行为一个整数n（$n \leqslant 9$），表示行数。

随后为n行，每行包括n个字符串，每个字符串间以空格间隔（仅有一个字符串包含“+”，其他都由大写字母组成）。

【输出格式】

第一行按给出的字母顺序输出对应的值，第二行输出其进制，如果加法表错误则输出“ERROR！”

【输入样例】

```
5
+ L K V E
L L K V E
K K V E KL
V V E KL KK
E E KL KK KV
```

【输出样例】

```
L=0 K=1 V=2 E=3
4
```

11.11 stable_sort 稳定排序



扫码看视频

stable_sort 和 sort的区别在于前者排序后可以使原来的“相同”的值在序列中的相对位置不变。例如初始序列中有两个元素A和B的值相等，且A排在B的前面，排序后A仍然排在B的前面，这种排序叫作稳定排序。参考程序如下所示。

```
 1 //stable_sort例程
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 struct stu
 6 {
 7   int id;
 8   char *name;
 9   int score;
10 };
11
12 bool ComByscore(stu s1,stu s2)                          //按分数排
序
13 {
14   return s1.score<s2.score?1:0;
15 }
16
17 bool ComByid(stu s1,stu s2)                             //按id排序
18 {
19   return s1.id>s2.id?1:0;
20 }
21
22 int main()
23 {
24   vector<stu> v;
25   struct stu master;
26   master.id = 3;
27   master.name = "WangDong";
```

```
28    master.score=60;
29    v.push_back(master);                                    //插入数据
30    master.id = 2;
31    master.name = "Liuqiang";
32    master.score=99;
33    v.push_back(master);                                    //插入数据
34    master.id = 1;
35    master.name = "qiangYi";
36    master.score=99;
37    v.push_back(master);                                    //插入数据
38    stable_sort(v.begin(),v.end(),ComByscore);              //按分数排
序
39    for (int i=0; i<v.size(); i++)
40      cout<<v[i].id<<' '<<v[i].name<<' '<<v[i].score<<endl;
41    stable_sort(v.begin(),v.end(),ComByid);                 //按id逆排
序
42    for (int i=0; i<v.size(); i++)
43      cout<<v[i].id<<' '<<v[i].name<<' '<<v[i].score<<endl;
44    return 0;
45  }
```

11.11.1　给朋友排序（friends）

【题目描述】

每个人都会有N个朋友，每个人的名字都由“姓”和“名”两部分组成。你需要对你的朋友按照姓的“流行程度”（即拥有该姓的朋友人数）从大到小排序，姓的流行程度相同的朋友按照他们在原始名单中出现的顺序排序。

【输入格式】

输入排序前的原始序列，每行包含一个朋友的姓和名（以空格间隔），朋友数不超过

50 000，每行为3~50个字符，且只包含大写字母和一个空格（行首、行末无空格）。

【输出格式】

输出排序后的序列，每行包括一个朋友的姓和名，中间以一个空格间隔。

【输入样例】

```
ZHANG SAN
LI SI
WANG WU
WANG LIU
WANG QI
ZHANG WU
LI WU
```

【输出样例】

```
WANG WU
```

WANG LIU

WANG QI

ZHANG SAN

LI SI

ZHANG WU

LI WU

参考程序如下所示。

```
 1 //给朋友排序
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 map<string,int>freq;//字符串到整数映射，保存姓出现的次数，数组下标为字符串
 6
 7 string Key(const string &s)           //输入全名，返回姓
 8 {
 9   return s.substr(0,s.find(" "));     //找到第一个空格，然后取它前面的字符串
10 }
11
12 int Cmp(const string &a,const string &b)
13 {
14   return freq[Key(a)]>freq[Key(b)];
15 }
16
17 int main()
18 {
19   vector<string> v;
20   string s1,s2;
21   while (cin>>s1>>s2)
22   {
23     v.push_back(s1+" "+s2);//用push_back,pop_back往末尾添加或删除元素
```

```
24      ++freq[s1];
25    }
26    stable_sort(v.begin(),v.end(),Cmp); //稳定排序，可保持排序前的顺序
27    for (int i=0; i<(int)v.size(); i++)
28      cout<<v[i]<<endl;
29    return 0;
30  }
```

11.12 multimap多重映照容器



扫码看视频

multimap与map功能基本相同，唯独不同的是multimap允许插入重复键值的元素，由于允许重复键值的元素存在，因此multimap的元素的插入、删除及查找都与map不相同。

multimap的简单参考程序如下所示。

```
 1 //multimap例程
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   multimap <string, double> mp;      //定义multimap对象，当前无任何元素
 8   mp.insert(pair<string, double>("Jack", 300.5));//插入元素
 9   mp.insert(pair<string, double>("Kity", 200));
10   mp.insert(pair<string, double>("Memi", 500));
11   mp.insert(pair<string, double>("Jack", 306));  //重复插入键值"Jack"
12   multimap <string, double>:: iterator it;
```

```
13    mp.erase("Jack");                          //删除键值等于"Jack"的元素
14    for(it=mp.begin(); it != mp.end(); it++)//正向迭代器中序遍历
multimap
15      cout << (*it).first << " " << (*it).second << endl;
16    it = mp.find("Nacy");                      //元素的查找
17    if (it != mp.end())                        //找到元素
18      cout << (*it).first << " " << (*it).second << endl;
19    else                                       //没找到元素
20      cout << "Not find it!" << endl;
21    return 0;
22 }
```

11.12.1　银行业务(bank)

