目　录

第一章　最简单的问题与算法

作为趣味程序设计的入门，我们先来涉猎一些简单的例子。通过这些例子，让读者体验一下程序设计的乐趣。

1. 绘制余弦曲线

在屏幕上用“*”显示 0°～360°的余弦函数 cos(x)曲线。

* 问题分析与算法设计

如果在程序中使用数组，这个问题十分简单。但若规定不能使用数组，问题就变得不容易了。

关键在于余弦曲线在 0°～360°的区间内，一行中要显示两个点；而对一般的显示器来说，只能按行输出，即：输出第一行信息后，只能向下一行输出，不能再返回到上一行。为了获得本题要求的图形就必须在一行中一次顺序输出两个“*”。

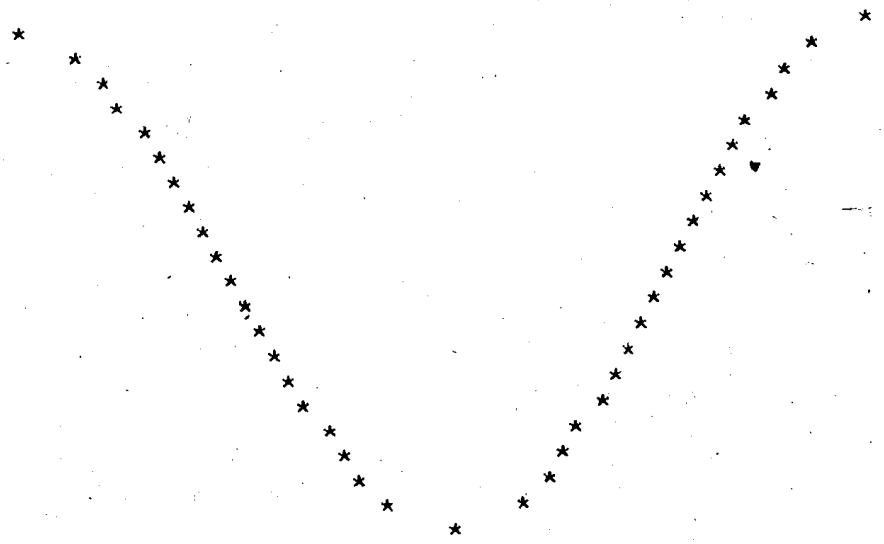
为了同时得到余弦函数 cos(x)图形在一行上的两个点，考虑利用 cos(x)的左右对称性。将屏幕的行方向定义为 x，列方向定义为 y，则 0°～180°的图形与 180°～360°的图形是左右对称的。若定义图形的总宽度为 62 列，计算出 x 行 0°～180°时 y 点的坐标 m，那么在同一行与之对称的 180°～360°的 y 点的坐标就应为 62－m。程序中利用反余弦函数 acos 计算坐标(x,y)的对应关系。

使用这种方法编出的程序短小精练，体现了一定的技巧。

* 程序说明与注释

```
#include <stdio.h>
#include <math.h>
main()
{  double y;
   int x, m;
   for (y=1; y>=-1; y-=0.1) {          /* y 为列方向，值从 1 到－1，步长为 0.1 */
      m=acos(y) * 10;                  /* 计算出 y 对应的弧度 m，乘 10 为图形放大倍数 */
      for (x=1; x<m; x++) printf(" ");
      printf("*");                                   /* 控制打印左侧的 * 号 */
      for (; x<62-m; x++) printf(" ");
      printf("*\n");                          /* 控制打印同一行中对称的右侧 * 号 */
    }
}
```

* 运行结果



* 思考题

如何实现用“*”显示 0°～360°的 sin(x)曲线。

2. 绘制余弦曲线和直线

在屏幕上显示 0°～360°的 cos(x)曲线与直线 f(x)=45*(y−1)+31 的迭加图形。其中 cos(x)图形用“*”表示,f(x)用“+”表示,在两个图形的交点处则用 f(x)图形的符号。

* 问题分析与算法设计

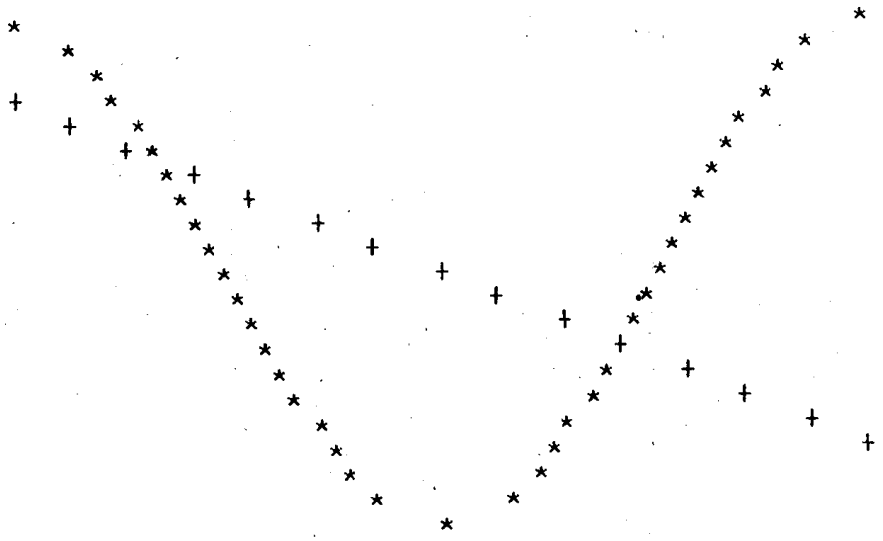
本题可以在上题的基础上进行修改。图形迭加的关键是要在分别计算出同一行中两个图形的列方向点坐标后,正确判断相互的位置关系。为此,可以先判定图形的交点,再分别控制打印两个不同的图形。

* 程序说明与注释

```
#include <stdio.h>
#include <math.h>
main()
{ double y;
  int x, m, n, yy;
  for (yy=0; yy<=20; yy++) {
                                  /* 对于每一个 y 坐标进行计算并在一行中打印图形 */
    y = 0.1 * yy;                                    /* y:屏幕行方向坐标 */
    m = acos(1-y) * 10;          /* m:cos(x)曲线上 y 点对应的屏幕列坐标 */
    n = 45 * (y-1)+31;                  /* n:直线上 y 点对应的列坐标 */
    for (x=0; x<=62; x++)                        /* x:屏幕列方向坐标 */
      if ( x==m && x==n )                printf("+");
                                          /* 直线与 cos(x)相交时打 '+' */
      else if ( x==n )                   printf("+");
                                            /* 打印不相交时的直线图形 */
```

```
    else if (x==m || x==62-m) printf("*");   /* 打印不相交时的 cos(x)图形 */
    else                      printf(" ");   /* 其它情况打印空格 */
     printf("\n");
   }
 }
```

* 运行结果



* 思考题

如何实现 sin(x)曲线与 cos(x)曲线图形的同时显示。

3. 绘制圆

在屏幕上用“*”画一个空心的圆。

* 问题分析与算法设计

打印圆可利用图形的左右对称性。根据圆的方程：

$$R \times R = X \times X + Y \times Y$$

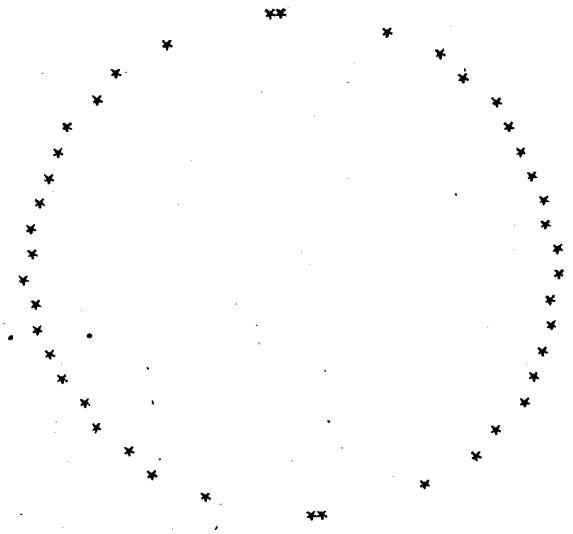
可以计算出圆上每一点行和列的对应关系。

* 程序说明与注释

```
#include <stdio.h>
#include <math.h>
main()
{ double y;
  int x, m;
  for (y=10; y>=-10; y--) {                    /* 圆的半径为 10 */
  m = 2.5 * sqrt(100-y*y);
                  /* 计算行 y 对应的列坐标 m。2.5 是屏幕纵横比调节系数,因为屏幕的
                     行距大于列距,不进行调节显示出来的将是椭圆 */
  for (x=1; x<30-m; x++) printf(" ");          /* 图形左侧空白控制 */
```

```
        printf("*");                                    /* 圆的左侧 */
        for (; x<30+m; x++) printf(" ");                /* 图形的空心部分控制 */
        printf("*\n");                                  /* 圆的右侧 */
    }
}
```

* 运行结果



* 思考题

实现函数 $y=x^2$ 的图形与圆的图形的叠加显示。

4. 歌星大奖赛

在歌星大奖赛中，有 10 个评委为参赛的选手打分，分数为 1 到 100 分。选手最后得分为：去掉一个最高分和一个最低分后其余 8 个分数的平均值。请编写一个程序实现。

* 问题分析与算法设计

这个问题的算法十分简单，但要注意在程序中判断最大、最小值的变量是如何赋初值的。

* 程序说明与注释

```
main()
{ int integer, i, max, min, sum;
    max=-32768;          /* 先假设当前的最大值 max 为 C 语言整型数的最小值 */
    min=32767;           /* 先假设当前的最小值 min 为 C 语言整型数的最大值 */
    sum=0;                                  /* 将求累加和变量的初值置为 0 */
    for (i=1; i<=10; i++) {
        printf("Input number %d=", i);
        scanf("%d", &integer);                          /* 输入评委的评分 */
        sum += integer;                                 /* 计算总分 */
        if (integer>max) max=integer;       /* 通过比较筛选出其中的最高分 */
        if (integer<min) min=integer;       /* 通过比较筛选出其中的最低分 */
    }
    printf ("Canceled max score: %d\nCanceled min score: %d\n", max, min);
```

```
    printf (" Average score: %d\n", (sum-max-min)/8);          /* 输出结果 */
  }
```

* 运行结果

```
    Input number 1=90
    Input number 2=91
    Input number 3=93
    Input number 4=94
    Input number 5=90
    Input number 6=99
    Input number 7=97
    Input number 8=92
    Input number 9=91
    Input number 10=95
    Canceled max score: 99
    Canceled min score: 90
        Average score: 92
```

* 思考题

题目条件不变，但考虑同时对评委的评分进行裁判，即在10个评委中找出最公平(即评分最接近平均分)和最不公平(即与平均分的差距最大)的评委，程序应怎样实现？

5. 求最大约数

问555555的约数中最大的三位数是多少？

* 问题分析与算法设计

根据约数的定义，对于一个整数N，除去1和它自身之外，凡能整除N的数即为N的约数。因此，最简单的方法是用2到N-1之间的所有数去除N，即可求出N的全部约数。本题只要求取约数中最大的三位数，则其取值范围可限制在100到999之间。

* 程序说明与注释

```
main()
{ long i;                              /* 使用长整型变量，以免超出整数的表示范围 */
    int j;
    printf("Please input number:");
    scanf ("%ld", &i);
    for (j=999; j>=100; j--)
                                  /* 所求的约数的可能取值是从999到100，j从大到小 */
      if (i%j==0) {                                  /* 若是约数，则输出结果 */
        printf("The max factor with 3 digits in %ld is: %d. \n", i, j);
        break;
      }
}
```

* 运行结果

输入:555555

输出:The max factor with 3 digits in 555555 is: 777.

6. 高次方数的尾数

求13的13次方的最后三位数。

* 问题分析与算法设计

解本题最直接的方法是:将13累乘13次后截取最后三位即可。

但是由于计算机所能表示的整数范围有限,用这种"正确"的算法不可能得到正确的结果。事实上,题目仅要求后三位的值,完全没有必要求13的13次方的完整结果。

研究乘法的规律会发现:乘积的最后三位的值只与乘数和被乘数的后三位有关,与乘数和被乘数的高位无关。利用这一规律,可以大大简化程序。

* 程序说明与注释

```
main()
{ int i, x, y, last=1;          /* 变量last保存求X的Y次方过程中的部分积的后三位 */
    printf("Input X and Y (X**Y):");
    scanf ("%d**%d", &x, &y);
    for (i=1; i<=y; i++)                                    /* X自乘Y次 */
       last = last * x %1000;       /* 将last乘X后对1000取模,即求积的后三位 */
    printf("The last 3 digits of %d**%d is: %d\n", x, y, last%1000);
}                                                           /* 打印结果 */
```

* 运行结果

```
Input X and Y (X**Y):13**13
The last 3 digits of 13**13 is: 253
Input X and Y (X**Y):13**20
The last 3 digits of 13**20 is: 801
```

7. 阶乘尾数零的个数

100! 的末尾有多少个零?

* 问题分析与算法设计

可以设想:先求出100!的值,然后数一下末尾有多少个零。事实上,与上题一样,由于计算机所能表示的整数范围有限,这是不可能的。

为了解决这个问题,必须首先从数学上分析在100! 结果值的末尾产生零的条件。不难看出:一个整数若含有一个因子5则必然会在求100!时产生一个零。因此问题转化为求1到100这100个整数中包含了多少个因子5。若整数N能被25整除,则N包含2个因子5;若整数N能被5整除,则N包含1个因子5。

* 程序说明与注释

```
main()
{ int a, count=0;
```

```
    for (a=5; a<=100; a+=5) {        /* 循环从 5 开始,以 5 的倍数为步长,考察整数 */
        count++;                                    /* 若为 5 的倍数,计数器加 1 */
        if ( ! (a%25) ) count++;                    /* 若为 25 的倍数,计数器再加 1 */
    }
    printf("The number of 0 in the end of 100! is: %d.\n", count);     /* 打印结果 */
}
```

* 运行结果

The number of 0 in the end of 100! is: 24.

* 问题的进一步讨论

本题的求解程序是正确的,但存在明显的缺点。程序中判断整数 N 包含多少个因子 5 的方法是与题目中 100 有关的,若题目中的 100 改为 1000,则就要修改程序中求因子 5 的数目的算法了。

* 思考题

修改程序中求因子 5 的数目的算法,使程序可以求出任意 N! 的末尾有多少个零。

8. 借书方案知多少

小明有五本新书,要借给 A、B、C 三位小朋友,若每人每次只能借一本,则可以有多少种不同的借法?

* 问题分析与算法设计

本问题实际上是一个排列问题,即求从 5 个中取 3 个进行排列的方法的总数。首先对五本书从 1 至 5 进行编号,然后使用穷举的方法,假设三个人分别借这五本书中的一本,当三个人所借的书的编号都不相同时,就是满足题意的一种借阅方法。

* 程序说明与注释

```
main()
{int a, b, c, count=0;
printf("There are different methods for XM to distribute books to 3 readers:\n");
for (a=1; a<=5; a++)                    /* 穷举第一个人借 5 本书中的 1 本的全部情况 */
    for (b=1; b<=5; b++)                /* 穷举第二个人借 5 本书中的 1 本的全部情况 */
        for (c=1; a! =b && c<=5; c++)
            /* 当前两个人借不同的书时,穷举第三个人借 5 本书中的 1 本的全部情况 */
          if ( c! =a && c! =b )          /* 判断第三人与前两个人借的书是否不同 */
            printf( count%8 ? "%2d:%d,%d,%d " : "%2d:%d,%d,%d\n" ,
                                ++count,a, b, c);       /* 打印可能的借阅方法 */
}
```

* 运行结果

There are different methods for XM to distribute books to 3 readers:

1:1,2,3	2:1,2,4	3:1,2,5	4:1,3,2	5:1,3,4
6:1,3,5	7:1,4,2	8:1,4,3	9:1,4,5	10:1,5,2
11:1,5,3	12:1,5,4	13:2,1,3	14:2,1,4	15:2,1,5

16:2,3,1	17:2,3,4	18:2,3,5	19:2,4,1	20:2,4,3
21:2,4,5	22:2,5,1	23:2,5,3	24:2,5,4	25:3,1,2
26:3,1,4	27:3,1,5	28:3,2,1	29:3,2,4	30:3,2,5
31:3,4,1	32:3,4,2	33:3,4,5	34:3,5,1	35:3,5,2
36:3,5,4	37:4,1,2	38:4,1,3	39:4,1,5	40:4,2,1
41:4,2,3	42:4,2,5	43:4,3,1	44:4,3,2	45:4,3,5
46:4,5,1	47:4,5,2	48:4,5,3	49:5,1,2	50:5,1,3
51:5,1,4	52:5,2,1	53:5,2,3	54:5,2,4	55:5,3,1
56:5,3,2	57:5,3,4	58:5,4,1	59:5,4,2	60:5,4,3

9. 杨辉三角形

在屏幕上显示杨辉三角形：

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        ..... ..... ..... ..... ..... ..... .....
```

* 问题分析与算法设计

杨辉三角形中的数，正是(x+y)的N次方幂展开式中各项的系数。本题作为程序设计中具有代表性的题目，求解的方法很多，这里仅给出一种。

从杨辉三角形的特点出发，可以总结出：

①第N行有N+1个值(设起始行为第0行)；

②对于第N行的第J个值：(N>=2)

当J=1或J=N+1时：　其值为1；

当J! =1且J! =N+1时：　其值为第N-1行的第J-1个值与第N-1行第J个值之和；

将这些特点提炼成数学公式可表示为：

$$c(x,y)=\begin{cases}1 & x=1 \text{ 或 } x=N+1\\ c(x-1,y-1)+c(x-1,y) & \text{其它}\end{cases}$$

本程序就是根据以上递归的数学表达式编制的。

* 程序说明与注释