【题目描述】

每个办理银行业务的客户有一个整数编号和优先级，银行有4种代码代表4种操作，如下。

0：结束系统。

1 K P：把一个编号为K的客户加入系统队列，他的优先级是P。

2：输出最高优先级的客户的名字，同时从系统队列中删除该客户的名字。

3：输出最低优先级的客户的名字，同时从系统队列中删除该客户的名字。

【输入格式】

输入的每一行包含一个可能的请求，只有最后一行包含停止请求（代码0）。同一操作中保证优先级是唯一的。编号K总是小于10^6，优先级P小于10^7。客户可以多次办理业务，每次可获得不同的优先级。

【输出格式】

对于每个代码2或3的操作，程序必须在标准输出的单独行中打印客户的编号。如果操作时系统队列为空，则程序输出0。

【输入样例】

```
2
1 20 14
1 30 3
2
1 10 99
3
2
2
0
```

【输出样例】

0

20

30

10

0

【算法分析】

此题标准的解法是手写数据结构中的平衡树，但实际上使用multimap多重映照容器更容易实现。

参考程序如下所示。

```
//银行业务
#include <bits/stdc++.h>
using namespace std;

multimap<int, int>G;
multimap<int ,int>::iterator it;

int main()
{
  int Request;
  while (1)
  {
    scanf("%d", &Request);
    if (Request==1)
    {
      int k, p;
```

```
17        scanf("%d%d", &k, &p);
18        G.insert(make_pair(p, k));//插入客户元素，multimap自动按p的值
排序
19      }
20      if (Request==3)
21      {
22        it=G.begin();
23        if (it==G.end())
24          printf("0\n");
25        else
26        {
27          printf("%d\n", it -> second);  //输出最低优先级客户的名字
28          G.erase(it);
29        }
30      }
31      if (Request==2)
32      {
33        it=G.end();
34        if (G.begin() == it)
35          printf("0\n");
36        else
37        {
38          --it;
39          printf("%d\n", it -> second);  //输出最高优先级客户的名字
40          G.erase(it);
41        }
42      }
43      if (Request==0)
44        break;
45    }
46    return 0;
47  }
```

11.13 stack堆栈容器



扫码看视频

堆栈类似于一个一端开口、一端封闭的容器，开口端称为栈顶（Stack Top），封闭端称为栈底（Stack Bottom），元素的插入和删除只能在栈顶操作。堆栈的元素插入称为入栈，元素的删除称为出栈。堆栈的主要特点是“后进先出”（Last In First Out），即后入栈的元素先出栈。每次入栈的元素都放在原当前栈顶元素之上，成为新的栈顶元素，每次出栈的元素都是原当前栈顶元素，如图11.8所示。

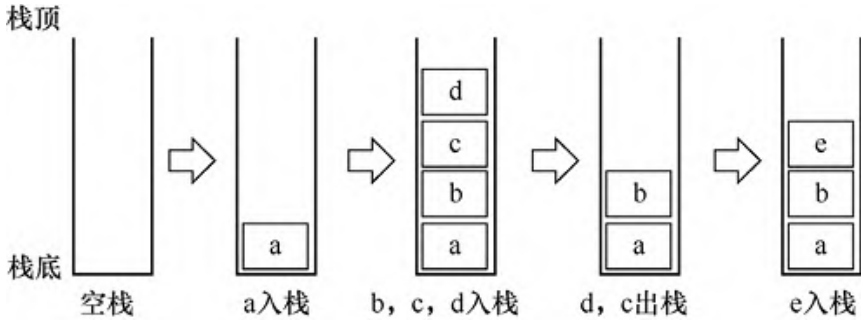


图11.8

C++语言 STL的stack堆栈容器不设最大容量，提供入栈、出栈、栈顶元素访问及判断是否为空的基本操作，其在头文件<stack>中定义。参考程序如下所示。

```
 1 //stack例程
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4 const int STACK_SIZE=100;
 5
 6 int main()
 7 {
 8   stack<string> s;
 9   s.push("aaa");                                    //入栈
10   s.push("bbb");
11   if (s.size()<STACK_SIZE)                          //可限制大小
12     s.push("68");
```

```
13    cout<<s.size()<<endl;                                    //输出栈内元素个
数
14    while (!s.empty())                                       //当堆栈不为空时
15    {
16      cout<<s.top()<<endl;                                   //输出栈顶元素
17      s.pop();                                               //栈顶元素出栈
18    }
19    return 0;
20  }
```

11.13.1　字符栈（CharStack）

【题目描述】

依次对小写英文字符a～z进行栈操作，操作方法是入栈两次，出栈一次，试输出最后栈内的元素。

参考程序如下所示。

```
 1 //字符栈
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   stack <char> st;
 8   for (char i='a'; i<='z';)
 9   {
10     st.push(i++);
11     st.push(i++);
12     st.pop();
13   }
14   while (!st.empty())
15   {
16     cout<<st.top()<<" ";
17     st.pop();
18   }
19   return 0;
20 }
```

11.13.2　表达式括号匹配1（bracket1）

【题目描述】

假设表达式中允许包含两种括号，即圆括号和方括号，其嵌套的顺序任意，例如( )或[([ ]
[ ])]等均为正确的匹配，[(])、(或(()))均为错误的匹配。

现输入一个只包含圆括号和方括号的字符串，试判断字符串中的括号是否为正确的匹配。

【输入格式】

输入一行字符串（字符串中字符的个数小于255）。

【输出格式】

如果字符串中的括号为正确的匹配则输出“OK”，否则输出“Wrong”。

【输入样例】

[(])

【输出样例】

Wrong

参考程序如下所示。

```
//表达式括号匹配
#include <bits/stdc++.h>
using namespace std;

int main()
{
  string s;
  stack <char> st;
  cin>>s;
  for (int i=0; i<s.size(); i++)
  {
    if (s[i]=='['||s[i]=='(')
      st.push(s[i]);
    if (s[i]==']')
      if (st.top()=='[')
        st.pop();
      else
        printf("Wrong\n"),exit(0);
    if (s[i]==')')
      if (st.top()=='(')
        st.pop();
      else
        printf("Wrong\n"),exit(0);
  }
  printf("%s\n",st.empty()?"OK":"Wrong");
  return 0;
}
```

11.13.3　表达式括号匹配2（bracket2）

【题目描述】

表达式由数字、英文字母（小写）、运算符（“+”“-”“*”“/”）及圆括号构成，以“@”作为表达式的结束符。请编写一个程序检查表达式中的左、右圆括号是否匹配。

【输入格式】

输入一行表达式，表达式长度小于255，左圆括号少于20个。

【输出格式】

表达式左、右圆括号若匹配则返回“YES”，否则返回“NO”。

【输入样例1】

```
2*(x+y)/(1-x)@
```

【输出样例1】

```
YES
```

【输入样例2】

```
(25+x)*(a*(a+b+b))@
```

【输出样例2】

```
NO
```

11.13.4　表达式求值（expr）

【题目描述】

给定一个只包含加法运算和乘法运算的表达式，请编程计算表达式的值。

【输入格式】

输入一行表达式，保证只有0～9、“+”及“*”这12种字符，且没有括号，所有参与运算的数字均为0～2^{31-1}的整数。

【输出格式】

输出只有一行，包含一个整数，表示表达式的值。注意：当结果长度多于4位时，请只输出最后4位，前导0不输出。

【输入样例1】

1+1234567890*1

【输出样例1】

7891

【输入样例2】

1+1000000003*1

【输出样例2】

4

【样例说明】

样例1计算的结果为1 234 567 891，输出后4位，即7 891。

样例2计算的结果为1 000 000 004，输出后4位，即4。

【数据范围】

对于100%的数据，$0 \leqslant$ 表达式中加法运算符和乘法运算符的总数 $\leqslant 100\,000$。

【算法分析】

对于一般的（甚至更复杂的）表达式求值的题目，一般采用如下方法。

（1）设两个栈：符号栈与数字栈。

（2）扫描表达式，遇到数字则进栈，遇到符号则转至（3），扫描完毕则转至（4）。

（3）遇到符号：首先将符号栈中优先级不大于当前符号优先级的都弹出，每次取出数字栈顶的两个元素，求值后压入数字栈；然后将当前符号压入符号栈。转至（2）。

（4）扫描完毕后数字栈顶对应的便是所求的值。本题只要求输出最后4位，由于只包含“+”和“*”，因此可以边计算边取余。

11.14　queue队列容器



扫码看视频

如图11.9所示，queue队列容器是一个线性存储表，在头文件<queue>中声明。与后进先出的堆栈不同，元素数据的插入在表的一端进行，元素数据的删除在表的另一端进行，从而构成了一个“先进先出”（First In First Out） 表。插入一端称为队尾，删除一端称为队首。

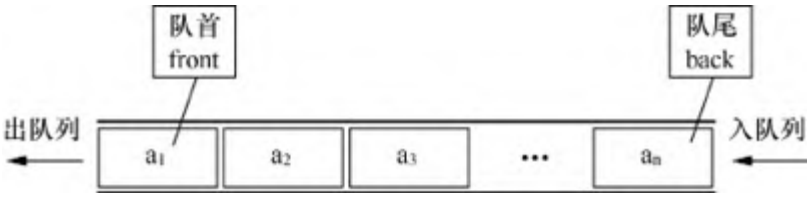


图11.9

C++语言STL 的queue队列容器提供队列的基本操作，参考程序如下所示。

```
 1 //queue例程
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   queue<int> q;
 8   q.push(3);                                          //入队
 9   q.push(5);
10   cout<<"元素个数为："<<q.size()<<endl;
11   cout<<q.back();                                     //取队尾元素
12   while (!q.empty())                                  //当队列不为
空时
13   {
14     cout<<q.front()<<endl;                            //输出队首元
素
15     q.pop();                                          //出队
16   }
17   return 0;
18 }
```

11.14.1 海港（port）

【题目描述】

小光对到达海港的船只非常感兴趣，他按照时间记录了到达海港的每一艘船的情况。对于第i艘到达的船，他记录了这艘船到达海港的时间t_i（单位：秒）、船上的乘客数k_i，以及每名乘客的国籍$x(i,1),x(i,2)$,…,$x(i,k)$。

小光统计了n艘船的信息，希望你帮忙计算出以每一艘船到达时间为止的24小时（24小时＝86 400秒）内所有乘船到达的乘客来自多少个不同的国家。

形式化地讲，你需要计算n条信息。对于输出的第i条信息，你需要统计满足$t_i - 86\,400 < t_p \leqslant t_i$的船只$p$，在所有的$x(p, j)$中，总共有多少个不同的数。