```
main()
{ int i,j,n=13;
```

```
  printf("N=");
  while (n>12)
    scanf("%d", &n);                    /* 控制输入正确的值以保证屏幕显示的图形正确 */
  for (i=0; i<=n; i++) {                                    /* 控制输出 N 行 */
    for (j=0; j<12-i; j++) printf(" ");          /* 控制输出第 i 行前面的空格 */
    for (j=1; j<i+2; j++) printf("%6d", c(i,j));       /* 输出第 i 行的第 j 个值 */
    printf("\n");
   }
}
int c(x,y)                                          /* 求杨辉三角形中第 x 行第 y 列的值 */
  int x, y;
{ int z;
  if ( (y==1) || (y==x+1) ) return(1);
                                          /* 若为 x 行的第 1 或第 x+1 列,则输出 1 */
  z = c(x-1,y-1) + c(x-1,y);
                                  /* 否则,其值为前一行中第 y-1 列与第 y 列值之和 */
  return(z);
}
```

* 运行结果

输入:N=12

输出:

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
    1   8   28  56  70  56  28  8   1
  1   9   36  84  126 126 84  36  9   1
1   10  45  120 210 252 210 120 45  10  1
1   11  55  165 330 462 462 330 165 55  11  1
1   12  66  220 495 792 924 792 495 220 66  12  1
```

* 思考题

自行设计一种实现杨辉三角形的方法。

10. 数制转换

将任一整数转换为二进制形式。

* 问题分析与算法设计

将十进制整数转换为二进制形式的方法很多，这里介绍的实现方法利用了C语言能够对位进行操作的特点。对于C语言来说，一个整数在计算机内部就是以二进制形式存储的，所以没有必要再将一个整数经过一系列的运算转化为二进制形式，只要将整数在内存中的二进制表示输出即可。

* 程序说明与注释

```
#include <stdio.h>
main()
{int x;
 printf ("Input number:");
 scanf("%d", &x);
 printf("number of decimal form: %d\n", x);
 printf("     it's binary form: ");
 printb(x, sizeof(int) * 8);
        /* x:整数     sizeof(int):int 型在内存中所占的字节数,
           sizeof(int) * 8:int 型对应的位数 */
putchar('\n');
}
printb (x, n)                                   /* 输出整数 x 二进制形式的后 n 位 */
    int x, n;
{ if (n>0) {
    putchar('0' + ( (unsigned)(x&(1<<((n-1))) >> (n-1) ) );      /* 输出第 n 位 */
    printb (x,n-1);                             /* 递归调用,输出 x 的后 n-1 位 */
  }
}
```

* 运行结果

输入:8

输出:number of decimal form: 8

it's binary form: 0000000000001000

输入:-8

输出:number of decimal form: -8

it's binary form: 1111111111111000

输入:32767

输出:number of decimal form: 32767

it's binary form: 0111111111111111

输入:-32768

输出:number of decimal form: -32768

it's binary form: 1000000000000000

输入:128

输出:number of decimal form: 128

it's binary form: 0000000010000000

* 问题的进一步讨论

充分利用C语言可以对位进行操作的特点,可以编写许多其它高级语言不便于编写甚至根本无法编写的程序。位操作是C语言的一大特点,在深入学习C语言的过程中应力求很好掌握。

程序中使用的位运算方法不是最佳的,也可以不用递归操作,读者可自行对程序进行优化。

* 思考题

将任意正整数转换为四进制或八进制数。

第二章　生活中的数学问题

11. 打鱼还是晒网

中国有句俗语叫“三天打鱼两天晒网”。某人从1990年1月1日起开始“三天打鱼两天晒网”，问这个人在以后的某一天中是在“打鱼”，还是在“晒网”。

* 问题分析与算法设计

根据题意可以将解题过程分为三步：

①计算从1990年1月1日开始至指定日期一共有多少天；

②由于“打鱼”和“晒网”的周期为5天，所以将计算出的天数用5去除；

③根据余数判断他是在“打鱼”，还是在“晒网”：

若　余数为1、2、3，则他是在“打鱼”

否则　是在“晒网”。

在这三步中，关键是第一步。求从1990年1月1日至指定日期有多少天，要判断经历年份中是否有闰年，若是闰年，二月为29天，平年为28天。判断闰年的方法可以用伪语句描述如下：

如果　((年能被4除尽 且 不能被100除尽) 或 能被400除尽)

则　该年是闰年；

否则　不是闰年。

C语言中判断能否整除可使用求余(模)运算。

* 程序与程序注释

```
struct date {
    int year;
    int month;
    int day;
};                                              /* 定义日期结构 */
main()
{ struct date today,term;
  int yearday, year, day;
  printf("Enter year/month/day:");
  scanf("%d%d%d",&today.year,&today.month,&today.day);        /* 输入日期 */
  term.month = 12;                                  /* 设置变量的初始值：月 */
  term.day = 31;                                    /* 设置变量的初始值：日 */
  for (yearday=0,year=1990; year<today.year; year++) {
    term.year = year;
```

```
        yearday += days(term);          /* 计算从1990年至指定年的前一年共有多少天 */
    }
    yearday += days(today);                    /* 加上指定年中到指定日期的天数 */
    day = yearday%5;                                              /* 求余数 */
    if (day>0 && day<4) printf("He was fishing at that day.\n");    /* 打印结果 */
    else printf("He was sleeping at that day.\n");
}
days(day)
    struct date day;
{ static int day_tab[2][13]=                   /* 二维数组形式的天数表作为参数 */
        {{0,31,28,31,30,31,30,31,31,30,31,30,31,},          /* 平年每月的天数 */
         {0,31,29,31,30,31,30,31,31,30,31,30,31,},          /* 闰年每月的天数 */
        };
    int i,lp;
    lp = day.year%4==0 && day.year%100!=0 || day.year%400==0;
                          /* 判定year为闰年还是平年,lp=0为平年,非0为闰年 */
    for (i=1; i<day.month; i++)              /* 计算本年中自1月1日起的天数 */
        day.day+=day_tab[lp][i];
    return(day.day);
}
```

* 运行结果

```
Enter year/month/day: 1991 10 25
                      He was fishing at that day.
Enter year/month/day: 1992 10 25
                      He was sleeping at that day.
Enter year/month/day: 1993 10 25
                      He was sleeping at that day.
```

* 思考题

请打印出任意年份的日历。

12. 抓交通肇事犯

一辆卡车违反交通规则，撞人后逃跑。现场有三人目击事件，但都没有记住车号，只记下车号的一些特征。甲说：牌照的前两位数字是相同的；乙说：牌照的后两位数字是相同的，但与前两位不同；丙是位数学家，他说：四位的车号刚好是一个整数的平方。请根据以上线索求出车号。

* 问题分析与算法设计

按照题目的要求造出一个前两位数相同、后两位数相同且相互间又不同的整数，然后判断该整数是否是另一个整数的平方。

* 程序与程序注释

```
#include "math.h"
main()
{ int i, j, k, c;
    for (i=1; i<=9; i++)                                  /* i:车号前二位的取值 */
      for (j=0; j<=9; j++)                                /* j:车号后二位的取值 */
        if ( i! =j ) {                                    /* 判断两位数字是否相异 */
          k=i*1000+i*100+j*10+j;                          /* 计算出可能的整数 */
          for( c=31; c*c<k; c++);                         /* 判断该数是否为另一整数的平方 */
          if ( c*c==k) printf("Lorry—No. is %d .\n", k);   /* 若是,打印结果 */
        }
}
```

* 运行结果

```
Lorry _ No. is 7744.
```

13. 该存多少钱

假设银行一年整存零取的月息为0.63%。现在某人手中有一笔钱,他打算在今后的五年中每年的年底取出1000元,到第五年时刚好取完,请算出他存钱时应存入多少。

* 问题分析与算法设计

分析存钱和取钱的过程,可以采用倒推的方法。若第五年年底连本带息要取1000元,则要先求出第五年年初银行存款的钱数:

第五年年初存款=1000/(1+12*0.0063)

依次类推可以求出第四年、第三年……的年初银行存款的钱数:

第四年年初存款=(第五年年初存款+1000)/(1+12*0.0063)

第三年年初存款=(第四年年初存款+1000)/(1+12*0.0063)

第二年年初存款=(第三年年初存款+1000)/(1+12*0.0063)

第一年年初存款=(第二年年初存款+1000)/(1+12*0.0063)

通过以上过程就可以容易地求出第一年年初要存入多少钱。

* 程序与程序注释

```
main()
{ int i; float total=0.;
  for (i=0; i<5; i++)                                   /* i 为年数,取值为 0~4 年 */
      total = (total+1000.) / (1+ .0063*12);
                                  /* 累计算出年初存款数额,第五次的计算结果即为题解 */
  printf("He must save %.2f at first.\n", total);
}
```

* 运行结果

```
He must save 4039.44 at first.
```

14. 怎样存钱利最大

假设银行整存整取存款不同期限的月息利率分别为：

0.63%　　期限 = 1 年
0.66%　　期限 = 2 年
0.69%　　期限 = 3 年
0.75%　　期限 = 5 年
0.84%　　期限 = 8 年

利息=本金×月息利率×12×存款年限。

现在某人手中有 2000 元，请通过计算选择一种存钱方案，使得钱存入银行 20 年后得到的利息最多(假定银行对超过存款期限的那部分时间不付利息)。

* 问题分析与算法设计

为了得到最多的利息，存入银行的钱应在到期时马上就取出来，然后立刻将原来的本金和利息加起来再作为新的本金存入银行，这样本利不断地滚动直到满 20 年为止。由于存款的利率不同，所以不同的存款方法(年限)存 20 年得到的利息也是不一样的。

分析题意，设 2000 元存 20 年，其中 1 年存 i1 次，2 年存 i2 次，3 年存 i3 次，5 年存 i5 次，8 年存 i8 次，则到期时存款人应得的本利合计为：

$$2000*(1+rate1)^{i1}*(1+rate2)^{i2}*(1+rate3)^{i3}*(1+rate5)^{i5}*(1+rate8)^{i8}$$

其中 rateN 为对应存款年限的利率。根据题意还可得到以下限制条件：

0 <= i8 <= 2
0 <= i5 <= (20−8*i8)/5
0 <= i3 <= (20−8*i8−5*i5)/3
0 <= i2<= (20−8*i8−5*i5−3*i3)/2
0 <= i1 = 20−8*i8−5*i5−3*i3−2*i2

可以采用穷举法穷举所有的 i8、i5、i3、i2 和 i1 的组合，代入求本利的公式计算出最大值，就是最佳存款方案。

* 程序与程序注释

```
#include <stdio.h>
#include <math.h>
main()
{ int i8, i5, i3, i2, i1, n8, n5, n3, n2, n1;
  float max=0, term;
  for(i8=0; i8<3; i8++)                          /* 穷举全部可能的存款方式 */
    for(i5=0; i5<=(20-8*i8)/5; i5++)
      for(i3=0; i3<=(20-8*i8-5*i5)/3; i3++)
        for(i2=0; i2<=(20-8*i8-5*i5-3*i3)/2; i2++) {
          i1 = 20-8*i8-5*i5-3*i3-2*i2;
          term=2000.0 *pow((double)(1+0.0063*12),(double)i1)
                      *pow((double)(1+2*0.0066*12),(double)i2)
```

```
                    * pow((double)(1+3 * 0.0069 * 12),(double)i3)
                    * pow((double)(1+5 * 0.0075 * 12),(double)i5)
                    * pow((double)(1+8 * 0.0084 * 12),(double)i8);
                                                    /* 计算到期时的本利合计 */
    if (term>max) {                                 /* 若为最大值,记录存款方式 */
      max=term; n1=i1; n2=i2; n3=i3; n5=i5; n8=i8;
    }
}
printf("For maxinum profit, he should so save his money in a bank:\n");
printf("   made fixed deposit for 8 year: %d times\n", n8);
printf("   made fixed deposit for 5 year: %d times\n", n5);
printf("   made fixed deposit for 3 year: %d times\n", n3);
printf("   made fixed deposit for 2 year: %d times\n", n2);
printf("   made fixed deposit for 1 year: %d times\n", n1);
printf("                          Total: %.2f\n", max);
                                                    /* 输出存款方式 */
}
```

* 运行结果

```
For maxinum profit, he should so save his money in a bank:
    made fixed deposit for 8 year: 0 times
    made fixed deposit for 5 year: 4 times
    made fixed deposit for 3 year: 0 times
    made fixed deposit for 2 year: 0 times
    made fixed deposit for 1 year: 0 times
                Total: 8841.01
```

可见最佳存款方案为连续四次存 5 年期。

* 思考题

某单位对职工出售住房,每套为 2 万元。买房付款的方法是:

- 一次交清,优惠 20%。
- 从第一年开始,每年年初分期付款:

  5 年交清,优惠 15%;

  10 年交清,优惠 10%;

  20 年交清,没有优惠。

现在有人手中正好有 2 万元,若假定在今后 20 年中物价和银行利率均保持不变,问他应当选择哪种付款方式可以使应付的钱最少?

15. 捕鱼和分鱼

A、B、C、D、E 五人在某天夜里合伙去捕鱼,到第二天凌晨时都疲惫不堪,于是各自找地方睡觉。日上三杆,A 第一个醒来,他将鱼分为五份,把多余的一条鱼扔掉,拿走自己的一份。B 第

二个醒来，也将鱼分为五份，把多余的一条鱼扔掉，拿走自己的一份。C、D、E 依次醒来，也按同样的方法拿鱼。问他们合伙至少捕了多少条鱼？

* 问题分析与算法设计

根据题意，总计将所有的鱼进行了五次平均分配，每次分配时的策略是相同的，即扔掉一条后剩下的鱼正好分为五份，然后拿走自己的一份，余下其它四份。

假定鱼的总数为 X，则 X 可以按照题目的要求进行五次分配：X－1 后可被 5 整除，余下的鱼为 4×(X－1)÷5。若 X 满足上述要求，则 X 就是题目的解。

* 程序与程序注释

```
main()
{ int n, i, x, flag=1;                                    /* flag:控制标记 */
  for (n=6; flag; n++) {                    /* 采用试探的方法,令试探值 n 逐步加大 */
    for (x=n, i=1 && flag ; i<=5; i++)
                                            /* 判断是否可以按题目要求进行 5 次分配 */
      if ( (x-1)%5 ==0 ) x=4*(x-1)/5;
      else flag=0;                          /* 若不能分配则置标记 flag=0 退出分配过程 */
    if (flag) break;                        /* 若分配过程正常结束则找到结果退出试探的过程 */
    else flag=1;                                          /* 否则继续试探下一个数 */
  }
  printf("Total number of fish catched = %d\n", n);                /* 输出结果 */
}
```

* 运行结果

```
Total number of fish catched = 3121
```

* 问题的进一步讨论

程序采用试探法，试探的初值为 6，每次试探的步长为 1。这是过分保守的做法。可以在进一步深入分析题目的基础上修改初值，增大试探的步长值，以减少试探次数。

* 思考题

请使用其它的方法求解本题。

16. 出售金鱼

买买提将养的一缸金鱼分五次出售：第一次卖出全部的一半加二分之一条；第二次卖出余下的三分之一加三分之一条；第三次卖出余下的四分之一加四分之一条；第四次卖出余下的五分之一加五分之一条；最后卖出余下的 11 条。问原来鱼缸中共有几条鱼？

* 题目分析与算法设计

题目中所有的鱼是分五次出售的，每次卖出的策略相同：第 j 次卖剩下的(j+1)分之一再加 1/(j+1)条。第五次将第四次余下的 11 条全卖了。

假定第 j 次鱼的总数为 X，则第 j 次留下：

$$x - (x+1)/(j+1)$$

当第四次出售完毕时，应该剩下 11 条。若 X 满足上述要求，则 X 就是题目的解。

应当注意的是："(x+1) /(j+1)"应满足整除条件。试探 X 的初值可以从 23 开始，试探的

步长为2,因为X的值一定为奇数。

* 程序说明与注释

```
main()
{ int i, j, n=0, x;                                          /* n为标志变量 */
  for (i=23; n==0; i+=2) {                                   /* 控制试探的过程和步长 */
    for (j=1, x=i; j<=4 && x>=11; j++)                       /* 完成出售四次的操作 */
      if ( (x+1) % (j+1) ==0 )                               /* 若满足整除条件则进行实际的出售操作 */
          x -= (x+1)/(j+1);
      else { x=0; break; }                                   /* 否则停止计算过程 */
    if ( j==5&& x==11 ) {                                    /* 若第四次余下11条则满足题意 */
      printf("There are %d fishes at first.\n", i);          /* 输出结果 */
      n=1;                                                   /* 控制退出试探过程 */
    }
  }
}
```

* 运行结果

```
There are 59 fishes at first.
```

* 思考题

日本著名数学游戏专家中村义作教授提出这样一个问题:父亲将2520个桔子分给六个儿子。分完后父亲说:"老大将分给你的桔子的1/8给老二;老二拿到后连同原先的桔子分1/7给老三;老三拿到后连同原先的桔子分1/6给老四;老四拿到后连同原先的桔子分1/5给老五;老五拿到后连同原先的桔子分1/4给老六;老六拿到后连同原先的桔子分1/3给老大"。结果大家手中的桔子正好一样多。问六兄弟原来手中各有多少桔子?

17. 平分七筐鱼

甲、乙、丙三位渔夫出海打鱼,他们随船带了21只箩筐。当晚返航时,他们发现有7筐装满了鱼,还有7筐只装了半筐鱼,另外7筐则是空的。由于他们没有秤,只好通过目测认为7个满筐鱼的重量是相等的,7个半筐鱼的重量是相等的。在不将鱼倒出来的前提下,怎样将鱼和筐平分为三份?

* 问题分析与算法设计

根据题意可以知道:每个人应分到七个箩筐,其中有3.5筐鱼。采用一个3×3的数组a来表示三个人分到的东西。其中每个人对应数组a的一行,数组的第0列放分到的鱼的整筐数,数组的第1列放分到的半筐数,数组的第2列放分到的空筐数。由题目可以推出:

- 数组的每行或每列的元素之和都为7;
- 对数组的行来说,满筐数+半筐数=3.5;
- 每个人所得到的满筐数不能超过3筐;
- 每个人都必须至少有1个半筐,且半筐数一定为奇数

对于找到的某种分鱼方案,三个人谁拿哪一份都是相同的。为了避免出现重复的分配方案,可以规定:第二个人的满筐数大于等于第一个人的满筐数;第二个人的半筐数大于等于第

一个人的半筐数。

* 程序与程序注释

```
int a[3][3], count;
main()
{ int i, j, k, m, n, flag;
  printf("It exists possible distribution plans:\n");
  for (i=0; i<=3; i++) {          /* 试探第一个人满筐 a[0][0]的值,满筐数不能>3 */
    a[0][0]=i;
    for (j=i; j<=7-i && j<=3; j++) {
                                  /* 试探第二个人满筐 a[1][0]的值,满筐数不能>3 */
      a[1][0] = j;
      if ( (a[2][0]=7-j-a[0][0])>3 ) continue;      /* 第三个人满筐数不能>3 */
      if ( a[2][0]<a[1][0] ) break;
                        /* 要求后一个人分的满筐数 >= 前一个人,以排除重复情况 */
      for (k=1; k<=5; k+=2) {          /* 试探半筐 a[0][1]的值,半筐数为奇数 */
        a[0][1]=k;
        for (m=1; m<7-k; m+=2) {       /* 试探半筐 a[1][1]的值,半筐数为奇数 */
          a[1][1]=m;
          a[2][1]=7-k-m;
          for (flag=1, n=0; flag && n<3; n++)
                  /* 判断每个人分到的鱼是否为 3.5 筐,flag 为满足题意的标记变量 */
            if ( a[n][0]+a[n][1]<7 && a[n][0]*2+a[n][1]==7 )
               a[n][2] = 7-a[n][0]-a[n][1];        /* 计算应得到的空筐数量 */
            else flag=0;                          /* 不符合题意则置标记为 0 */
          if (flag) {                      /* 若符合题意则输出每个人的分配情况 */
            printf(" No. %d          Full basket Semi-basket Empty\n",++count);
            for (n=0; n<3; n++)
                printf("     fisher %c:        %d          %d          %d\n",
                                          'A'+n, a[n][0], a[n][1], a[n][2]);
          }
        }
      }
    }
  }
}
```

* 运行结果

It exists possible distribution plans:

No. 1	Full basket	Semi—basket	Empty
fisher A:	1	5	1
fisher B:	3	1	3
fisher C:	3	1	3
No. 2	Full basket	Semi—basket	Empty
fisher A:	2	3	2
fisher B:	2	3	2
fisher C:	3	1	3

* 思考题

宴会上数学家出了一道难题:假定桌子上有三瓶啤酒,将每瓶中的酒平分给几个人喝,但喝各瓶酒的人数是不一样的,不过其中有一个人喝了每一瓶中的酒,且加起来刚好是一瓶,请问喝这三瓶酒的各有多少人?

(答案:喝三瓶酒的人数分别为 2 人、3 人和 6 人。)

第三章　整数趣题

18. 有限5位数

个位数为6且能被3整除的五位数共有多少？

* 题目分析与算法设计

据题意可知，满足条件的五位数的选择范围是10006、10016、...、99996。可设基础数i=1000，通过计算i*10+6即可得到欲选的数(i的变化范围是1000～9999)，再判断该数能否被3整除。

* 程序说明与注释

```
main()
{ long int i;
  int count=0;                          /* count:统计满足条件的五位数的个数 */
  for (i=1000; i<9999; i++)
    if (! ((i*10+6)%3))                 /* 判断所选的数能否被3整除 */
      count++;                          /* 若满足条件则计数 */
  printf("count=%d\n",count);
}
```

* 运行结果

```
count=2999
```

* 思考题

求100到1000之间有多少个其数字之和为5的整数。

(答案:104,113,122,131,140,203,212,221,230,302,311,320,401,410,500。)

19. 8除不尽的自然数

一个自然数被8除余1，所得的商被8除也余1，再将第二次的商被8除后余7，最后得到一个商为a。又知这个自然数被17除余4，所得的商被17除余15，最后得到一个商是a的2倍。求这个自然数。

* 题目分析与算法设计

据题意，可设最后的商为i(i从0开始取值)，用逆推法可以列出关系式：

$$(((i*8+7)*8)+1)*8+1 = ((2*i*17)+15)*18+4$$

再用试探法求出商i的值。

* 程序说明与注释

```
main()
{ int i;
```

```
    for ( i=0; ;i++ )                                    /* 试探商的值 */
    if ( ((i * 8+7) * 8+1) * 8+1==(34 * i+15) * 17+4) {
                                        /* 逆推判断所取的当前 i 值是否满足关系式 */
                                                        /* 若满足则输出结果 */
      printf ("The required number is :%d\n",(34 * i+15) * 17+4);
      break;                                             /* 退出循环 */
    }
}
```

* 运行结果

The required number is : 1993

20. 一个奇异的三位数

一个自然数的七进制表达式是一个三位数，而这个自然数的九进制表示也是一个三位数，且这两个三位数的数码顺序正好相反，求这个三位数。

* 题目分析与算法设计

据题意可知，七进制和九进制表示的这个自然数的每一位一定小于 7，可设其七进制数形式为 kji(i、j、k 的取值分别为 1～6)，然后设其九进制表示形式为 ijk。

* 程序说明与注释

```
main()
{ int i,j,k;
  for (i=1; i<7; i++)                              /* 穷举 9 进制的第一位数字 */
    for (j=0; j<7; j++)                            /* 穷举 9 进制的第二位数字 */
      for (k=1; k<7; k++)                          /* 穷举 9 进制的第三位数字 */
        if (i * 9 * 9+j * 9+k==i+j * 7+k * 7 * 7) {      /* 判断是否满足题意 */
          printf ("The special number with 3 digits is: ");
          printf ("%d%d%d(7)=%d%d%d(9)=%d(10)\n",
                                  k,j,i, i,j,k, i * 9 * 9+j * 9+k);
    }
}
```

* 运行结果

The special number with 3 digits is: 503(7) = 305(9) = 248(10)

21. 4 位反序数

设 N 是一个四位数，它的 9 倍恰好是其反序数，求 N。反序数就是将整数的数字倒过来形成的整数。例如：1234 的反序数是 4321。

* 题目分析与算法设计

可设整数 N 的千、百、十、个位为 i、j、k、l，其取值均为 0～9，则满足关系式：

$$(i\times10^3+j\times10^2+10\times k+l)\times9=(l\times10^3+k\times10^2+10\times j+i)$$

的 i、j、k、l 即构成 N。

* 程序说明与注释

```
main()
{ int i;
  for (i=1002; i<1111; i++)                          /* 穷举四位数可能的值 */
    if ( i%10*1000+i/10%10*100+i/100%10*10+i/1000==i*9 )
                                                     /* 判断反序数是否是原整数的 9 倍 */
      printf ("The number satisfied states condition is: %d\n", i);
                                                          /* 若是则输出 */
}
```

* 运行结果

```
The number satisfied states condition is: 1089
```

22. 求车速

一辆以固定速度行驶的汽车，司机在上午 10 点看到里程表上的读数是一个对称数(即这个数从左向右读和从右向左读是完全一样的)，为 95859。两小时后里程表上出现了一个新的对称数。问该车的速度是多少？新的对称数是多少？

* 题目分析与算法设计

据题意，设所求对称数为 i，其初值为 95859，对其依次递增取值，将 i 值的每一位分解后与其对称位置上的数进行比较，若每个对称位置上的数皆相等，则可判定 i 即为所求的对称数。

* 程序说明与注释

```
main()
{ int t,a[5];                                        /* 数组 a 存放分解的数字位 */
  long int k, i;
  for (i=95860; ;i++) {                              /* 以 95860 为初值，循环试探 */
    for(t=0, k=100000; k>=10; t++) {                 /* 从高到低分解所取 i 值的每位数 */
      a[t] = (i%k)/(k/10);                           /* 字，依次存放于 a[0]~a[5]中 */
      k/=10;
    }
    if((a[0]==a[4]) && (a[1]==a[3])) {               /* 判断若为对称数，则输出 */
      printf("The new symmetrical number kelometers is: %d%d%d%d%d\n",
                                                     a[0], a[1], a[2], a[3], a[4]);
      printf("The velocity of the car is: %.2f\n",(i-95859)/2.0);
      break;
    }
  }
}
```

* 运行结果

```
The new symmetrical number kelometers is: 95959.
The velocity of the car is: 50.00
```

* 思考题

将一个数的数码倒过来所得到的新数叫原数的反序数。如果一个数等于它的反序数，则称它为对称数。求不超过 1993 的最大的二进制的对称数。

23. 由两个平方三位数获得三个平方二位数

已知两个平方三位数 abc 和 xyz，其中 a、b、c、x、y、z 未必是不同的；而 ax、by、cz 是三个平方二位数。请编程求三位数 abc 和 xyz。

* 题目分析与算法设计

任取两个平方三位数 n 和 n1，将 n 从高向低分解为 a、b、c，将 n1 从高到低分解为 x、y、z。判断 ax、by、cz 是否均为完全平方数。

* 程序说明与注释

```
#include "math.h"
main()
{ void f();
  int i,t; float a[3]b[3];
  printf ("The possible perfect squares combinations are:\n");
  for (i=11; i<=31; i++)                          /* 穷举平方三位数的取值范围 */
    for (t=11; t<=31; t++) {
      f(i*i,a);                    /* 分解平方三位数的各位，每位数字分别存入数组中 */
      f(t*t,b);
      if (sqrt(a[0]*10+b[0])==(int)sqrt(a[0]*10+b[0])
          && sqrt(a[1]*10+b[1])==(int)sqrt(a[1]*10+b[1])
          && sqrt(a[2]*10+b[2])==(int)sqrt(a[2]*10+b[2]) )
                                                  /* 若三个新的数均是完全平方数 */
        printf(" %d and %d\n",i*i,t*t);                          /* 则输出 */
    }
}
void f(n,s)                                  /* 分解三位数 n 的各位数字，将各个数字 */
  int n; float *s;                          /* 从高到低依次存入指针 s 所指向的数组中 */
{ int k;
  for (k=1000; k>=10; s++) {
      *s = (n%k)/(k/10);
      k /= 10;
  }
}
```

* 运行结果

```
The possible perfect squares combinations are:
    400 and 900
    841 and 196
```

* 思考题

求这样一个三位数，该三位数等于其每位数字的阶乘之和。

即　　abc = a! + b! + c!

（正确结果：145 = 1! + 4! + 5!）

24. 阿姆斯特朗数

如果一个正整数等于其各个数字的立方和，则该数称为阿姆斯特朗数（亦称为自恋性数）。如 $407=4^3+0^3+7^3$ 就是一个阿姆斯特朗数。试编程求 1000 以内的所有阿姆斯特朗数。

* 题目分析与算法设计

可采用穷举法，依次取 1000 以内的各数（设为 i），将 i 的各位数字分解后，据阿姆斯特朗数的性质进行计算和判断。

* 程序说明与注释

```
main()
{int i,t,k,a[3];
 printf ("There are following Armstrong number smaller than 1000:\n");
 for (i=2; i<1000; i++) {                    /* 穷举要判定的数 i 的取值范围 2～1000 */
   for (t=0,k=1000;k>=10; t++) {             /* 截取整数 i 的各位(从高位向低位) */
     a[t]=(i%k)/(k/10);                      /* 分别赋于 a[0]～a[2] */
     k/=10;
   }
   if ( a[0]*a[0]*a[0]+a[1]*a[1]*a[1]+a[2]*a[2]*a[2]==i )
                                             /* 判断 i 是否为阿姆斯特朗数 */
       printf(" %d ",i);                     */若满足条件，则输出 */
  }
}
```

* 运行结果

There are following Armstrong number smaller than 1000:

153　　370　　371　　407

25. 完全数

如果一个数恰好等于它的因子之和，则称该数为“完全数”。如：6 的因子是 1、2、3，而 6=1+2+3，则 6 是个“完全数”。试求出 1000 以内的全部“完全数”。

* 题目分析与算法设计

据完全数的定义，先计算所选取的整数 a（a 的取值 1～1000）的因子，将各因子累加于 m，若 m 等于 a 则可确认 a 为完全数。

* 程序说明与注释

```
main()
{ int a, i, m;
  printf ("There are following perfect numbers smaller than 1000:\n");
```

```
  for (a=1; a<1000; a++) {              /* 循环控制选取 1～1000 中的各数进行判断 */
    for (m=0,i=1; i<=a/2; i++)  /* 计算 a 的因子,并将各因子依次累加于 m 中 */
      if (! (a%i)) m+=i;
    if (m==a)                    /* 判断若数 a 的各因子之和 m=a,则 a 是完全数输出 */
      printf(" %4d", a);
  }
}
```

* 运行结果

There are following perfect numbers smaller than 1000:

6　　28　　496

26. 亲密数

如果整数 A 的全部因子(包括 1,不包括 A 本身)之和等于 B;且整数 B 的全部因子(包括 1,不包括 B 本身)之和等于 A,则将整数 A 和 B 称为亲密数。求 3000 以内的全部亲密数。

* 题目分析与算法设计

按照亲密数定义,要判断数 a 是否有亲密数,只要计算出 a 的全部因子的累加和为 b,再计算 b 的全部因子的累加和为 n,若 n 等于 a 则可判定 a 和 b 是亲密数。计算数 a 的各因子的算法:

用 a 依次对 i(i=1～a/2)进行模运算,若模运算结果等于 0,则 i 为 a 的一个因子;否则 i 就不是 a 的因子。

* 程序说明与注释

```
main()
{ int a, i, b, n;
  printf("There are following friendly - numbers pair samller than 3000:\n");
  for (a=1; a<3000; a++) {                          /* 穷举 1000 以内的全部整数 */
    for (b=0; i=1; i<=a/2; i++)          /* 计算数 a 的各因子,各因子之和存于 b */
      if (! (a%i)) b+=i;                    /* 计算 b 的各因子,各因子之和存于 n */
    for (n=0, i=1; i<=b/2; i++)
      if (! (b%i)) n+=i;
    if (n==a && a<b)
      printf(" %4d..%4d",a,b);              /* 若 n=a,则 a 和 b 是一对亲密数,输出 */
  }
}
```

* 运行结果

There are following friendly－numbers pair samller than 3000:

220.. 284　　1184..1210　　2620..2924

27. 自守数

自守数是指一个数的平方的尾数等于该数自身的自然数。例如:

$25^2=625$　　$76^2=5776$　　$9376^2=87909376$

请求出 200000 以内的自守数。

* 题目分析与算法设计

若采用“求出一个数的平方后再截取最后相应位数”的方法显然是不可取的，因为计算机无法表示过大的整数。

分析手工方式下整数平方(乘法)的计算过程，以 376 为例：

```
      376        被乘数
×     376          乘数
-------------
      2256       第一个部分积＝被乘数×乘数的倒数第一位
     2632        第二个部分积＝被乘数×乘数的倒数第二位
    1128         第三个部分积＝被乘数×乘数的倒数第三位
-------------
    141376       积
```

本问题所关心的是积的最后三位。分析产生积的后三位的过程，可以看出：在每一次的部分积中，并不是它的每一位都会对积的后三位产生影响。总结规律可以得到：在三位数乘法中，对积的后三位产生影响的部分积分别为：

第一个部分积中：被乘数最后三位×乘数的倒数第一位

第二个部分积中：被乘数最后二位×乘数的倒数第二位

第三个部分积中：被乘数最后一位×乘数的倒数第三位

将以上的部分积的后三位求和后截取后三位就是三位数乘积的后三位。这样的规律可以推广到同样问题的不同位数乘积。

按照手工计算的过程可以设计算法编写程序。

* 程序说明与注释

```
main()
{ long mul, number, k, ll, kk;
  printf("It exists following automorphic numbers smaller than 200000:\n");
  for (number=0; number<200000; number++) {
    for ( mul=number, k=1; (mul/=10)>0; k*=10) ;
                              /* 由 number 的位数确定截取数字进行乘法时的系数 k */
    kk = k*10;                                    /* kk 为截取部分积时的系数 */
    mul=0;                                        /* 积的最后 N 位 */
    ll=10;                                        /* ll 为截取乘数相应位时的系数 */
    while ( k>0 ) {
      mul = (mul + (number%(k*10)) * (number%ll- number%(ll/10)) ) % kk;
       /* (部分积+截取被乘数的后 N 位*截取乘数的第 M 位), %kk 再截取部分积 */
      k /= 10;                                    /* k 为截取被乘数时的系数 */
      ll *= 10;
    }
    if ( number == mul )                          /* 判断若为自守数则输出 */
```

```
        printf(" %ld", number);
    }
}
```

* 运行结果

It exists following automorphic numbers smaller than 200000:

0 1 5 6 25 76 376 625 9376 90625 109376

28. 回文数

打印所有不超过 n(取 n<256)的,其平方具有对称性质的数(也称为回文数)。

* 题目分析与算法设计

对于要判断的数 n,计算出其平方后(存于 a),将 a 的每一位进行分解,再按 a 的从低到高的顺序将其恢复成一个数 k(如 n=13,则 a=169 且 k=961),若 a 等于 k 则可判定 n 为回文数。

* 程序说明与注释

```
main()
{ int m[16], n, i, t, count=0;
  long unsigned a, k;
  printf(" No. number it's square (palindrome)\n");
  for(n=1; n<256; n++) {                          /* 穷举 n 的取值范围 */
    k=0; t=1; a=n*n;                              /* 计算 n 的平方 */
    for (i=1; a! =0; i++) {/* 从低到高分解数 a 的每一位存于数组 m[1]~m[16] */
      m[i] = a%10;
      a/=10;
    }
    for (; i>1; i--) {                            /* 生成 a 的反序数 */
      k += m[i-1]*t;
      t *= 10;
    }
    if (k==n*n)                                   /* 判断是否为回文数,若是则输出 */
      printf(" %2d %10d %10d\n", ++count, n, n*n);
  }
}
```