【输入格式】

第一行输入一个正整数n，表示小光统计了 n艘船的信息。

接下来为n行，每行描述一艘船的信息：前两个整数t_I和k_I分别表示这艘船到达海港的时间和船上的乘客数，后面的k_I个整数$x(i, j)$表示船上乘客的国籍。

保证输入的t_I是递增的，表示从小光第一次上班开始计时，这艘船在第 t_I 秒到达海港。保证$1 \leqslant n \leqslant 10^5$，$\sum k_i \leqslant 3 \times 10^5$，$1 \leqslant x(i,j) \leqslant 10^5$，$1 \leqslant t_{(i-1)} \leqslant t_i \leqslant 10^9$。

其中Σk_I表示所有的k_I的和。

【输出格式】

输出n行，第i行输出一个整数，表示第i艘船到达后的统计信息。

【输入样例1】

3

1 4 4 1 2 2

2 2 2 3

10 1 3

【输出样例1】

3

4

4

【样例说明1】

第1艘船在第1秒到达海港，最近24小时到达的船是第1艘船，共有4名乘客， 分别来自国家4、1、2、2，共来自3个不同的国家。

第2艘船在第2秒到达海港，最近24小时到达的船是第1艘船和第2艘船，共有4＋2＝6
名乘客，分别是来自国家4、1、2、2、2、3，共来自4个不同的国家。

第3艘船在第10秒到达海港，最近24小时到达的船是第1、第2及第3艘船，共有4＋2＋1＝7名乘客，分别来自国家4、1、2、2、2、3、3，共来自4个不同的国家。

【输入样例2】

```
4
1 4 1 2 2 3
3 2 2 3
86401 2 3 4
86402 1 5
```

【输出样例2】

```
3
3
3
4
```

【样例说明2】

第1艘船在第1秒到达海港，最近24小时到达的船是第1艘船，共有4名乘客，分别来自国家1、2、2、3，共来自3个不同的国家。

第2艘船在第3秒到达海港，最近24小时到达的船是第1艘船和第2艘船，共有4＋2＝6
名乘客，分别来自国家1、2、2、3、2、3，共来自3个不同的国家。

第3艘船在第86 401秒到达海港，最近24小时到达的船是第2艘船和第3艘船，共有
2＋2＝4名乘客，分别来自国家2、3、3、4，共来自3个不同的国家。

第4艘船在第86 402秒到达海港，最近24小时到达的船是第2、第3及第4艘船，共有2＋2＋1＝5名乘客，分别来自国家2、3、3、4、5，共来自4个不同的国家。

11.14.2 团体队列（queue）

【题目描述】

有t个团队的人正在排长队，每有一个新来的人时，他会从队首开始向后搜寻，如果发现有队友正在排队，他就会插队到他队友的身后；如果没有发现任何一个队友排队，他就只好站在长队的队尾。

输入每个团队中所有队员的编号，要求支持如下3种指令。（题目来源：UVA 540）

ENQUEUE x：编号为x的人进入长队。

DEQUEUE：长队队首的人出队。

STOP：停止模拟。

对于每个DEQUEUE指令，输出出队的人的编号。

【输入格式】

有多组测试数据，每组数据的第一行为一个整数t（$1 \leqslant t \leqslant 1000$），表示有$t$个团队。随后$t$行描述每一个团队，即每一个团队有一个表示该团队人数的整数n（可能多达1 000个）和n个整数编号（范围为0～999 999）。

随后是指令列表（可能多达200 000条指令），有如题所示3种不同的指令。

当t是0时，输入终止。

【输出格式】

对应每组测试数据，首先输出一行“Scenario #k”，其中k表示第几次测试；然后每一个DEQUEUE指令打印出队的人的编号（单独占一行）。

在每一组测试数据之后打印一空行，即使这组测试数据是最后一组。

【输入样例】

```
2
3 101 102 103
3 201 202 203
ENQUEUE 101
ENQUEUE 201
ENQUEUE 102
ENQUEUE 202
ENQUEUE 103
ENQUEUE 203
DEQUEUE
DEQUEUE
DEQUEUE
```

STOP

2

5 259001 259002 259003 259004 259005

6 260001 260002 260003 260004 260005 260006

ENQUEUE 259001

ENQUEUE 260001

ENQUEUE 259002

ENQUEUE 259003

ENQUEUE 259004

ENQUEUE 259005

DEQUEUE

DEQUEUE

ENQUEUE 260002

ENQUEUE 260003

DEQUEUE

STOP

0

【输出样例】

Scenario #1

101

102

103

Scenario #2

259001

259002

259003

11.15　priority_queue优先队列容器



扫码看视频

priority_queue优先队列容器在头文件<queue>中声明，与队列容器一样，只能从队尾插入元素，从队首删除元素。其一般形式为priority_queue<Type, Container, Functional>，其中Type为数据类型，Container为保存数据的容器，Functional为元素比较方式。

Container必须是用数组实现的容器，例如vector、deque等，但不能用 list，STL里面默认用的是vector。如果后面两个参数采用默认设置，优先队列容器就是大顶堆（降序），队首元素最大。可以重载运算符来重新定义比较规则。

默认降序输出的参考程序如下所示。

```
 1 //priority_queue例程
 2 #include <bits/stdc++.h>
 3 using namespace std;
 4
 5 int main()
 6 {
 7   priority_queue<int> q;
 8   q.push(93);                                    //入队，插入新元素
 9   q.push(5);
10   q.push(12);
11   cout<<"元素个数为: "<<q.size()<<endl;
12   while (!q.empty())                            //当队列不为空时
13   {
14     cout<<q.top()<<endl;                        //取队首元素
15     q.pop();                                    //出队要先判断是否为空
16   }
17   return 0;
18 }
```