* 运行结果

No.	number	it's square(palindrome)
1	1	1
2	2	4
3	3	9
4	11	121
5	22	484

6	26	676
7	101	10201
8	111	12321
9	121	14641

29. 求具有 abcd=(ab+cd)² 性质的四位数

3025 这个数具有一种独特的性质:将它平分为两段,即 30 和 25,使之相加后求平方,即 $(30+25)^2$,恰好等于 3025 本身。请求出具有这样性质的全部四位数。

* 题目分析与算法设计

具有这种性质的四位数没有分布规律,可采用穷举法,对所有四位数进行判断,从而筛选出符合这种性质的四位数。具体算法实现,可任取一个四位数,将其截为两部分,前两位为 a,后两位为 b,然后套用公式计算并判断。

* 程序说明与注释

```
main()
{ int n,a,b;
  printf ("There are following numbers with 4 digits satisfied condition:\n");
  for (n=1000; n<10000; n++)}                    /* 四位数 N 的取值范围 1000～9999 */
    a=n/100;                                     /* 截取 N 的前两位数存于 a */
    b=n%100;                                     /* 截取 N 的后两位数存于 b */
    if( (a+b)*(a+b)==n )      /* 判断 N 是否为符合题目所规定的性质的四位数 */
      printf(" %d",n);
  }
}
```

* 运行结果

```
There are following numbers with 4 digits satisfied condition :
    2025    3025    9801
```

第四章　素数的家族

30. 求素数

求素数表中 1 到 1000 之间的所有素数。

＊ 问题分析与算法设计

素数就是仅能被 1 和它自身整除的整数。判断一个整数 n 是否为素数就是要判断整数 n 能否被除 1 和自身之外的任意整数整除，若都不能整除，则 n 为素数。

程序设计时 i 可以从 2 开始，到该整数 n 的 1/2 为止，用 i 依次去除需要判断的整数，只要存在可以整除该数的情况，即可确定要判断的整数不是素数，否则是素数。

＊ 程序与程序注释

```
#include <stdio.h>
main()
{int n1, nm, i, j, flag, count=0;
 do {
     printf("Input START and END=?");
     scanf ("%d%d", &n1, &nm);                          /* 输入求素数的范围 */
   } while (! (n1>0 && n1<nm));                          /* 输入正确的范围 */
   printf
   ("------------------------------ PRIME TABLE(%d - %d)------------------------------\n",n1,nm);
   if (n1==1 || n1==2) {                                 /* 处理素数 2 */
      printf("%4d ", 2);
      n1 =3; count++;
  }
    for (i=n1; i<=nm; i++) {                  /* 判断指定范围内的整数是否为素数 */
       if (! (i%2)) continue;
       for (flag=1,j=3; flag&&j<i/2; j+=2)
                                  /* 判断能否被从 3 到整数的一半中的某一数所整除 */
           if (! (i%j)) flag =0;                         /* 若能整除则不是素数 */
       if (flag) printf (++count%15 ? "%4d " : "%4d\n", i);
  }
}
```

＊ 运行结果

```
Input START and END=? 1 1000
```

PRIME TABLE(1 — 1000)

2	3	5	7	11	13	17	19	23	29	31	37	41	43	47
53	59	61	67	71	73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173	179	181	191	193	197
199	211	223	227	229	233	239	241	251	257	263	269	271	277	281
283	293	307	311	313	317	331	337	347	349	353	359	367	373	379
383	389	397	401	409	419	421	431	433	439	443	449	457	461	463
467	479	487	491	499	503	509	521	523	541	547	557	563	569	571
577	587	593	599	601	607	613	617	619	631	641	643	647	653	659
661	673	677	683	691	701	709	719	727	733	739	743	751	757	761
769	773	787	797	809	811	821	823	827	829	839	853	857	859	863
877	881	883	887	907	911	919	929	937	941	947	953	967	971	977
983	991	997												

* 思考题

请找出十个最小的连续自然数，它们个个都是合数(非素数)。

31. 哥德巴赫猜想

验证：2000 以内的正偶数都能够分解为两个素数之和(即验证哥德巴赫猜想对 2000 以内的正偶数成立)。

* 问题分析与算法设计

为了验证哥德巴赫猜想对 2000 以内的正偶数是成立的，要将整数分解为两部分，然后判断分解出的两个整数是否均为素数。若是，则满足题意；否则，重新进行分解和判断。

程序中对判断是否为素数的算法进行了改进，对整除判断“用从 2 开始到该整数的一半”改为“2 开始到该整数的平方根”。原因何在请读者自行分析。

* 程序与程序注释

```
#include <stdio.h>
#include <math.h>
main()
{ int i, n;
  for (i=4; i<=2000; i+=2) {
    for(n=2; n<i; n++)                          /* 将偶数 i 分解为两个整数 */
      if (fflag(n))                             /* 分别判断两个整数是否均为素数 */
        if (fflag(i-n)) {
          printf("%d=%d+%d\n", i, n ,i-n);      /* 若均是素数则输出 */
          break;
        }
      if (n==i) printf("error %d\n", i);
  }
}
```

```
fflag(i)                                                    /* 判断整数是否为素数 */
    int i;
{ int j;
  if (i<=1) return(0);
  if (i==2) return(1);
  if (! (i%2)) return(0);                                   /* if no, return 0 */
  for (j=3; j<=(int)(sqrt((double)i)+1); j+=2)
    if (! (i%j)) return(0);
  return(1);                                                /* if yes, return 1 */
}
```

* 运行结果

最后的15组解为：

1972=23+1949	1974=23+1951	1976=3+1973	1978=5+1973	1980=7+1973
1982=3+1979	1984=5+1979	1986=7+1979	1988=37+1951	1990=3+1987
1992=5+1987	1994=7+1987	1996=3+1993	1998=5+1993	2000=3+1997

32. 可逆素数

求四位的可逆素数。可逆素数是指：一个素数将其各位数字的顺序倒过来构成的反序数也是素数。

* 问题分析与算法设计

本题的重点不是判断素数的方法，而是求一个整数的反序数。

求反序数的方法是从整数的末尾依次截取最后一位数字，每截取一次后整数缩小10倍，将截取的数字作为新的整数的最后一位（新的整数扩大10倍后加上被截取的数字）。这样原来整数的数字从低到高被不断地截取，依次作为新的整数从高到低的各位数字。

* 程序与程序注释

```
#include <math.h>
main()
{ int i, count;
  printf ("There are invertable primes with 4 digits: \n");
  for ( count=0, i=1001; i<9999; i+=2)                      /* 穷举全部的奇数 */
    if ( num(i) )                                           /* 若为可逆素数,则输出 */
      printf( count%9 ? "%3d:%d " : "%3d:%d\n", ++count, i);
}
num (int number)
{ int j, i;
  if ( ! ok(number) ) return(0);                            /* 判断整数是否为素数 */
  for (i=number, j=0; i>0; i/=10)                           /* 按位将整数倒过来,产生反序数 */
    j = j*10 + i%10;
  if ( number < j )                                         /* 若原数小于反序数 */
```

```
    if ( ! ok(j) ) return(0);                    /* 则判断对应的反序数是否为素数 */
    else return(1);                              /* 若是可逆素数,则返回 1 */
  else return(0);                                /* 不是可逆素数,返回 0 */
}
ok ( int number)
{ int i, j;
  if (number%2==0) return (0);                   /* 判断整数 number 是否为素数 */
  j = sqrt( (double) number) + 1;                /* 取整数的平方根为判断的上限 */
  for (i=3; i<=j ; i+=2)                         /* 若为素数则返回 1,否则返回 0 */
    if ( number % i == 0 ) return(0);
  return(1);
}
```

* 运行结果

There are invertable primes with 4 digits:

```
1:1009    2:1021    3:1031    4:1033    5:1061    6:1069    7:1091    8:1097    9:1103
10:1109   11:1151   12:1153   13:1181   14:1193   15:1213   16:1217   17:1223   18:1229
19:1231   20:1237   21:1249   22:1259   23:1279   24:1283   25:1381   26:1399   27:1409
28:1429   29:1439   30:1453   31:1471   32:1487   33:1499   34:1523   35:1559   36:1583
37:1597   38:1619   39:1657   40:1669   41:1723   42:1733   43:1753   44:1789   45:1847
46:1867   47:1879   48:1913   49:1933   50:1949   51:1979   52:3019   53:3023   54:3049
55:3067   56:3083   57:3089   58:3109   59:3163   60:3169   61:3257   62:3299   63:3319
64:3343   65:3347   66:3359   67:3373   68:3389   69:3407   70:3463   71:3467   72:3469
73:3527   74:3583   75:3697   76:3719   77:3767   78:3889   79:3917   80:3929   81:7027
82:7057   83:7177   84:7187   85:7219   86:7229   87:7297   88:7349   89:7457   90:7459
91:7529   92:7577   93:7589   94:7649   95:7687   96:7699   97:7879   98:7949   99:9029
100:9349 101:9479 102:9679
```

* 思考题

求 1000 以内的孪生素数。孪生素数是指:若 a 为素数,且 a+2 也是素数,则素数 a 和 a+2 称为孪生素数。

33. 回文素数

求不超过 1000 的回文素数。

* 问题分析与算法设计

所谓回文素数是指,对一个整数 n 从左向右和从右向左读其结果值相同且是素数,即称 n 为回文素数。所以本题的重点不是判断素数的方法,而是求回文整数。构造回文数的方法很多,这里仅介绍一种最简单的算法。实现思路是先求出一个整数的回文数,再判断是否为素数。

不超过 1000 的回文数包括二位和三位的回文数,我们采用穷举法来构造一个整数并求与其对应的反序数,若整数与其反序数相等,则该整数就是回文数。

* 程序与程序注释

```
main()
{ int i, j, t, k, s;
  printf ("Following are palindrome primes not greater than 1000:\n");
  for (i=0; i<=9; i++)                                    /* 穷举第一位 */
    for (j=0; j<=9; j++)                                  /* 穷举第二位 */
      for (k=0; k<=9; k++) {                              /* 穷举第三位 */
        s=i*100+j*10+k;                                   /* 计算组成的整数 */
        t=k*100+j*10+i;                                   /* 计算对应的反序数 */
        if (i==0 && j==0 ) t/=100;                        /* 处理整数的前两位为 0 情况 */
        else if (i==0) t/=10;                             /* 处理整数的第一位为 0 情况 */
        if ( s>10 && s==t && a(s) )                       /* 若大于 10 且为回文素数,则输出 */
          printf("%d\t", s);
    }
}
a(n)                                                      /* 函数 a[n]的功能是判断整数 n 是否为素数 */
  int n;
{ int i;
  for (i=2; i<=(n-1)/2; i++)
    if(n%i==0) return(0);
  return(1);
}
```

* 运行结果

```
Following are palindrome primes not greater than 1000:
  11   101   131   151   181   191   313   353
 373   383   727   757   787   797   919   929
```

* 思考题

优化生成回文数的算法。

34. 要发就发

“1898—要发就发”。今年是一九九三年,现将不超过 1993 的所有素数从小到大排成第一行,第二行上的每个数都等于它“右肩”上的素数与“左肩”上的素数之差。请编程求出:第二行数中是否存在这样的若干个连续的整数,它们的和恰好是 1898?假如存在的话,又有几种这样的情况?

第一行:2 3 5 7 11 13 17 ..... 1979 1987 1993

第二行:1 2 2 4 2 4 ..... 8 6

* 问题分析与算法设计

首先从数学上分析该问题。

假设第一行中的素数为 n[1]、n[2]、n[3]、…、n[i]、…,第二行中的差值为 m[1]、m[2]、m[3]、…、m[j]、…。其中 m[j]为:

$$m[j] = n[j+1] - n[j]。$$

则第二行连续 N 个数的和为：

$$\begin{aligned} SUM &= m[1] + m[2] + m[3] + \ldots + m[j] \\ &= (n[2]-n[1])+(n[3]-n[2])+(n[4]-n[3])+ \ldots + (n[j+1]-n[j]) \\ &= n[j+1] - n[1] \end{aligned}$$

由此，原题目就变成了：在不超过 1993 的所有素数中是否存在这样两个素数，它们的差恰好是 1898。若存在，则第二行中必有所需整数序列，其和恰为 1898。

对等价问题的求解是比较简单的。

由分析可知，在素数序列中不必包含 2，因为任意素数与 2 的差一定为奇数，所以不必考虑。

* 程序与程序注释

```
#include <stdio.h>
#include <math.h>
#define NUM 320
int number[NUM];                                    /* 存放小于 1993 的全部奇素数 */
main()
{ int i, j , count=0;
  printf ("There are following primes sequences in first row:\n");
  for (j=0, i=3; i<=1993; i+=2)                     /* 求出不超过 1993 的全部奇素数 */
    if (fflag(i)) number[j++]=i;
  for ( j--; number[j]>1898; j--) {                 /* 从最大的素数开始向 1898 搜索 */
    for (i=0; number[j]-number[i]>1898; i++) ;      /* 循环查找满足条件的素数 */
    if (number[j]-number[i]==1898)                  /* 若两个素数的差为 1898,则输出 */
      printf(" (%d). %3d,.....,%d\n", ++count, number[i], number[j]);
  }
}
fflag(i)
  int i;
{ int j;
  if (i<=1) return(0);                              /* 判断是否为素数 */
  if (i==2) return(1);
  if (! (i%2)) return(0);                           /* if no, return 0 */
  for (j=3; j<=(int)( sqrt((double)i)+1 ); j+=2)
    if (! (i%j)) return(0);
  return(1);                                        /* if yes, return 1 */
}
```

* 运行结果

```
There are following primes sequences in first row:
    (1). 89,.....,1987
```

(2). 53,.....,1951

(3). 3,.....,1901

* 思考题

将 1,2,3,…,20 这 20 个连续的自然数排成一圈，使任意两个相邻的自然数之和均为素数。

35. 素数幻方

求四阶的素数幻方。即在一个 4×4 的矩阵中，每一个格填入一个数字，使每一行、每一列和两条对角线上的 4 个数字所组成的四位数，均为可逆素数。

* 问题分析与算法设计

有了前面的基础，本题应当说是不困难的。

最简单的算法是：采用穷举法，设定 4×4 矩阵中每一个元素的值后，判断每一行、每一列和两条对角线上的 4 个数字组成的四位数是否都是可逆素数。若是则求出了满足题意的一个解。

这种算法在原理上是对的，也一定可以求出满足题意的全部解。但是，按照这一思路编出的程序效率极低，在微机上几个小时也不会运行结束。这一算法致命的缺陷是：要穷举和判断的情况过多。

充分利用题目中的“每一个四位数都是可逆素数”这一条件，可以放弃对矩阵中每个元素进行穷举的算法，先求出全部的四位可逆素数(204 个)，以矩阵的行为单位，在四位可逆素数的范围内进行穷举，然后将穷举的四位整数分解为数字后，再进行列和对角线方向的条件判断。改进的算法与最初的算法相比，大大地减少了穷举的次数。

考虑矩阵的第一行和最后一行数字，它们分别是列方向四位数的第一个数字和最后一个数字，由于这些四位数也必须是可逆素数，所以矩阵的第一行和最后一行中的各个数字都不能为偶数或 5。这样穷举矩阵的第一行和最后一行时，它们的取值范围是：所有位的数字均不是偶数或 5 的四位可逆素数。由于符合这一条件的四位可逆素数很少，所以这一范围限制又一次减少了穷举的次数。

对算法进一步研究会发现：当设定了第一和第二行的值以后，就已经可以判断出当前的这种组合是否一定是错误的(尚不能肯定该组合一定是正确的)。若按列方向上的四个两位数与四位可逆素数的前两位矛盾(不是其中的一种组合)，则第一、二行的取值一定是错误的。同理，在设定了前三行数据后，可以立刻判断出当前的这种组合是否一定是错误的。若判断出矛盾情况，则可以立刻设置新的一组数据。这样就可避免将四个数据全设定好以后再进行判断所造成的低效。

根据以上分析，可以用伪语言描述以上改进的算法：

```
开始
  找出全部四位的可逆素数；
  确定全部可出现在第一和最后一行的四位可逆素数；
  在指定范围内穷举第一行
    在指定范围内穷举第二行
        若第一、二行已出现矛盾，则继续穷举下一个数；
```

在指定范围内穷举第三行

若第一、二、三行已出现矛盾，则继续穷举下一个数；

在指定范围内穷举第四行

判断列和对角线方向是否符合题意

若符合题意，则输出矩阵；

否则继续穷举下一个数；

结束。

在实际编程中，采用了很多程序设计技巧，例如设置若干辅助数组，其目的就是要最大限度地提高程序的执行效率，缩短运行时间。下面的程序在386机上仅运行10分钟。

```
* 程序与程序注释
#include <math.h>
int number[210][5];                    /* 存放可逆素数及素数分解后的各位数字 */
int select[110];                       /* 可以放在矩阵第一行和最后一行的素数的下标 */
int array[4][5];                       /* 4×4的矩阵，每行0号元素存可逆素数对应的数组下标 */
int count;                             /* 可逆素数的数目 */
int selecount;                         /* 可以放在矩阵第一行和最后一行的可逆素数的数目 */
int larray[2][200];                    /* 存放素数前二、三位数值的临时数组 */
int lcount[2];                         /* 存放素数前二、三位的临时数组所对应的数量计数器 */
main()
{ int i, k, flag, cc=0, i1,i4;
  printf("There are magic squares with invertable primes as follow:\n");
  for ( i=1001; i<9999; i+=2) {                    /* 求满足条件的可逆素数 */
    k = i / 1000;
    if ( k%2! =0 && k! =5 && num(i) ) {    /* 若可逆素数的第一位不是偶数或5 */
      number[count][0]=i;                          /* 存入数组 */
      process(count++);                            /* 分解素数的各位数字 */
      if ( number[count-1][2]%2! =0 &&     /* 若可逆素数满足放在矩阵第一行 */
           number[count-1][3]%2! =0 &&     /* 和最后一行的条件，记录可逆素 */
           number[count-1][2]! =5 &&       /* 数的下标，计数器加1 */
           number[count-1][3]! =5 )
        select[selecount++] = count-1 ;
  }
}
larray[0][lcount[0]++] = number[0][0]/100;       /* 临时数组的第一行存前二位 */
larray[1][lcount[1]++] = number[0][0]/10;        /* 临时数组的第二行存前三位 */
for (i=1; i<count; i++) {                /* 将素数不重复的前二、三位存入临时数组中 */
  if (larray[0][ lcount[0]-1 ] ! = number[i][0]/100 )
    larray[0][ lcount[0]++ ] = number[i][0]/100;
  if (larray[1][ lcount[1]-1 ] ! = number[i][0]/10 )
```

```
    larray[1][ lcount[1]++ ] = number[i][0]/10;
}
for (i1=0; i1<selecount; i1++) {                    /* 在第一行允许的范围内穷举 */
  array[0][0] = select[i1];                          /* 取对应的素数下标 */
  copy _num ( 0 );                                   /* 复制分解的数字 */
  for (array[1][0]=0; array[1][0]<count; array[1][0]++) {     /* 穷举第二行 */
    copy _num ( 1 );                                 /* 复制分解的数字 */
    if ( ! comp _num (2) )
      continue;                  /* 若每列的前两位的组成与素数矛盾,则试探下一个数 */
    for (array[2][0]=0; array[2][0]<count; array[2][0]++){     /* 穷举第三行 */
    copy _num ( 2 );                                 /* 复制分解的数字 */
    if ( ! comp _num (3) )
      continue;                  /* 若每列的前三位的组成与素数矛盾,则试探下一个数 */
    for (i4=0; i4<selecount; i4++) {                 /* 在最后一行允许的范围内穷举 */
      array[3][0] = select[i4];
      copy _num ( 3 );                               /* 复制分解的数字 */
      for (flag=1, i=1; flag && i<=4; i++)           /* 判断每列是否可逆素数 */
        if ( ! find1(i) ) flag=0;
      if ( flag && find2() )                         /* 判断对角线是否为可逆素数 */
        { printf("No. %d\n", ++cc); p _array(); }              /* 输出幻方矩阵 */
    }
   }
  }
 }
}
num (int number)                                     /* 判断是否可逆素数 */
{ int j;
  if ( ! ok(number) ) return(0);
  for (j=0; number>0; number/=10)                    /* 将素数变为反序数 */
    j = j*10 + number%10;
  if ( ! ok(j) ) return(0);                          /* 判断反序数是否为素数 */
  return(1);
}
ok ( int number)                                     /* 判断是否为素数 */
{ int i, j;
  if (number%2==0) return (0);
  j = sqrt( (double) number) + 1;
  for (i=3; i<=j ; i+=2)
    if ( number % i == 0 ) return(0);
```

```
  return(1);
}
process (int i)                                  /* 将第 i 个整数分解为数字并存入数组 */
{ int j, num;
  num = number[i][0];
  for (j=4; j>=1; j--, num/=10)
    number[i][j] = num%10;
}
copy_num (int i)              /* 将 array[i][0]指向的素数的各位数字复制到 array[i]中 */
{ int j;
  for (j=1; j<=4; j++)
    array[i][j] = number[array[i][0]][j];
}
comp_num ( int n ) /* 判断 array 中每列的前 n 位是否与可逆素数允许的前 n 位矛盾 */
{ static int ii;                          /* 用内部静态变量保存前一次查找到的元素下标 */
  static int jj;                 /* ii:前一次查找前二位的下标,jj 前一次查找前三位的下标 */
  int i, num, k, *p;                        /* p:指向对应的要使用的"前一次下标"ii 或 jj */
  int *pcount;                             /* pcount:指向要使用的临时数组数量的计数器 */
  switch ( n ) {                                    /* 根据 n 值选择对应的一组控制变量 */
    case 2:pcount=&lcount[0];p=&ii;break;
    case 3:pcount=&lcount[1];p=&jj;break;
    default: return(0);
   }
    for (i=1; i<=4; i++) {                                    /* 对四列进行分别处理 */
      for (num=0, k=0; k<n; k++)                        /* 计算前 n 位数字代表的数值 */
        num = num*10+array[k][i];
      if ( num <= larray[n-2][*p] )                 /* 与前一次最后查到的元素进行比较 */
        for( ; *p>=0 && num<larray[n-2][*p]; (*p)--) ;
                                                        /* 若前次查到的元素大,则向前找 */
      else
        for( ; *p<*pcount && num>larray[n-2][*p]; (*p)++) ;
                                                                  /* 否则向后查找 */
      if ( *p<0 || *p>=*pcount ) { *p=0; return(0); }
      if ( num! =larray[n-2][*p] )
        return(0);                              /* 前 n 位不是可逆素数允许的值则返回 0 */
   }
  return(1);
}
find1 ( int i)                                          /* 判断列方向是否是可逆素数 */
```

```
{ int num, j;
  for (num=0,j=0; j<4; j++) num = num * 10+ array[j][i];
  return ( find0(num) );
}
find2 ( void )                                  /* 判断对角线方向是否是可逆素数 */
{ int num1, num2, j, i;
  for ( num1=0,j=0; j<4; j++)
     num1 = num1 * 10+ array[j][j+1];
  for (num2=0,j=0,i=4; j<4; j++,i--)
     num2 = num2 * 10+ array[j][i];
  if ( find0(num1) ) return(find0(num2));
  else return ( 0 );
}
find0 ( int num )                               /* 查找是否为满足要求的可逆素数 */
{ static int j;
  if (num<=number[j][0]) for ( ; j>=0 && num<number[j][0]; j--) ;
  else   for ( ; j<count && num>number[j][0]; j++) ;
  if (j<0 || j>=count) { j=0; return(0); }
  if ( num==number[j][0] ) return(1);
  else return(0);
}
p _array (void )                                /* 输出矩阵 */
{ int i, j;
  for (i=0; i<4; i++) {
    for (j=1; j<=4; j++) printf ("%6d", array[i][j]);
    printf("\n");
   }
}
```

* 运行结果

There are magic squares with invertable primes as follow:

No. 1	No. 2	No. 3	No. 4	No. 5	No. 6	No. 7	No. 8	No. 9
1193	1193	1193	1193	1193	1193	1193	1193	1193
1009	1021	1201	1229	7043	7253	7253	7457	7963
9221	9029	9209	9001	3697	3821	3851	3821	3407
3191	3911	3911	3191	3911	3319	3319	3719	3911
No. 10	No. 11	No. 12	No. 13	No. 14	No. 15	No. 16	No. 17	No. 18
1193	1193	1193	1193	1193	1193	1193	1193	1193
9001	9011	9013	9029	9041	9209	9221	9257	9257

1229	1669	1669	1021	1399	1201	1009	3821	3851
3191	3911	3911	3911	3371	3911	3191	3319	3319
No. 19	No. 20	No. 21	No. 22	No. 23	No. 24	No. 25	No. 26	No. 27
1193	1193	1193	1193	1733	1733	1733	1733	1733
9479	9661	9661	9923	1069	1283	1283	1487	1949
1381	1109	3109	1009	9491	9521	9551	9521	9601
3917	3911	3911	3191	3371	3319	3319	3719	3371
No. 28	No. 29	No. 30	No. 31	No. 32	No. 33	No. 34	No. 35	No. 36
1733	1733	1913	1913	1913	1913	1913	1913	1913
9227	9931	1009	1021	1033	1069	1069	1201	1229
1021	1409	9221	9029	9497	9161	9161	9209	9001
3719	3911	3911	3191	3191	3191	3391	3191	3911
No. 37	No. 38	No. 39	No. 40	No. 41	No. 42	No. 43	No. 44	No. 45
1913	1913	1913	1913	1913	1913	1913	1913	1913
1439	1619	1901	7207	7949	9001	9001	9029	9173
9781	9601	9209	3221	3301	1229	3299	1021	3389
3917	3191	3391	3719	3191	3911	3911	3191	3391
No. 46	No. 47	No. 48	No. 49	No. 50	No. 51	No. 52	No. 53	No. 54
1913	1913	1933	1933	1933	1933	1933	1933	1933
9209	9221	1283	1283	1619	9029	9133	9833	9871
1201	1009	9521	9551	9601	1091	1789	3719	3319
3191	3911	3719	3719	3191	3191	3391	3191	3391
No. 55	No. 56	No. 57	No. 58	No. 59	No. 60	No. 61	No. 62	No. 63
3191	3191	3191	3191	3191	3191	3191	3191	3191
1009	1009	1021	1091	1201	1229	3301	3719	7027
9221	9923	9029	9029	9209	9001	7949	9833	1223
1193	1193	1913	1933	1913	1193	1913	1933	9173
No. 64	No. 65	No. 66	No. 67	No. 68	No. 69	No. 70	No. 71	No. 72
3191	3191	3191	3191	3191	3191	3191	3191	3191
9001	9029	9161	9209	9221	9341	9497	9601	9601
1229	1021	1069	1201	1009	1879	1033	1619	1619
1193	1913	1913	1913	1193	7193	1913	1913	1933
No. 73	No. 74	No. 75	No. 76	No. 77	No. 78	No. 79	No. 80	No. 81
3319	3319	3319	3319	3319	3319	3371	3371	3371
3821	3821	3851	3851	9521	9551	1399	3821	3821
7253	9257	7253	9257	1283	1283	9041	1259	1559
1193	1193	1193	1193	1733	1733	1193	9133	9133
No. 82	No. 83	No. 84	No. 85	No. 86	No. 87	No. 88	No. 89	No. 90

3371	3371	3371	3371	3391	3391	3391	3391	3391
7229	7841	9491	9601	1789	3319	3389	3821	3821
1201	1259	1069	1949	9133	9871	9173	1259	1559
9173	9173	1733	1733	1933	1933	1913	9173	9173
No. 91	No. 92	No. 93	No. 94	No. 95	No. 96	No. 97	No. 98	No. 99
3391	3391	3719	3719	3719	3719	3719	3719	3911
9161	9209	1021	3221	3821	9521	9521	9551	1009
1069	1901	9227	7207	7457	1283	1487	1283	9221
1913	1913	1733	1913	1193	1933	1733	1933	1913
No. 100	No. 101	No. 102	No. 103	No. 104	No. 105	No. 106	No. 107	No. 108
3911	3911	3911	3911	3911	3911	3911	3911	3911
1021	1109	1201	1229	1409	1669	1669	3109	3299
9029	9661	9209	9001	9931	9011	9013	9661	9001
1193	1193	1193	1913	1733	1193	1193	1193	1913
No. 109	No. 110	No. 111	No. 112	No. 113	No. 114	No. 115	No. 116	No. 117
3911	3911	3911	3911	3911	3911	3911	3911	3911
3407	3527	3527	3697	7529	7529	7547	9001	9029
7963	1283	1583	7043	1283	1583	1283	1229	1021
1193	9133	9133	1193	9133	9133	9173	1913	1193
No. 118	No. 119	No. 120	No. 121	No. 122	No. 123	No. 124	No. 125	No. 126
3911	3911	3911	3917	3917	7193	7193	9133	9133
9209	9221	9749	1381	9781	1831	1879	1259	1283
1201	1009	1831	9479	1439	9749	9341	3821	3527
1193	1913	7193	1193	1913	3911	3191	3371	3911
No. 127	No. 128	No. 129	No. 130	No. 131	No. 132	No. 133	No. 134	No. 135
9133	9133	9133	9133	9173	9173	9173	9173	9173
1283	1559	1583	1583	1201	1223	1259	1259	1283
7529	3821	3527	7529	7229	7027	3821	7841	7547
3911	3371	3911	3911	3371	3191	3391	3371	3911

No. 136
9173
1559
3821
3391

* 问题的进一步讨论

程序中大量技巧是用于尽早发现矛盾，减少循环次数，缩短运行时间。从实际效果看是相当不错的。但目前的程序仍然可以进一步优化。

当四行数设定了前三行后，尚未设定的行就没必要再使用穷举的方法，因为列方向设定好

的三位数字，已经限制了最后一个数字可能的取值。在可逆素数中找出前三位数字与设定好的三位数字相同的素数。这些素数就是在这一列前面已设定好的三位数字的限制下可能的取值。此时每一列上只有不超过四个可能的取值。找出全部各列可能的取值（可能的四位可逆素数），求出它们的交集。若交集为空，即没有共同的可能取值，则列间数据相互矛盾；否则将交集中的数据填入矩阵中就是题目的一个解。

算法可再进一步优化。先穷举一、二和四列的数据，然后用上面的算法来确定第三行的值，这样可进一步缩小穷举的范围，提高运行效率。

分析输出的结果，可以看出本题的基本解只有 17 种，每个解可通过旋转与反射获得同构的其它 7 个解，可以进一步改进程序，只输出 17 个基本解。

* 思考题

用 1 到 16 构成一个四阶幻方，要求任意相邻两个方格中的数字之和均为素数。

第五章　不定方程求整数解

36. 百钱百鸡问题

中国古代数学家张丘建在他的《算经》中提出了一个著名的“百钱百鸡问题”：鸡翁一，值钱五，鸡母一，值钱三，鸡雏三，值钱一，百钱买百鸡，问翁、母、雏各几何？

* 题目分析与算法设计

设鸡翁、鸡母、鸡雏的个数分别为 x、y、z，题意给定共 100 钱要买百鸡，若全买公鸡最多买 20 只，显然 x 的值在 0～20 之间；同理，y 的取值范围在 0～33 之间，可得到下面的不定方程：

$$\begin{cases} 5x+3y+z/3=100 \\ x+y+z=100 \end{cases}$$

所以此问题可归结为求这个不定方程的整数解。

由程序设计实现不定方程的求解，与手工计算不同。在分析确定方程中未知数变化范围的前题下，可通过对未知数可变范围的穷举，验证方程在什么情况下成立，从而得到相应的解。

* 程序说明与注释：

```
#include <stdio.h>
main()
{ int x, y, z, j=0;
  printf("Following are possible plans to buy 100 fowls with 100 Yuan.\n");
  for (x=0; x<=20; x++)                    /* 外层循环控制鸡翁数 x 在 0～20 变化 */
    for (y=0; y<=33; y++) {                /* 内层循环控制鸡母数 y 在 0～33 变化 */
      z =100-x-y;              /* 内外层循环控制下，鸡雏数 z 的值受 x、y 值的制约 */
      if ( z%3==0 && 5*x+3*y+z/3==100)
                                  /* 验证取 z 值的合理性及得到的一组解的合理性 */
          printf("%2d: cock=%2d hen=%2d chicken=%2d\n",++j,x,y,z);
    }
  }
```

* 运行结果

```
Following are possible plans to buy 100 fowls with 100 Yuan.
    1: cock= 0 hen=25 chicken=75
    2: cock= 4 hen=18 chicken=78
    3: cock= 8 hen=11 chicken=81
    4: cock=12 hen= 4 chicken=84
```

* 问题的进一步讨论

这类求解不定方程问题的实现，各层循环的控制变量直接与方程未知数相关，且采用对未

知数的取值范围穷举和组合的方法来覆盖可能得到的全部各组解。能否据题意更合理的设置循环控制条件来减少这种穷举和组合的次数，提高程序的执行速度，请读者考虑。

* 思考题

100匹马驮100担货，大马一匹驮3担，中马一匹驮2担，小马两匹驮1担。试编写程序计算大、中、小马的数目。

37. 爱因斯坦的数学题

爱因斯坦出了一道这样的数学题：有一条长阶梯，若每步跨2阶，则最后剩1阶，若每步跨3阶，则最后剩2阶，若每步跨5阶，则最后剩4阶，若每步跨6阶则最后剩5阶。只有每次跨7阶，最后才正好一阶不剩。请问，这条阶梯共有多少阶？

* 题目分析与算法设计

据题意，阶梯数满足下面一组同余式：

$$x \equiv 1 \pmod 2$$
$$x \equiv 2 \pmod 3$$
$$x \equiv 4 \pmod 5$$
$$x \equiv 5 \pmod 6$$
$$x \equiv 0 \pmod 7$$

* 程序说明与注释

```
main()
{int i=1;                                          /* i为所设的阶梯数 */
 while ( ! ((i%2==1)&&(i%3==2)&&(i%5==4)&&(i%6==5)&&(i%7==0)) )
     ++i;                                          /* 满足一组同余式的判别 */
 printf("Staris _number=%d\n", i);
}
```

* 运行结果

Staris _number=119

* 问题的进一步讨论

此题算法还可考虑求1、2、4、5的最小公倍数n，然后判t(t为n-1)≡0(mod7)是否成立，若不成立则t=t+n，再进行判别，直至选出满足条件的t值。请读者自编程序实现。

38. 换分币

用一元人民币兑换成1分、2分和5分硬币，共有多少种不同的兑换方法。

* 题目分析与算法设计

据题意设i、j、k分别为兑换的1分、2分、5分硬币所具有的钱数(分)，则i、j、k的值应满足：

$$i+j+k=100$$

* 程序说明与注释

```
main()
{ int i,j,k, count=1;
```

```
  printf("There are following small exchange plans for 1 Yuan note:\n");
  for (i=0; i<=100; i++)                    /* i为1分硬币钱数,可取值0,1,2,...,100 */
    for (j=0; j<=100-i; j+=2)               /* j为2分硬币钱数,可取值0,2,4,..,100 */
      for (k=0; k<=100-i-2*j; k+=5)                    /* k为5分硬币钱数 */
        if (i+j+k==100)
          printf(count%4 ? "%d: 1*%d+2*%d+5*%d\t"
              : "%d: 1*%d+2*%d+5*%d\n",count++, i, j/2, k/5);
}
```

* 运行结果

There are following small exchange plans for 1 Yuan note:

1: 1*0+2*0+5*20	2: 1*5+2*0+5*19	3: 1*10+2*0+5*18	4: 1*15+2*0+5*17
5: 1*20+2*0+5*16	6: 1*25+2*0+5*15	7: 1*30+2*0+5*14	8: 1*35+2*0+5*13
9: 1*40+2*0+5*12	10: 1*45+2*0+5*11	11: 1*50+2*0+5*10	12: 1*55+2*0+5*9
13: 1*60+2*0+5*8	14: 1*65+2*0+5*7	15: 1*70+2*0+5*6	16: 1*75+2*0+5*5
17: 1*80+2*0+5*4	18: 1*85+2*0+5*3	19: 1*90+2*0+5*2	20: 1*95+2*0+5*1
21: 1*100+2*0+5*0			

* 思考题

此题若修改条件,要求应换若干2分的,还应换若干1分的,且1分的个数是2分个数的10倍,其余的换成5分的,问每种硬币换多少个?