从小到大排列，让小的元素优先出队的基本写法如下。

```
priority_queue<int,vector<int>,greater<int> >q;
```

从大到小排列，让大的元素优先出队的基本写法如下。

```
priority_queue<int,vector<int>,less<int> >q;
```

结构体中重载运算符的参考程序如下所示。

```
//结构体中重载运算符
#include <bits/stdc++.h>
using namespace std;

struct Info
{
  string name;
  float score;
  bool operator <(const Info &a) const
  {
    return a.score<score;       //按score由小到大排列用"<",否则用">"
  }
} ;

int main()
{
  priority_queue<Info> pq;
  Info info;
  info.name="Alice";
  info.score=98;
  pq.push(info);
  info.name="Jone";
  info.score=92;
  pq.push(info);
  info.name="Kate";
  info.score=95.5;
```

```
27    pq.push(info);
28    while (pq.empty()!=true)
29    {
30      cout<<pq.top().name<<": "<<pq.top().score<<endl;
31      pq.pop();
32    }
33    return 0;
34  }
```

11.15.1　推销员（salesman）

【题目描述】

螺丝街是一条死胡同，出口与入口是同一个，街道的一侧是围墙，另一侧是住户。螺丝街一共有N家住户，第i家住户到入口的距离为S_i米。由于同一栋房子里可以有多家住户，因此可能有多家住户与入口的距离相等。小光会从入口进入，依次向螺丝街的X家住户推销产品，然后按原路走出去。

小光每走1米就会积累1点疲劳值，向第i家住户推销产品会积累A_i点疲劳值。小光想知道，对于不同的X家住户，在不走多余的路的前提下，他最多会积累多少点疲劳值。

【输入格式】

第1行有一个正整数N，表示螺丝街住户的数量。

第2行有N个正整数，其中第i个正整数S_i表示第i家住户到入口的距离。数据保证$S_1 \leqslant S_2 \leqslant \cdots \leqslant S_n < 10^8$。

第3行有N个正整数，其中第i个正整数A_i表示向第i家住户推销产品会积累的疲劳值。数据保证$A_i < 10^3$。

【输出格式】

输出N行，每行一个正整数，第i行的正整数表示当$X=i$时，小光最多积累的疲劳值。

【输入样例1】

5

1 2 3 4 5

1 2 3 4 5

【输出样例1】

15

19

22

24

25

【样例说明1】

X＝1：向住户5推销，往返走路的疲劳值为5＋5，推销的疲劳值为5，总疲劳值为5+5+5=15。

X＝2：向住户4、5推销，往返走路的疲劳值为5＋5，推销的疲劳值为4＋5，总疲劳值为5＋5＋4＋5＝19。

X＝3：向住户3、4、5推销，往返走路的疲劳值为5＋5，推销的疲劳值为3＋4＋5，总疲劳值为5＋5＋3＋4＋5＝22。

X＝4：向住户2、3、4、5推销，往返走路的疲劳值为5＋5，推销的疲劳值为2＋3＋4＋5，总疲劳值为5＋5＋2＋3＋4＋5＝24。

X＝5：向住户1、2、3、4、5推销，往返走路的疲劳值为5＋5，推销的疲劳值为1＋2＋3＋4＋5，总疲劳值为5＋5＋1＋2＋3＋4＋5＝25。

【输入样例2】

5

1 2 2 4 5

5 4 3 4 1

【输出样例2】

12

17

21

24

27

【样例说明2】

X＝1：向住户4推销，往返走路的疲劳值为4＋4，推销的疲劳值为4，总疲劳值为4＋4＋4＝12。

X＝2：向住户1、4推销，往返走路的疲劳值为4＋4，推销的疲劳值为5＋4，总疲劳值为4＋4＋5＋4＝17。

X＝3：向住户1、2、4推销，往返走路的疲劳值为4＋4，推销的疲劳值为5＋4＋4，总疲劳值为4＋4＋5＋4＋4＝21。

X＝4：向住户1、2、3、4推销，往返走路的疲劳值为4＋4，推销的疲劳值为5＋4＋3＋4，总疲劳值为4＋4＋5＋4＋3＋4＝24。或者向住户1、2、4、5推销，往返走路的疲劳值为5＋5，推销的疲劳值为5＋4＋4＋1，总疲劳值为5＋5＋5＋4＋4＋1＝24。

X=5：向住户1、2、3、4、5推销，往返走路的疲劳值为5＋5，推销的疲劳值为5＋4＋3＋4＋1，总疲劳值为5＋5＋5＋4＋3＋4＋1＝27。

【数据范围】

对于100%的数据，$1 \leqslant N \leqslant 100\,000$。

【算法分析】

使用贪心算法，贪心算法是指在对问题求解时，总是做出在当前看来最好的选择。

首先选择疲劳值最大的一个点，即$s[i]\times 2+A[i]$值最大的点，将这个点记为now，用优先队列保存now左边的点的疲劳值。每次选择点时，枚举now右边的点j，找出now右边疲劳值最大的点记为Rmax（注意这里的距离应该为now到j的距离），与优先队列的最大值Lmax比较，取最大值的点为新的点now。如果左边的点的疲劳值大，则将该点弹出优先队列；如果右边的点的疲劳值大，则将原先now点到新now点之间的点压入优先队列。时间复杂度在最好的情况下是$O(n\log n)$，在最坏情况下是$O(n^2)$。具体情况如图11.10所示。

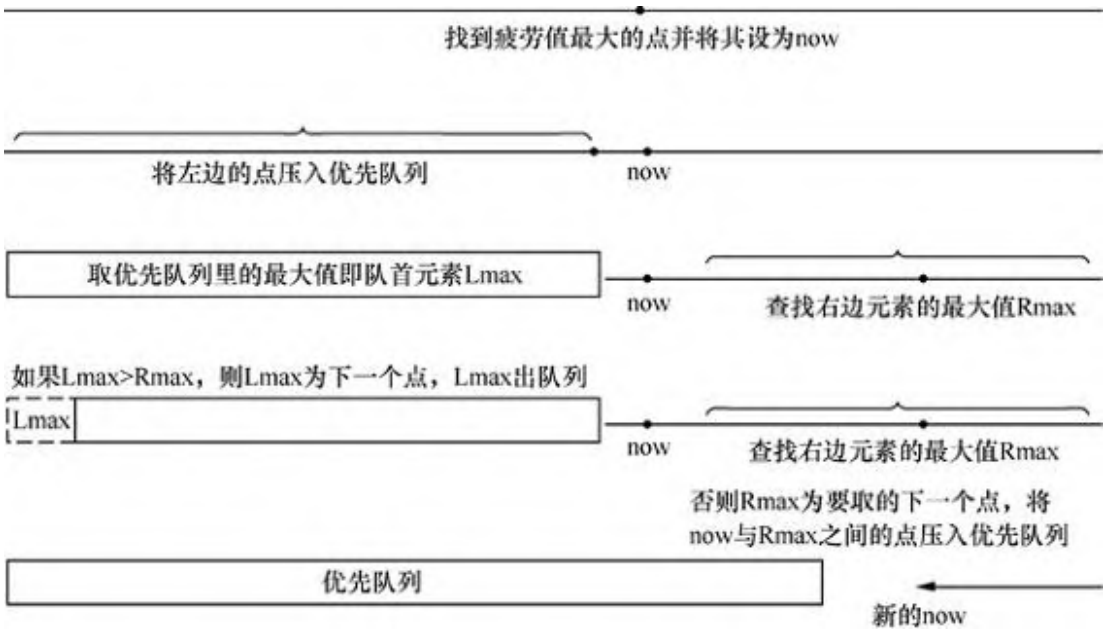


图11.10

11.15.2　有序表最小和（MinSum）

【题目描述】

有两个长度为n的有序表A和B，在A和B中各任取一个元素，可以得到n^2个和，求这些和中最小的n个。

【输入格式】

第一行为一个正整数n（$n \leqslant 400\,000$）。

随后两行分别为单调递增的有序表A和B，表中整数大小在超长整数范围内。

【输出格式】

输出n个单调递增的整数，数据保证在超长整数范围内。

【输入样例】

3

1 2 5

2 4 7

【输出样例】

3 4 5

附录A　ASCII 对照表

ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
0	NUL	32	（空格）	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	”	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	’	71	G	103	g
8	BS	40	（	72	H	104	h

ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s

ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~

ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
31	US	63	?	95	_	127	DEL

附录B　C++语言的关键字

asm	double	new	switch
auto	else	operator	template
break	enum	private	this
case	extern	protected	throw
catch	float	public	try
char	for	return	typedef
class	friend	register	union
const	goto	short	unsigned
continue	If	signed	virtual
default	inline	sizeof	void
delete	Int	static	volatile

asm	double	new	switch
do	long	struck	while

加粗的关键字为C++语言所特有，其余关键字为C与C++语言共有。

“标识符”或“符号”是程序中提供的变量、类型、函数及标号的名称。标识符在拼写和大小写上必须不同于关键字。不能使用关键字作为标识符，它们是为了特定用途而被保留的。

附录C　C++语言运算符及其优先级

优先级	运算符	名称或含义	使用形式	结合方向	说明
1	[]	数组下标	数组名[常量表达式]	左到右	
	()	圆括号	(表达式)/函数名(形参表)		
	.	成员选择（对象）	对象.成员名		
	->	成员选择（指针）	对象指针->成员名		
2	-	负号运算符	-表达式	右到左	单目运算符
	(类型)	强制类型转换	(数据类型)表达式		
	++	自增运算符	++变量名/变量名++		单目运算符
	--	自减运算符	--变量名/变量名--		单目运算符
	*	取值运算符	*指针变量		单目运算符
	&	取地址运算符	&变量名		单目运算符
	!	逻辑非运算符	!表达式		单目运算符
	~	取反运算符	~表达式		单目运算符
	sizeof	长度运算符	sizeof(表达式)		
3	/	除	表达式/表达式	左到右	双目运算符
	*	乘	表达式*表达式		双目运算符
	%	求余数（取模）	整型表达式/整型表达式		双目运算符
4	+	加	表达式+表达式	左到右	双目运算符
	-	减	表达式-表达式		双目运算符
5	<<	左移运算符	变量<<表达式	左到右	双目运算符
	>>	右移运算符	变量>>表达式		双目运算符
6	>	大于	表达式>表达式	左到右	双目运算符
	>=	大于等于	表达式>=表达式		双目运算符
	<	小于	表达式<表达式		双目运算符