39. 年龄几何

张三、李四、王五、刘六的年龄成一等差数列,他们四人的年龄相加是26,相乘是880,求以他们的年龄为前4项的等差数列的前20项。

* 题目分析与算法设计

设数列的首项为a,公差为n,则前4项之和为"4*n+6*a",前4项之积为"n*(n+a)*(n+a+a)*(n+a+a+a)"。同时"1<=a<=4","1<=n<=6"。可采用穷举法求出此数列。

* 程序说明与注释

```
main()
{ int n, a, i;
  printf ("The series with equal difference are:\n");
  for (n=1; n<=6; n++)                              /* 公差n取值为1~6 */
    for (a=1; a<=4; a++)                            /* 首项a取值为1~4 */
      if (4*n+6*a==26 && n*(n+a)*(n+a+a)*(n+a+a+a)==880)
                                                    /* 判断结果 */
        for (i=0; i<20; i++)
          printf("%d ", n+i*a);                     /* 输出前20项 */
}
```

* 运行结果

The series with equal difference are:

2 5 8 11 14 17 20 23 26 29 32 35 38 41 44 47 50 53 56 59

40. 三色球问题

若一个口袋中放有 12 个球，其中有 3 个红的，3 个白的和 6 个黑的，问从中任取 8 个共有多少种不同的颜色搭配？

* 题目分析与算法设计

设任取的红球个数为 i，白球个数为 j，则黑球个数为 8－i－j，据题意红球和白球个数的取值范围是 0～3，在红球和白球个数确定的条件下，黑球个数取值应为 8－i－j<＝6。

* 程序说明与注释

```
main()
  { int i, j, count=0;
    printf(" RED BALLL   WHITE BALL   BLACK BALL\n");
    printf("-------------------------------------------------------------------\n");
    for (i=0; i<=3; i++)                    /* 循环控制变量 i 控制任取红球个数 0～3 */
      for (j=0; j<=3; j++)                  /* 循环控制变量 j 控制任取白球个数 0～3 */
        if ((8-i-j)<=6)
          printf(" %2d:      %d      %d      %d\n", ++count, i,j,8-i-j);
}
```

* 运行结果

	RED BALL	WHITE BALL	BLACK BALL
1:	0	2	6
2:	0	3	5
3:	1	1	6
4:	1	2	5
5:	1	3	4
6:	2	0	6
7:	2	1	5
8:	2	2	4
9:	2	3	3
10:	3	0	5
11:	3	1	4
12:	3	2	3
13:	3	3	2

41. 马克思手稿中的数学题

马克思手稿中有一道趣味数学问题：有 30 个人，其中有男人、女人和小孩，在一家饭馆吃饭共花了 50 先令；每个男人花 3 先令，每个女人花 2 先令，每个小孩花 1 先令；问男人、女人和小孩各有几人？

* 题目分析与算法设计

设 x，y，z 分别代表男人，女人和小孩的人数，按题目的要求，可得到下面的方程：

$$\begin{cases} x+y+z=30 & ① \\ 3x+2y+z=50 & ② \end{cases}$$

用程序求此不定方程的非负整数解，可先通过②－①式得：

$$2x+y=20 \qquad ③$$

由③式知，x 变化范围是 0～10。

* 程序说明与注释

```
main()
{ int x, y, z, count=0;
  printf("MEN     WOMEN     CHILDREN\n");
  printf("------------------------------------------------------\n");
  for (x=0;x<=10;x++){
    y=20-2*x;                                   /* x 定值据③式求 y */
    z=30-x-y;                                   /* 由①式求 z */
    if (3*x+2*y+z==50)                          /* 当前得到的一组解是否满足②式 */
      printf(" %2d:     %d     %d     %d\n", ++count, x, y, z);
   }
}
```

* 运行结果

	MEN	WOMEN	CHILDREN
1:	0	20	10
2:	1	18	11
3:	2	16	12
4:	3	14	13
5:	4	12	14
6:	5	10	15
7:	6	8	16
8:	7	6	17
9:	8	4	18
10:	9	2	19
11:	10	0	20

第六章　分数趣题

42. 最大公约数和最小公倍数

求任意两个正整数的最大公约数(GCD)和最小公倍数(LCM)。

* 问题分析与算法设计

手工方式求两个正整数的最大公约数的方法是用辗转相除法,在程序中可以模拟这种方式。

* 程序与程序注释

```
main()
{   int a,b,num1,num2,temp;
    printf("Input a & b:");
    scanf("%d%d", &num1, &num2);
    if (num1>num2) {                                    /* 找出两个数中的较大值 */
        temp=num1;    num1=num2;    num2=temp;          /* 交换两个整数 */
    }
    a=num1; b=num2;
    while ( b! =0 ) {                                   /* 采用辗转相除法求最大公约数 */
        temp=a%b;
        a=b;
        b=temp;
    }
    printf(" The GCD of %d and %d is: %d\n", num1,num2,a);
                                                        /* 输出最大公约数 */
    printf(" The LCM of them is: %d\n", num1 * num2/a);    /* 输出最小公倍数 */
}
```

* 运行结果

```
1. Input a & b: 20 55
     The GCD of 20 and 55 is: 5
     The LCM of them is: 220
2. Input a & b: 17 71
     The GCD of 17 and 71 is: 1
     The LCM of them is: 1207
3. Input a & b: 24 88
     The GCD of 24 and 88 is: 8
     The LCM of them is: 264
```

4. Input a & b: 35 85

The GCD of 35 and 85 is: 5

The LCM of them is: 595

* 思考题

求一个最小的正整数，这个正整数被任意 n(2≤n≤10)除都是除不尽的，而且余数总是(n－1)。例如：被9除时的余数为8。要求设计一个算法，不允许枚举与除2、除3、……、除9、除10有关的命令，求出这个正整数。 (答案：2519)

43. 分数比较

比较两个分数的大小。

* 问题分析与算法设计

人工方式下比较分数大小最常用的方法是：进行分数的通分后比较分子的大小。可以编程模拟手工方式。

* 程序与程序注释

```
main()
{   int i, j, k, l, m, n;
    printf("Input two FENSHU :\n");
    scanf("%d/%d,%d/%d", &i, &j, &k, &l);                    /* 输入两个分数 */
    m = zxgb(j,l)/j * i;                                  /* 求出第一个分数通分后的分子 */
    n = zxgb(j,l)/l * k;                                  /* 求出第二个分数通分后的分子 */
    if(m>n)          printf("%d/%d > %d/%d\n",i,j,k,l);    /* 比较分子的大小 */
    else if (m==n)   printf("%d/%d = %d/%d\n",i,j,k,l);    /* 输出比较的结果 */
        else         printf("%d/%d < %d/%d\n",i,j,k,l);
}
zxgb(a,b)
    int a,b;
{   long int c;
    int d;
    if (a<b) c=a, a=b, b=c;                               /* 若a<b,则交换两变量的值 */
    for ( c=a*b; b!=0; ) {                                /* 用辗转相除法求a和b的最大公约数 */
        d=b;    b=a%b;    a=d;
    }
    return ( (int) c/a);                                  /* 返回最小公倍数 */
}
```

* 运行结果

输入：4/5,6/7　　输出：4/5 < 6/7

输入：8/4,16/32　　输出：8/4 > 16/32

输入：16/32,4/8　　输出：16/32 = 4/8

44. 分数之和

求这样的四个自然数 p,q,r,s(p<=q<=r<=s),使得以下等式成立:

$$\frac{1}{p}+\frac{1}{q}+\frac{1}{r}+\frac{1}{s}=1$$

* 问题分析与算法设计

若规定 p<=q<=r<=s,将原式通分、化简并整理后得到:

$$2<=p<5 \qquad p<=q<7 \qquad q<r<13$$

采用最简单的穷举法可以很方便地求解。

* 程序与程序注释

```
main()
{   int p, q, r, s, count=0;
  printf("The 4 fractions which sum is equal 1 are:\n");
  for (p=2;p<5;p++)                                          /* 穷举分母 */
    for (q=p;q<7;q++)
      for (r=q;r<13;r++)
        if (p*q*r-q*r-p*r-p*q!=0) {
          s=(p*q*r)/(p*q*r-q*r-p*r-p*q);                    /* 求出 s 的值 */
          if (!((p*q*r)%(p*q*r-q*r-p*r-p*q)) && s>=r)
            printf("[%2d] 1/%d+1/%d+1/%d+1/%d=1\n",++count,p,q,r,s);
                                                             /* 输出结果 */
        }
}
```

* 运行结果

```
The 4 fractions which sum is equal 1 are:
 [ 1] 1/2+1/3+1/7+1/42=1
 [ 2] 1/2+1/3+1/8+1/24=1
 [ 3] 1/2+1/3+1/9+1/18=1
 [ 4] 1/2+1/3+1/10+1/15=1
 [ 5] 1/2+1/3+1/12+1/12=1
 [ 6] 1/2+1/4+1/5+1/20=1
 [ 7] 1/2+1/4+1/6+1/12=1
 [ 8] 1/2+1/4+1/8+1/8=1
 [ 9] 1/2+1/5+1/5+1/10=1
 [10] 1/2+1/6+1/6+1/6=1
 [11] 1/3+1/3+1/4+1/12=1
 [12] 1/3+1/3+1/6+1/6=1
 [13] 1/3+1/4+1/4+1/6=1
 [14] 1/4+1/4+1/4+1/4=1
```

* 思考题

将1、2、3、4、5、6、7、8、9九个数字分成以下三种分数形式之一，每个数字只能用一次，使得该分数刚好等于一个整数。

$$N = X\frac{XXXX}{XXXX} \qquad N = X\frac{XXXXX}{XXX} \qquad N = XX\frac{XXXX}{XXX}$$

例如：

$$100 = 3\frac{69258}{714} = 81\frac{5648}{297}$$

求所有满足条件的表示形式。

（参考答案：某些自然数没有这种表示形式，如：1、2、3、4、15、18等。此外整数100有11种满足条件的表示形式；89的表示形式最多，共36种；三种形式中，最大可表示的整数为794。）

45. 将真分数分解为埃及分数

分子为1的分数称为埃及分数。现输入一个真分数，请将该分数分解为埃及分数。如：8/11=1/2+1/5+1/55+1/110。

* 问题分析与算法设计

若真分数的分子a能整除分母b，则真分数经过化简就可以得到埃及分数。若真分数的分子不能整除分母，则可以从原来的分数中分解出一个分母为b/a+1的埃及分数。用这种方法将余数部分反复分解，最后可得到结果。

* 程序与程序注释

```
main()
{   long int a, b, c;
    printf("Please enter a optional fraction (a/b):");
    scanf("%ld/%ld", &a, &b);                          /* 输入分子a和分母b */
    printf(" It can be decomposed to :" );
    while(1) {
      if (b%a)                  /* 若分子不能整除分母 */
        c=b/a+1;                /* 则分解出一个分母为b/a+1的埃及分数 */
      else {   c=b/a; a=1; }    /* 否则，输出化简后的真分数(埃及分数) */
      if (a==1) {
        printf ("1/%ld\n",c);
        break;                  /* a为1标志结束 */
      }
      else
        printf ("1/%ld + ",c);
      a=a*c-b;                                         /* 求出余数的分子 */
      b=b*c;                                           /* 求出余数的分母 */
      if (a==3)                                  /* 若余数为3，输出最后两个埃及分数 */
        {   printf("1/%ld + 1/%ld\n", b/2, b); break; }
```

```
    }
}
```

* 运行结果

```
1. Please enter a optional fraction (a/b):1/6
      It can be decomposed to :1/6
2. Please enter a optional fraction (a/b): 20/33
      It can be decomposed to :1/2 + 1/10 + 1/165
3. Please enter a optional fraction (a/b): 10/89
      It can be decomposed to :1/9 + 1/801
4. Please enter a optional fraction (a/b): 19/99
      It can be decomposed to :1/6 + 1/40 + 1/3960
5. Please enter a optional fraction (a/b): 8/89
      It can be decomposed to :1/11 + 1/957
```

46. 列出真分数序列

按递增顺序依次列出所有分母为40,分子小于40的最简真分数。

* 问题分析与算法设计

对分子采用穷举法,利用求最大公约数的方法,判断分子与40是否构成真分数。

* 程序与程序注释

```
main()
{   int i, num1,num2,temp;
    printf("The fraction series with demominator 40 is :\n");
    for (i=1; i<=40; i++) {                          /* 穷举40以内的全部分子 */
        num1=40;
        num2=i;
        while ( num2!=0) {                           /* 采用辗转相除法求出最大公约数 */
          temp=num1%num2;
          num1=num2;
          num2=temp;
        }
        if (num1==1)                                 /* 若最大公约数为1,则为最简真分数 */
          printf("%d/40\n",i);
    }
  }
```

* 运行结果

```
The fraction series with demominator 40 is :
  1/40     3/40     7/40     9/40    11/40    13/40    17/40    19/40
 21/40    23/40    27/40    29/40    31/40    33/40    37/40    39/40
```

* 思考题

按递增顺序依次列出所有分母小于等于40的最简真分数。

47. 计算分数的精确值

使用数组精确计算 M/N(0<M<N<=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)。

* 问题分析与算法设计

由于计算机内字长有限,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,就必须自行设计实现算法。

为了实现高精度计算,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来表示一个高精度的计算结果。

进行除法运算时可以模拟人的手工操作,即每次求出商的一位后,将余数乘以10,再计算商的下一位,重复以上过程。当某次计算后的余数为0时,表示 M/N 为有限不循环小数;当某次计算后的余数与前面的某个余数相同时,则 M/N 为无限循环小数,从该余数第一次出现之后所求得的各位就是小数的循环节。

程序具体实现时,采用了数组和其它一些技巧来保存除法运算所得到的余数和商的各位数。

* 程序与程序注释

```
#include <stdio.h>
int remainder[101], quotient[101];
                    /* remainder:存放除法的余数     quotient:依次存放商的每一位 */
main ()
{   int m, n, i, j;
    printf("Please input a fraction (m/n) (<0<m<n<=100):");
    scanf("%d/%d", &m,&n);                                  /* 输入被除数和除数 */
    printf("%d/%d it's accuracy value is: 0.", m, n);
    for (i=1; i<=100; i++) {                                /* i:商的位数 */
      remainder[m]=i;      /* m:除的余数  remainder[m]:该余数对应的商的位数 */
      m *= 10;                                              /* 余数扩大10倍 */
      quotient[i]=m/n;                                      /* 商 */
      m = m%n;                                              /* 求余数 */
      if (m==0) {                                   /* 余数为0则表示是有限小数 */
        for ( j=1; j<=i; j++) printf("%d", quotient[j]);    /* 输出商 */
        break;                                              /* 退出循环 */
      }
      if ( remainder[m]!=0 ) {          /* 若该余数对应的位在前面已经出现过 */
        for ( j=1; j<=i; j++) printf("%d", quotient[j]);   /* 则输出循环小数 */
        printf
```

```
            ("\n\tand it is a infinite cyclic fraction from %d\n", remainder[m]);
        printf ("\tdigit to %d digit after decimal point.\n", i);
                                                    /* 输出循环节的位置 */
        break;                                      /* 退出 */
      }
    }
}
```

* 运行结果

① Please input a fraction (m/n) (<0<m<n<=100): 1/40
1/40 it's accuracy value is: 0.025

② Please input a fraction (m/n) (<0<m<n<=100): 1/6
1/6 it's accuracy value is: 0.16
and it is a infinite cyclic fraction from 2
digit to 2 digit after decimal point.

③ Please input a fraction (m/n) (<0<m<n<=100): 25/95
25/95 it's accuracy value is: 0.263157894736842105
and it is a infinite cyclic fraction from 1
digit to 18 digit after decimal point.

④ Please input a fraction (m/n) (<0<m<n<=100): 16/19
16/19 it's accuracy value is: 0.842105263157894736
and it is a infinite cyclic fraction from 1
digit to 18 digit after decimal point.

⑤ Please input a fraction (m/n) (<0<m<n<=100): 23/87
23/87 it's accuracy value is: 0.264367816091954022988505747 1
and it is a infinite cyclic fraction from 1
digit to 28 digit after decimal point.