	<=	小于等于	表达式<=表达式		双目运算符
7	==	等于	表达式==表达式	左到右	双目运算符
	!=	不等于	表达式!=表达式		双目运算符
8	&	按位与运算符	表达式&表达式	左到右	双目运算符
9	^	按位异或运算符	表达式^表达式	左到右	双目运算符
10	|	按位或运算符	表达式|表达式	左到右	双目运算符
11	&&	逻辑与运算符	表达式&&表达式	左到右	双目运算符
12	||	逻辑或运算符	表达式||表达式	左到右	双目运算符
13	?:	条件运算符	表达式1? 表达式2: 表达式3	右到左	三目运算符
14	=	赋值运算符	变量=表达式	右到左	
	/=	除后赋值	变量/=表达式		
	*=	乘后赋值	变量*=表达式		
	%=	取模后赋值	变量%=表达式		
	+=	加后赋值	变量+=表达式		
	-=	减后赋值	变量-=表达式		
	<<=	左移后赋值	变量<<=表达式		
	>>=	右移后赋值	变量>>=表达式		
	&=	按位与后赋值	变量&=表达式		
	^=	按位异或后赋值	变量^=表达式		
	|=	按位或后赋值	变量|=表达式		
15	,	逗号运算符	表达式,表达式,…	左到右	从左向右顺序运算

说明：同一优先级的运算符，运算次序由结合方向所决定。

附录D　常用函数库

C语言

```
#include <assert.h>          //设定插入点
#include <ctype.h>           //字符处理
#include <errno.h>           //定义错误码
#include <float.h>           //浮点数处理
#include <fstream.h>         //文件输入/输出
#include <iomanip.h>        //参数化输入/输出
#include <iostream.h>       //数据流输入/输出
#include <limits.h>          //定义各种数据类型的最值常量
#include <locale.h>          //定义本地化函数
#include <math.h>           //定义数学函数
#include <stdio.h>           //定义输入/输出函数
#include <stdlib.h>          //定义杂项函数和内存分配函数
```

#include <string.h>　　//字符串处理

#include <strstrea.h>　　//基于数组的输入/输出

#include <time.h>　　//定义关于时间的函数

#include <wchar.h>　　//宽字符处理和输入/输出

#include <wctype.h>　　//宽字符分类

标准 C++语言

#include <algorithm>　　//STL 通用算法

#include <bitset>　　//STL 位集容器

#include <cctype>　　//字符处理

#include <cerrno>　　//定义错误码

#include <clocale>　　//定义本地化函数

#include <cmath>　　//定义数学函数

#include <complex>　　//复数类

#include <cstdio>　　//定义输入/输出函数

#include <cstdlib>　　//定义杂项函数和内存分配函数

```
#include <cstring>          //字符串处理
#include <ctime>            //定义关于时间的函数
#include <deque>            //STL 双端队列容器
#include <exception>        //异常处理类
#include <fstream>          //文件输入/输出
#include <functional>       //STL 定义运算函数（代替运算符）
#include <limits>           //定义各种数据类型的最值常量
#include <list>             //STL 双向链表容器
#include <map>              //STL 映照容器
#include <iomanip>          //参数化输入/输出
#include <ios>              //基本输入/输出支持
#include <iosfwd>           //输入/输出系统使用的前置声明
#include <iostream>         //数据流输入/输出
```

```
#include <istream>          //基本输入流
#include <ostream>          //基本输出流
#include <queue>            //STL 队列容器
#include <set>              //STL 集合容器
#include <sstream>          //基于字符串的流
#include <stack>            //STL 堆栈容器
#include <stdexcept>        //标准异常类
#include <streambuf>       //底层输入/输出支持
#include <string>           //字符串类
#include <utility>          //STL 通用模板类
#include <vector>          //STL 向量容器
#include <cwchar>         //宽字符处理和输入/输出
#include <cwctype>        //宽字符分类
```

附录E　常用函数

分类函数，所在函数库为ctype.h

函数原型	函数功能
int isalpha(int ch)	若ch是字母（A~Z，a~z）则返回非0值，否则返回0
int isalnum(int ch)	若ch是字母（A~Z，a~z)或数字（0~9）则返回非0值，否则返回0
int isascii(int ch)	若ch是字符（ASCII中的0~127）则返回非0值，否则返回0
int isdigit(int ch)	若ch是数字（0~9）则返回非0值，否则返回0
int islower(int ch)	若ch是小写字母（a~z）则返回非0值，否则返回0
int isupper(int ch)	若ch是大写字母（A~Z）则返回非0值，否则返回0

函数原型	函数功能
int isxdigit(int ch)	若ch是十六进制数（0~9，A~F，a~f）则返回非0值，否则返回0
int tolower(int ch)	若ch是大写字母（A~Z）则返回相应的小写字母（a~z）
int toupper(int ch)	若ch是小写字母（a~z）则返回相应的大写字母（A~Z）

数学函数，所在函数库为math.h、stdlib.h、string.h、float.h

函数原型	函数功能
int abs(int i)	返回整数i的绝对值
double fabs(double x)	返回双精度浮点数x的绝对值
long labs(long n)	返回长整数n的绝对值
double exp(double x)	返回指数函数e^x的值

函数原型	函数功能
double log(double x)	返回ln(x)的值
double log10(double x)	返回x的常用对数（基数为10的对数）
double pow(double x,double y)	返回x^y的值
double pow10(int p)	返回10^p的值
double sqrt(double x)	返回x的平方根
double acos(double x)	返回x的反余弦arccos (x)值，x为弧度
double asin(double x)	返回x的反正弦arcsin (x)值，x为弧度
double atan(double x)	返回x的反正切arctan (x)值，x为弧度
double cos(double x)	返回x的余弦cos(x)值，x为弧度
double sin(double x)	返回x的正弦sin(x)值，x为弧度
double tan(double x)	返回x的正切tan(x)值，x为弧度