* 思考题

使用数组实现计算 M×N 的精确值。

第七章　逻辑推理与判断

逻辑推理能力是在智力测验或智力游戏中取胜的关键，它要求人们利用已知的条件，通过分析和判断，得出正确的答案。能否让计算机来解决逻辑推理问题呢？回答是肯定的。进行逻辑推理的前提是正确理解题意，因此如何让计算机正确地理解题意是让计算机进行正确推理的关键。

逻辑问题与常见的数学问题不同，它需要使用逻辑表达式来表示各种逻辑关系。

48. 新娘和新郎

三对情侣参加婚礼，三个新郎为A、B、C，三个新娘为X、Y、Z。有人不知道谁和谁结婚，于是询问了六位新人中的三位，但听到的回答是这样的：A说他将和X结婚；X说她的未婚夫是C；C说他将和Z结婚。这人听后知道他们在开玩笑，全是假话。请编程找出谁将和谁结婚。

＊ 问题分析与算法设计

将A、B、C三人用1、2、3表示，将X和A结婚表示为“X＝1”，将Y不与A结婚表示为“Y!＝1”。按照题目中的叙述可以写出表达式：

x!＝1	A不与X结婚
x!＝3	X的未婚夫不是C
z!＝3	C不与Z结婚

题意还隐含着X、Y、Z三个新娘不能结为配偶，则有：

x! ＝ y 且 x! ＝ z 且 y! ＝ z

穷举各种可能情况，代入上述表达式中进行推理运算，若假设的情况使上述表达式计算的结果均为“真”，则假设情况就是正确的结果。

＊ 程序与程序注释

```
main ()
{   int x, y, z;
  for (x=1; x<=3; x++)                                  /* 穷举X的全部可能配偶 */
    for (y=1; y<=3; y++)                                /* 穷举Y的全部可能配偶 */
      for (z=1; z<=3; z++)                              /* 穷举Z的全部可能配偶 */
        if (x!=1 && x!=3 && z!=3 && x!=y && x!=z && y!=z) {
                                                        /* 判断配偶是否满足题意 */
          printf ("X will marry to %c.\n", 'A'+x-1);    /* 打印判断结果 */
          printf ("Y will marry to %c.\n", 'A'+y-1);
          printf ("Z will marry to %c.\n", 'A'+z-1);
        }
}
```

＊ 运行结果

X will marry to B.　　(X 与 B 结婚)
Y will marry to C.　　(Y 与 C 结婚)
Z will marry to A.　　(Z 与 A 结婚。)

49. 委派任务

某侦察队接到一项紧急任务，要求在 A、B、C、D、E、F 六个队员中尽可能多地挑若干人，但有以下限制条件：

①A 和 B 两人中至少去一人；
②A 和 D 不能一起去；
③A、E 和 F 三人中要派两人去；
④B 和 C 都去或都不去；
⑤C 和 D 两人中去一个；
⑥若 D 不去，则 E 也不去。

问应当让哪几个人去？

* 问题分析与算法设计

用 A、B、C、D、E、F 六个变量表示六个人是否去执行任务的状态，变量的值为1，则表示该人去；变量的值为0，则表示该人不参加执行任务。根据题意可写出表达式：

a+b> 1	A 和 B 两人中至少去一人；
a+d!=2	A 和 D 不能一起去；
a+e+f==2	A、E 和 F 三人中要派两人去；
b+c==0 或 b+c==2	B 和 C 都去或都不去；
c+d==1	C 和 D 两人中去一个；
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”的关系。穷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”的情况就是正确的结果。

* 程序与程序注释

```
main ()
{   int a, b, c, d, e, f;
  for (a=1; a>=0; a--)                          /* 穷举每个人是否去的所有情况 */
    for (b=1; b>=0; b--)                                /* 1:去    0:不去 */
      for (c=1; c>=0; c--)
        for (d=1; d>=0; d--)
          for (e=1; e>=0; e--)
              for (f=1; f>=0; f--)
                if ( a+b>=1 && a+d!=2 && a+e+f==2
                    && (b+c==0 || b+c==2) && c+d==1
                    && (d+e==0 || d==1) )
                { printf ("A will%s be assigned.\n", a ? "" :" not");
                  printf ("B will%s be assigned.\n", b ? "" :" not");
```

```
        printf ("C will%s be assigned.\n", c ? "" :" not");
        printf ("D will%s be assigned.\n", d ? "" :" not");
        printf ("E will%s be assigned.\n", e ? "" :" not");
        printf ("F will%s be assigned.\n", f ? "" :" not");
    }
}
```

* 运行结果

A will be assigned. (去)
B will be assigned. (去)
C will be assigned. (去)
D will not be assigned. (不去)
E will not be assigned. (不去)
F will be assigned. (去)

* 思考题

某参观团按以下条件限制从A、B、C、D、E五个地方中选定若干参观点：

①如去A则必须去B；
②D、E两地只能去一地；
③B、C两地只能去一地；
④C、D两地都去或都不去；
⑤若去E地，A、D也必去。

问该团最多能去哪几个地方？

50. 谁在说谎

张三说李四在说谎，李四说王五在说谎，王五说张三和李四都在说谎。现在问：这三人中到底谁说的是真话，谁说的是假话？

* 问题分析与算法设计

分析题目，每个人都有可能说的是真话，也有可能说的是假话，这样就需要对每个人所说的话进行分别判断。假设三个人所说的话的真假用变量A、B、C表示，等于1表示该人说的是真话；等于0表示该人说的是假话。由题目可以得到：

·张三说李四在说谎　　张三说的是真话：a==1 && b==0
　　或　张三说的是假话：a==0 && b==1
·李四说王五在说谎　　李四说的是真话：b==1 && c==0
　　或　李四说的是假话：b==0 && c==1
·王五说张三和李四都在说谎　　王五说的是真话：c==1 && a+b==0
　　或　王五说的是假话：c==0 && a+b!=0

上述三个条件之间是“与”关系。将表达式进行整理就可得到C语言的表达式：

(a&&!b || !a&&b) && (b&&!c || !b&&c) && (c&&a+b==0 || !c&&a+b!=0)

穷举每个人说真话或说假话的各种可能情况，代入上述表达式中进行推理运算，使上述表达式均为“真”的情况就是正确的结果。

* 程序与程序注释

```
main ()
{ int a,b,c;
  for (a=0; a<=1; a++)
    for( b=0; b<=1; b++)
      for (c=0; c<=1; c++)
        if( (a&&!b || !a&&b)
          && ( b&&!c || !b&&c)
          && ( c&&a+b==0 || !c&&a+b!=0) )
          {  printf("Zhangsan told a %s.\n", a ? "truth" : "lie");
             printf("Lisi told a %s.\n", b ? "truth" : "lie");
             printf("Wangwu told a %s.\n", c ? "truth" : "lie");
          }
}
```

* 运行结果

```
Zhangsan told a lie.        (张三说假话)
Lisi told a truth.          (李四说真话)
Wangwu told a lie.          (王五说假话)
```

51. 谁是窃贼

公安人员审问四名窃贼嫌疑犯。已知,这四人当中仅有一名是窃贼,还知道这四个人中每人要么是诚实的,要么总是说谎的。在回答公安人员的问题中:

甲说:“乙没有偷,是丁偷的。”

乙说:“我没有偷,是丙偷的。”

丙说:“甲没有偷,是乙偷的。”

丁说:“我没有偷。”

请据这四人的答话判断谁是盗窃者。

* 问题分析与算法设计

假设用A、B、C、D分别代表四个人,变量的值为1代表该人是窃贼。

由题目已知:四人中仅有一名窃贼,且这四个人中的每个人要么说真话,要么说假话,而由于甲、乙、丙三人都说了两句话:“×没偷,×偷了”,故不论该人是否说谎,他提到的两人之中必有一人是小偷。故在列条件表达式时,可以不关心谁说谎,谁说实话。这样,可以列出下列条件表达式:

甲说:“乙没有偷,是丁偷的。”	B+D=1
乙说:“我没有偷,是丙偷的。”	B+C=1
丙说:“甲没有偷,是乙偷的。”	A+B=1
丁说:“我没有偷。”	A+B+C+D=1

其中丁只说了一句话,无法判定其真假,表达式反映了四个人中仅有一名是窃贼的条件。

* 程序与程序注释

```
main()
{   int i, j, a[4], n;
  for (i=0; i<4; i++) {                                    /* 假定只有第 i 个人为窃贼 */
    for (j=0; j<4; j++)                                    /* 将第 i 个人设置为1表示窃贼,其余为0 */
      if (j==i) a[j]=1;
      else      a[j]=0;
    if (a[3]+a[1]==1 && a[1]+a[2]==1 && a[0]+a[1]==1) {
                                                           /* 判断条件是否成立 */
      printf("The thief is");                                  /* 成立 */
      for (j=0; j<=3; j++)                                 /* 输出计算结果 */
        if(a[j]) printf(" %c.",j+'A');
      printf("\n");
    }
  }
}
```

* 运行结果

The thief is B.　　(乙为窃贼。)

52. 黑与白

有 A、B、C、D、E 五人,每人额头上都贴了一张或黑或白的纸。五人对坐,每人都可以看到其他人额头上的纸的颜色,但都不知道自己额头上的纸的颜色。五人相互观察后,

A 说:"我看见有三人额头上贴的是白纸,一人额头上贴的是黑纸。"

B 说:"我看见其他四人额头上贴的都是黑纸。"

C 说:"我看见有一人额头上贴的是白纸,其他三人额头上贴的是黑纸。"

D 说:"我看见四人额头上贴的都是白纸。"

E 什么也没说。

现在已知额头贴黑纸的人说的都是谎话,额头贴白纸的人说的都是实话。问这五人谁的额头是贴白纸,谁的额头是贴黑纸?

* 问题分析与算法设计

假设变量 A、B、C、D、E 表示每个人额头上所贴纸的颜色,0代表是黑色,1代表是白色。根据题目中 A、B、C、D 四人所说的话可以总结出下列关系:

A 说: a&&b+c+d+e==3 || !a&&b+c+d+e!=3

B 说: b&&a+c+d+e==0 || !b&&a+c+d+e!=0

C 说: c&&a+b+d+e==1 || !c&&a+b+d+e!=1

D 说: d&&a+b+c+e==4 || !d&&a+b+c+e!=4

穷举每个人额头所贴纸的颜色的所有可能的情况,代入上述表达式中进行推理运算,使上述表达式为"真"的情况就是正确的结果。

* 程序与程序注释

```
main()
```

```
{   int a, b, c, d, e;
  for ( a=0; a<=1; a++)                                  /* 黑色:0    白色:1 */
    for (b=0; b<=1; b++)                          /* 穷举五个人额头贴纸的全部可能 */
      for (c=0; c<=1; c++)
        for (d=0; d<=1; d++)
          for (e=0; e<=1; e++)
            if ( (a&&b+c+d+e==3 || !a&&b+c+d+e!=3)
                                                        /* 判断是否符合题意 */
                && (b&&a+c+d+e==0 || !b&&a+c+d+e!=0)
                && (c&&a+b+d+e==1 || !c&&a+b+d+e!=1)
                && (d&&a+b+c+e==4 || !d&&a+b+c+e!=4) )
              {   printf("A is pasted a piece of %s paper on his forehead.\n",
                                                    a ? "white":"black");
                  printf("B is pasted a piece of %s paper on his forehead.\n",
                                                    b ? "white":"black");
                  printf("C is pasted a piece of %s paper on his forehead.\n",
                                                    c ? "white":"black");
                  printf("D is pasted a piece of %s paper on his forehead.\n",
                                                    d ? "white":"black");
                  printf("E is pasted a piece of %s paper on his forehead.\n",
                                                    e ? "white":"black");
              }
}
```

* 运行结果

```
A is pasted a piece of black paper on his forehead.      (黑)
B is pasted a piece of black paper on his forehead.      (黑)
C is pasted a piece of white paper on his forehead.      (白)
D is pasted a piece of black paper on his forehead.      (黑)
E is pasted a piece of white paper on his forehead.      (白)
```

53. 谜语博士的难题(1)

诚实族和说谎族是来自两个荒岛的不同民族,诚实族的人永远说真话,而说谎族的人永远说假话。谜语博士是个聪明的人,他要来判断所遇到的人分别是哪个民族的。

谜语博士遇到三个人,知道他们可能是来自诚实族或说谎族的。为了调查这三个人是什么族的,博士分别问了他们问题,这是他们的对话:

问第一个人:"你们是什么族?",答:"我们之中有两个来自诚实族。"第二个人说:"不要胡说,我们三个人中只有一个诚实族的。"第三个人听了第二个人的话后说:"对,就是只有一个诚实族的。"

请根据他们的回答判断他们分别是哪个族的。

* 问题分析与算法设计

假设这三个人分别为A、B、C，若说谎其值为0，若诚实其值为1。根据题目中三个人的话可分别列出：

第一个人：a&&a+b+c==2 || !a&&a+b+c!=2

第二个人：b&&a+b+c==1 || !b&&a+b+c!=1

第三个人：c&&a+b+c==1 || !c&&a+b+c!=1

利用穷举法，可以很容易地推出正确的结果。

* 程序与程序注释

```
main()
{   int a, b, c;
  for ( a=0; a<=1; a++)                    /* 穷举每个人是说谎还是诚实的全部情况 */
    for (b=0; b<=1; b++)                        /* 说谎:0    诚实:1 */
      for (c=0; c<=1; c++)
        if ( (a&&a+b+c==2 || !a&&a+b+c!=2)          /* 判断是否满足题意 */
          && (b&&a+b+c==1 || !b&&a+b+c!=1)
          && (c&&a+b+c==1 || !c&&a+b+c!=1) )
        {  printf("A is a %s.\n", a ? "honest" : "lier");          /* 输出判断结果 */
           printf("B is a %s.\n", b ? "honest" : "lier");
           printf("C is a %s.\n", c ? "honest" : "lier");
         }
}
```

* 运行结果

A is a lier.　　(说谎族)

B is a lier.　　(说谎族)

C is a lier.　　(说谎族)

* 思考题

谜语博士遇到四个人，知道他们可能是来自诚实族或说谎族的。为了调查这四个人是什么族的，博士照例进行询问："你们是什么族的？"

第一人说："我们四人全都是说谎族的。"

第二人说："我们之中只有一人是说谎族的。"

第三人说："我们四人中有两个是说谎族的。"

第四人说："我是诚实族的。"

问自称是"诚实族"的第四个人是否真是诚实族的？

(答案：第四个人是诚实族的。)

54. 谜语博士的难题(2)

两面族是荒岛上的一个新民族，他们的特点是说话真一句假一句且真假交替。如果第一句为真，则第二句就是假的；如果第一句为假的，则第二句就是真的。但是第一句是真是假没有规律。

谜语博士遇到三个人，知道他们分别来自三个不同的民族：诚实族、说谎族和两面族。三人并肩站在博士面前。

博士问左边的人："中间的人是什么族的？"，左边人回答："诚实族的"。

博士问中间的人："你是什么族的？"，中间人回答："两面族的"。

博士问右边的人："中间的人究竟是什么族的？"，右边人回答："说谎族的"。

请问：这三个人都是哪个民族的？

* 问题分析与算法设计

这个问题是两面族问题中最基本的问题，它比前面只有诚实族和说谎族的问题要复杂。解题时要使用变量将这三个民族分别表示出来。

令：变量 A=1表示：左边的人是诚实族的(用 C 语言表示为 A)；

变量 B=1表示：中间的人是诚实族的(用 C 语言表示为 B)；

变量 C=1表示：右边的人是诚实族的(用 C 语言表示为 C)；

变量 AA=1表示：左边的人是两面族的(用 C 语言表示为 AA)；

变量 BB=1表示：中间的人是两面族的(用 C 语言表示为 BB)；

变量 CC=1表示：右边的人是两面族的(用 C 语言表示为 CC)；

则：左边的人是说谎族可以表示为：A!=1且 AA!=1　　(不是诚实族和两面族的人)

用 C 语言表示为：!A && !AA

中间的人是说谎族可以表示为：B!=1且 BB!=1

用 C 语言表示为：!B && !BB

右边的人是说谎族可以表示为：C!=0且 CC!=1

用 C 语言表示为：!C && !CC

根据题目中"三人来自三个民族"的条件，可以列出：

a+aa!=2 && b+bb!=2 && c+cc!=2　且　a+b+c==1 && aa+bb+cc==1

根据左边人的回答可以推出：若他是诚实族，则中间的人也是诚实族；若他不是诚实族，则中间的人也不是诚实族。以上条件可以表示为：

a&&!aa&&b&&!bb || !a&&!b

根据中间人的回答可以知道：他不是诚实族的人。则可以用下列逻辑式表示：

!b

根据右边人的回答可以推出：若他是诚实族，则中间人是说谎族的；若他是说谎族的，中间的人就不是说谎族的(是诚实族或两面族的)；若右边的人是两面族的，则他的回答对问题的答案无关紧要。以上条件可以用下列逻辑表达式表示：

c&&!b&&!bb || (!c&&!cc)&&(b||bb) || !c&&cc

将全部逻辑条件联合在一起，利用穷举的方法求解，凡是使上述条件同时成立的变量取值就是题目的答案。

* 程序与程序注释

```
main ()
{  int a, b, c, aa, bb, cc;
   for (a=0; a<=1; a++)                                  /* 穷举全部情况 */
    for (b=0; b<=1; b++)
```

```
    for (c=0; c<=1; c++)
      for (aa=0; aa<=1; aa++)
        for (bb=0; bb<=1; bb++)
          for (cc=0; cc<=1; cc++)
            if ( a+aa!=2 && b+bb!=2 && c+cc!=2  &&    /* 判断逻辑条件 */
                 a+b+c==1 && aa+bb+cc==1        &&
                 (a&&!aa&&b&&!bb || !a&&!b)     &&
                 !b                             &&
                 (c&&!b&&!bb || (!c&&!cc)&&(b||bb) || !c&&cc ) )
              {  printf ("The man stand on left is a %s.\n",
                       aa ? "double-dealer" : (a ? "honest" : "lier") );
                 printf ("The man stand on center is a %s.\n",
                       bb ? "double-dealer" : (b ? "honest" : "lier") );
                 printf ("The man stand on right is a %s.\n",
                       cc ? "double-dealer" : (c ? "honest" : "lier") );
              }                                    /* 输出最终的推理结果 */
}
```

* 运行结果

```
The man stand on left is a double-dealer.      (左边的人是两面族的)
The man stand on center is a lier.             (中间的人是说谎族的)
The man stand on right is a honest.            (右边的人是诚实族的)
```

* 思考题

谜语博士遇到三个人,便问第一个人:"你是什么族的?",答:"诚实族的。"问第二个人:"你是什么族的?",答:"说谎族的。"博士又问第二个人:"第一个人真的是诚实族的吗?",答:"是的。"问第三个人:"你是什么族的?",答:"诚实族的。"博士又问第三个人:"第一个人是什么族的?",答:"两面族的。"

请判断这三个人到底是哪个民族的?