函数原型	函数功能
double cosh(double x)	返回x的双曲余弦cosh(x)值，x为弧度
double sinh(double x)	返回x的双曲正弦sinh(x)值，x为弧度
double tanh(double x)	返回x的双曲正切tanh(x)值，x为弧度
double ceil(double x)	返回不小于x的最小整数
double floor(double x)	返回不大于x的最大整数
void srand(unsigned seed)	初始化随机数发生器
Int rand(void)	产生一个随机数并返回这个数
double fmod(double x,double y)	返回浮点数x/y的余数
double atoi(char *nptr)	将字符串nptr转换成整数并返回这个整数
double atol(char *nptr)	将字符串nptr转换成长整数并返回这个整数

函数原型	函数功能
double atof(char *nptr)	将字符串nptr转换成双精度浮点数并返回这个数，错误返回0
int atoi(char *nptr)	将字符串nptr转换成整数并返回这个数，错误返回0
long atol(char *nptr)	将字符串nptr转换成长整数并返回这个数，错误返回0

进程函数，所在函数库为stdlib.h、process.h

函数原型	函数功能
int exec	装入和运行其他程序
void exit(int status)	终止当前程序，关闭所有文件
int system(char *command)	将MSDOS命令command传递给DOS执行

输入/输出子程序，所在函数库为io.h、conio.h、stat.h、dos.h、stdio.h、signal.h

函数原型	函数功能
int getch()	从控制台（键盘）读一个字符，不显示在显示器上
int putch()	向控制台（键盘）写一个字符
int getchar()	从控制台（键盘）读一个字符，显示在显示器上
int putchar()	向控制台（键盘）写一个字符
int scanf(char *format[,argument…])	从控制台读一个字符串，分别对各个参数进行赋值，使用基本输入/输出系统（Basic Input/Output System，BIOS）输出
int sscanf(char*string,char*format[,argument,…])	通过字符串string，分别对各个参数进行赋值
int puts(char *string)	发送字符串string给控制台（显示器），使用BIOS输出

函数原型	函数功能
int printf(char *format[,argument,…])	发送格式化字符串给控制台（显示器），使用BIOS输出
int eof(int *handle)	检查文件是否结束，结束返回1，否则返回0
long filelength(int handle)	返回文件长度，handle为文件号
int close(int handle)	关闭handle所表示的文件处理
FILE *fopen(char *filename,char *type)	打开一个文件filename
FILE *freopen(char *filename,char *type,FILE *stream)	重定向文件
char *fgets(char *string,int n,FILE *stream)	从流stream中读n个字符存入string
int fputs(char *string,FILE *stream)	将字符串string写入流stream
int fread(void*ptr,int size,int nitems,FILE *stream)	从流stream中读入nitems个长度为size的字符串存入ptr

函数原型	函数功能
int fwrite(void*ptr,int size,int nitems,FILE*stream)	向流stream中写入nitems个长度为size的字符串，字符串在ptr中
int fscanf(FILE*stream,char*format [,argument,…])	以格式化形式从流stream中读入一个字符串
int rewind(FILE *stream)	将当前文件指针stream移到文件开头
int fclose(FILE *stream)	关闭一个流，可以是文件或设备（例如LPT1）

string字符串操作函数

函数原型	函数功能
char strcat(char *dest,const char *src)	将字符串src添加到字符串dest末尾
char strchr(const char *s,int c)	检索并返回字符c在字符串s中第一次出现的位置

函数原型	函数功能
int strcmp(const char *s1,const char *s2)	比较字符串s1与s2的大小，并返回s1－s2
char strcpy(char *dest,const char *src)	将字符串src复制到字符串dest
unsigned int strlen(char *str)	返回字符串str的长度
int strncmp(const char *s1,const char *s2,size_t maxlen)	比较字符串s1与s2中的前maxlen个字符
char strncpy(char *dest,const char *src,size_t maxlen)	复制字符串src中的前maxlen个字符到字符串dest
char strrchr(const char *s,int c)	扫描最后出现一个给定字符c的字符串s
char strrev(char *s)	将字符串s中的字符全部颠倒顺序重新排列，并返回排列后的字符串
char strset(char *s,int ch)	将字符串s中的所有字符置于一个给定的字符ch
size_t strspn(const char *s1,const char *s2)	扫描字符串s1，并返回在字符串s1和字符串s2中均有的字符的个数

函数原型	函数功能
char strstr(const char *s1,const char *s2)	扫描字符串s2，并返回第一次出现字符串s1的位置
char strtok(char *s1,const char *s2)	检索字符串s1，字符串s1由字符串s2中定义的定界符所分隔

附录F　如何使用在线评测网站

www.magicoj.com是我们专为编程竞赛爱好者准备的在线评测网站，致力于为编程竞赛爱好者提供清爽、快捷的编程体验。它包含了从入门到省选层次的近万道精选编程题，无论是初学者还是久经沙场的选手，都能从中获益。

目前网站的课程分类有语法和算法入门、竞赛基础算法、动态规划、数据结构基础、编程与数学（见图A-1），分别对应于“编程竞赛宝典”系列书中用到的所有题目。



图A-1

建议读者按照书中编排的题目顺序学习，结合对应的讲解视频，在理解的基础上动手编程，并将代码提交到网站，获得实时反馈，从而形成了一个完整且高效的学习过程，使学习效果更好。

此外，每道题专设了讨论区和题解区（见图A-2），进一步帮助读者解惑并开拓思维。



图A-2