(答案:第一人是诚实族,第二人是两面族,第三人是说谎族。)

55. 哪个大夫哪天值班

医院有 A、B、C、D、E、F、G 七位大夫,在一星期内(星期一至星期天)每人要轮流值班一天。现在已知:

A 大夫比 C 大夫晚一天值班;

D 大夫比 E 大夫晚二天值班;

B 大夫比 G 大夫早三天值班;

F 大夫的值班日在 B 和 C 大夫的中间,且是星期四;

请确定每天究竟是哪位大夫值班?

* 问题分析与算法设计

由题目可推出如下已知条件:

·F 是星期四值班；

·B 值班的日期在星期一至星期三，且三天后是 G 值班；

·C 值班的日期在星期五至星期六，且一天后是 A 值班；

·E 两天后是 D 值班；E 值班的日期只能在星期一至星期三；

在编程时用数组元素的下标1到7表示星期一到星期天，用数组元素的值1到7分别表示 A～F 七位大夫。

* 程序与程序注释

```
int a[8];
char *day[]
    ={"","MONDAY","TUESDAY","WEDNESDAY","THURSDAY",
        "FRIDAY","SATURDAY","SUNDAY"};                    /* 建立星期表 */
main()
{  int i,t,j;
   a[4]=6;                                                /* 星期四是F值班 */
   for (i=1; i<=3; i++) {
       a[i]=2;                                            /* 假设B值班的日期 */
       if (!a[i+3]) a[i+3]=7;                  /* 若三天后无人值班则安排G值班 */
       else {   a[i]=0; continue; }                       /* 否则B值班的日期不对 */
       for (t=1; t<=3; t++) {                             /* 假设E值班的时间 */
           if (!a[t]) a[t]=5;                        /* 若当天无人值班则安排E值班 */
           else continue;
           if (!a[t+2]) a[t+2]=4;                  /* 若E值班两天后无人值班则应为D */
           else {   a[t]=0; continue; }                   /* 否则E值班的日期不对 */
           for (j=5; j<7; j++) {
             if (!a[j]) a[j]=3;                        /* 若当天无人值班则安排C值班 */
             else continue;
            if (!a[j+1]) a[j+1]=1;                /* C之后一天无人值班则应当是A值班 */
             else {   a[j]=0; continue; }                 /* 否则A值班日期不对 */
             for (i=1; i<=7; i++)                         /* 安排完毕输出结果 */
               printf("Doctor %c is on duty %s.\n", 'A'+a[i]-1, day[i]);
             exit(0);
           }
       }
    }
}
```

* 运行结果

```
Doctor E is on duty MONDAY.       (星期一:E)
Doctor B is on duty TUESDAY.      (星期二:B)
Doctor D is on duty WEDNESDAY.    (星期三:D)
```

Doctor F is on duty THURSDAY. （星期四：F）
Doctor G is on duty FRIDAY. （星期五：G）
Doctor C is on duty SATURDAY. （星期六：C）
Doctor A is on duty SUNDAY. （星期日：A）

* 思考题

在本题的求解过程中，我们只考虑了一星期之内的情况，没有考虑跨周的情况。对于“B 大夫比 G 大夫早三天值班”的条件只是简单的认为是在同一周之内早三天。若考虑跨周的情况就可能出现：B 大夫星期一值班，而 G 大夫是上周的星期五。同样，对“F 大夫的值班日在 B 和 C 大夫的中间”这个条件，也可以扩展为：“只要 F 大夫的值班日在 B 和 C 大夫的中间就可以”。

请考虑允许跨周的情况下，可能的值班时间安排表。

56. 区分旅客国籍

在一个旅馆中住着六个不同国籍的人，他们分别来自美国、德国、英国、法国、俄罗斯和意大利。他们的名字叫 A、B、C、D、E 和 F。名字的顺序与上面的国籍不一定是相互对应的。现在已知：

①A 和美国人是医生。
②E 和俄罗斯人是教师。
③C 和德国人是技师。
④B 和 F 曾经当过兵，而德国人从未参过军。
⑤法国人比 A 年龄大；意大利人比 C 年龄大。
⑥B 同美国人下周要去西安旅行，而 C 同法国人下周要去杭州度假。

试问由上述已知条件，A、B、C、D、E 和 F 各是哪国人？

* 问题分析与算法设计

首先进行题目分析，尽可能利用已知条件，确定谁不是哪国的人。

由①、②、③可知：A 不是美国人，E 不是俄罗斯人，C 不是德国人。另外因为 A 与俄、德人的职业不同，E 与美、德人的职业不同，C 与美、俄人的职业不同，故 A 不是俄罗斯人或德国人，E 不是美国人或德国人，C 不是美国人或俄罗斯人。

由④和⑤可知 B 和 F 不是德国人，A 不是法国人，C 不是意大利人。

由⑥可知 B 不是美国人，也不是法国人（因 B 与法国人下周的旅行地点不同）；C 不是法国人。

将以上结果汇总可以得到下列条件矩阵：

	美（医生）	英	法	德（技师）	意	俄（教师）
A（医生）	×		×	×		×
B	×		×	×		
C（技师）	×		×	×	×	×
D						
E（教师）	×			×		×
F				×		

根据此表使用消去法求解，可方便地得到问题的答案。

将条件矩阵输入计算机，用程序实现消去算法是很容易的。

* 程序与程序注释

```
char *m[7]={  " ","U.S.","U.K.","FRANCE","GER","ITALY","EUSSIAN"} ;
                                                          /* 国名 */
main()
{  int a[7][7], i, j, k, n, t, e, x, y;
   char b;
   for (i=0; i<7; i++)                                    /* 初始化条件矩阵 */
     for (j=0; j<7; j++)          /* 行为人，列为国家，元素的值表示某人是该国人 */
      a[i][j]=j;
   for (i=1; i<7; i++)   /* 条件矩阵每一列的第0号元素作为该列数据处理的标记 */
      a[0][i] = 1;                                        /* 标记该列尚未处理 */
   a[1][1]=a[2][1]=a[3][1]=a[5][1]=0;          /* 输入条件矩阵中的各种条件 */
   a[1][3]=a[2][3]=a[3][3]=0;                          /* 0表示不是该国的人 */
   a[1][4]=a[2][4]=a[3][4]=a[5][4]=a[6][4]=0;
   a[3][5]=0;
   a[1][6]=a[3][6]=a[5][6]=0;
   while ( a[0][1]+a[0][2]+a[0][3]+a[0][4]+a[0][5]+a[0][6]>0) {
                                          /* 当所有六列均处理完毕后退出循环 */
       for (i=1; i<7; i++)                                /* i:列坐标 */
         if ( a[0][i] ) {                          /* 若该列尚未处理，则进行处理 */
           for (e=0, j=1; j<7; j++)       /* j:行坐标 e:该列中非0元素计数器 */
            if (a[j][i]) {   x=j; y=i; e++;}
           if (e==1) {            /* 若该列只有一个元素为非零，则进行消去操作 */
            for (t=1; t<7; t++)
              if (t!=i) a[x][t]=0;       /* 将非零元素所在的行的其它元素置0 */
            a[0][y]=0;                          /* 设置该列已处理完毕的标记 */
           }
         }
    }
    for (i=1; i<7; i++) {                                 /* 输出推理结果 */
       printf("%c is coming from ",'A'-1+i);
       for (j=1; j<7; j++)
         if (a[i][j]!=0)
           {   printf("%s.\n",m[ a[i][j] ]);     break; }
    }
}
```

* 运行结果

A is coming from ITALY.　　（意大利人）

B is coming from EUSSIAN.　　(俄罗斯人)
C is coming from U.K..　　(英 国 人)
D is coming from GER.　　(德 国 人)
E is coming from FRANCE.　　(法 国 人)
F is coming from U.S..　　(美 国 人)

* 问题的进一步讨论

生成条件矩阵然后使用消去法进行推理判断是一种常用的方法,对于解决较为复杂的逻辑问题是十分有效的。

* 思考题

地理课上老师给出一张没有说明省份的中国地图,从中选出五个省从1到5编号,要大家写出省份的名称。交卷后五位同学每人只答了二个省份的名称如下,且每人只答对了一个省,问正确答案是什么?

A 答:2号陕西,5号甘肃　　B 答:2号湖北,4号山东
C 答:1号山东,5号吉林　　D 答:3号湖北,4号吉林
E 答:2号甘肃,3号陕西

57. 谁家孩子跑最慢

张、王、李三家各有三个小孩。一天,三家的九个孩子在一起比赛短跑,规定不分年龄大小,跑第一得9分,跑第二得8分,依次类推。比赛结果各家的总分相同,且这些孩子没有同时到达终点的,也没有一家的两个或三个孩子获得相连的名次。已知获第一名的是李家的孩子,获得第二的是王家的孩子。问获得最后一名的是谁家的孩子?

* 问题分析与算法设计

按题目的条件,共有1+2+3+…+9=45分,每家的孩子的得分应为15分。据题意可知:获第一名的是李家的孩子,获得第二的是王家的孩子,则可推出:获第三名的一定是张家的孩子。由"这些孩子没有同时到达终点的"可知:名次不能并列,由"没有一家的两个或三个孩子获得相连的名次"可知:第四名不能是张家的孩子。

程序中为了方便起见,直接使用分数表示。

* 程序与程序注释

```
int score[4][4];
main()
{  int i, j, k, who;
   score[1][1]=7;          /* 按已知条件进行初始化:score[1]:张家三个孩子的得分 */
   score[2][1]=8;                          /* score[2]:王家三个孩子的得分 */
   score[3][1]=9;                          /* score[3]:李家三个孩子的得分 */
   for (i=4; i<6; i++)                   /* i:张家孩子在4到6分段可能的分数 */
    for (j=4; j<7; j++)                  /* j:王家孩子在4到6分段可能的分数 */
      for (k=4; i!=j && k<7; k++)        /* k:李家孩子在4到6分段可能的分数 */
       if ( k!=i && k!=j                                  /* 分数不能并列 */
          && 15-i-score[1][1]!=15-j-score[2][1]
```

```
            && 15-i-score[1][1]!=15-k-score[3][1]
            && 15-j-score[2][1]!=15-k-score[3][1] ) {
              score[1][2]=i; score[1][3]=15-i-7;          /* 将满足条件的 */
              score[2][2]=j; score[2][3]=15-j-8;          /* 结果记入数组 */
              score[3][2]=k; score[3][3]=15-k-9;
          }
    for (who=0, i=1; i<=4; i++, printf("\n"))
      for (j=1; j<=3; j++) {
        printf("%d ", score[i][j]);                       /* 输出各家孩子的得分情况 */
        if (score[i][j]==1) who=i;                        /* 记录最后一名的家庭序号 */
      }
    if (who==1)                                           /* 输出最后判断的结果 */
        printf("The last one arrived to end is a child from family Zhang.\n");
    else if (who==2)
        printf("The last one arrived to end is a child from family Wang.\n");
    else printf("The last one arrived to end is a child from family Li.\n");
}
```

* 运行结果

```
7  5  3
8  6  1
9  4  2
The last one arrived to end is a child from family Wang.
```

(获得最后一名的是王家的孩子。)

第八章　数字0到9的奇妙变幻

58. 拉丁方

构造N×N阶的拉丁方阵(2≤N≤9),使方阵中的每一行和每一列中数字1到N只出现一次。如N=4时:

1 2 3 4
2 3 4 1
3 4 1 2
4 1 2 3

* 问题分析与算法设计

构造拉丁方阵的方法很多,这里仅给出最简单的一种方法。观察给出的例子,可以发现:若将每一行中第一列的数字和最后一列的数字连起来构成一个环,则该环正好是由1到N顺序构成;对于第i行,这个环的开始数字为i。按照此规律可以很容易的写出程序。下面给出构造6阶拉丁方阵的程序。

* 程序与程序注释

```
#define N 6                                              /* 确定N值 */
main()
{   int j,i,k,t;
   printf("The possible Latin Squares of order %d are:\n", N);
   for (j=0; j<N; j++) {                                 /* 构造N个不同的拉丁方阵 */
    for (i=0; i<N; i++) {                                /* 构造一个N行的拉丁方阵 */
       t = (i+j) % N;                                    /* 确定该拉丁方阵第i行的第一个元素的值 */
      for (k=0; k<N; k++)                                /* 按照环的形式输出该行中的各个元素 */
         printf("%d ", (k+t)%N+1);
       printf("\n");
    }
    printf("\n");
   }
}
```

* 运行结果

```
The possible Latin Squares of order 6 are:
1 2 3 4 5 6          2 3 4 5 6 1          3 4 5 6 1 2
2 3 4 5 6 1          3 4 5 6 1 2          4 5 6 1 2 3
3 4 5 6 1 2          4 5 6 1 2 3          5 6 1 2 3 4
4 5 6 1 2 3          5 6 1 2 3 4          6 1 2 3 4 5
```

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

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

59. 填表格

将1、2、3、4、5和6填入下表中，要使得每一列右边的数字比左边的数字大，每一行下面的数字比上面的数字大。按此要求，可有几种填写方法？

* 问题分析与算法设计

按题目的要求进行分析，数字1一定是放在第一行第一列的格中，数字6一定是放在第二行第三列的格中。在实现时可用一个一维数组表示，前三个元素表示第一行，后三个元素表示第二行。先根据原题初始化数组，再根据题目中填写数字的要求进行试探。

* 程序与程序注释

```
int count;                                                    /* 计数器 */
main()
{   static int a[]={  1, 2, 3, 4, 5, 6} ;                    /* 初始化数组 */
   printf("The possible table satisfied above conditions are:\n");
   for (a[1] = a[0]+1 ; a[1] <= 5 ; ++a[1])                  /* a[1]必须大于a[0] */
    for (a[2] = a[1]+1 ; a[2] <= 5 ; ++a[2])                 /* a[2]必须大于a[1] */
      for (a[3]=a[0]+1 ; a[3] <= 5 ; ++a[3]) /* 第二行的a[3]必须大于a[0] */
        for (a[4] = a[1]>a[3] ? a[1]+1: a[3]+1 ; a[4] <= 5 ; ++a[4])
                                  /* 第二行的a[4]必须大于左侧a[3]和上边a[1] */
            if ( jud1(a) )    print(a);                      /* 如果满足题意，打印结果 */
}
jud1(s)                                                  /* 判断数组中的数字是否有重复的 */
   int s[];
{   int i,l;
   for (l = 1 ; l < 4 ;l++)
    for (i = l+1 ; i < 5 ; ++i)
      if (s[l] == s[i])     return(0);                   /* 若数组中的数字有重复的，返回0 */
   return(1);                                           /* 若数组中的数字没有重复的，返回1 */
}
```

```
print(u)
   int u[];
{   int k;
   printf("\nNo. :%d", ++count);
   for ( k = 0 ; k < 6 ; k++)
    if (k % 3 == 0)                          /* 输出数组的前三个元素作为第一行 */
       printf("\n %d ",u[k]);
    else                                     /* 输出数组的后三个元素作为第二行 */
       printf("%d ",u[k]);
}
```

* 运行结果

```
The possible table satisfied above conditions are:
No. 1:      No. 2:      No. 3:      No. 4:      No. 5:
  1 2 3       1 2 4       1 2 5       1 3 4       1 3 5
  4 5 6       3 5 6       3 4 6       2 5 6       2 4 6
```

60. 1～9分成1:2:3的三个3位数

将1到9这九个数字分成三个3位数，要求第一个3位数，正好是第二个3位数的二倍，是第三个3位数的三倍。问应当怎样分法。

* 问题分析与算法设计

问题中的三个数之间是有数学关系的，实际上只要确定第一个三位数就可解决问题。

试探第一个三位数后，计算出另外两个数，将其分别分解成三位数字，进行判断后确定所试探的数是否就是答案。

需要提醒的是：试探的初值可以是123，最大值是333。因为不可能超出该范围。

* 程序与程序设计

```
int a[9];
main()
{   int m, count=0;
   for (m=123; m<=333; m++)                  /* 试探可能的三位数 */
    if ( ok(m, a) && ok(2*m, a+3) && ok(3*m, a+6) )     /* 若满足题意 */
       printf ("No. %d: %d %d %d\n", ++count, m, 2*m, 3*m);
                                                        /* 输出结果 */
}
int ok(t, z)          /* 分解 t 的值，将其存入 z 指向的三个数组元素，若满足要求返回1 */
   int t, *z;
{   int *p1, *p2;
   for (p1=z; p1<z+3; p1++) {
    *p1=t%10;                                           /* 分解整数 */
    t/=10;
```

```
    for (p2=a; p2<p1; p2++)                    /* 查询分解出的数字是否已经出现过 */
        if ( *p1==0 || *p2== *p1) return (0);          /* 若重复返回0 */
    }
    return(1);                                          /* 否则返回1 */
}
```

* 运行结果

```
No. 1: 192  384  576
No. 2: 219  438  657
No. 3: 273  546  819
No. 4: 327  654  981
```

* 思考题

求出所有可能的以下形式的算式，每个算式中有九个数位，正好用尽1到9这九个数字。

① ○○○+○○○=○○○　　(共有168种可能的组合)
② ○×○○○○=○○○○　　(共有 2 种可能的组合)
③ ○○×○○○=○○○○　　(共有 7 种可能的组合)
④ ○×○○○=○○×○○○　　(共有 13 种可能的组合)
⑤ ○×○○○=○×○○○○　　(共有 28 种可能的组合)
⑥ ○○×○○=○×○○○○　　(共有 7 种可能的组合)
⑦ ○○×○○=○○×○○○　　(共有 11 种可能的组合)

61. 1～9组成三个3位的平方数

将1、2、3、4、5、6、7、8、9九个数字分成三组，每个数字只能用一次，即每组三个数不许有重复数字，也不许同其它组的三个数字重复，要求每组中的三位数都组成一个平方数。

* 问题分析与算法设计

本问题的思路很多，我们介绍一种简单快速的算法。

首先求出三位数中不包含0且是某个整数平方的三位数，这样的三位数是不多的。然后将满足条件的三位数进行组合，使得所选出的3个三位数的9个数字没有重复。

程序中可以将寻找满足条件三位数的过程和对该三位数进行数字分解的过程结合起来。

* 程序与程序注释

```
main()
{   int a[20], num[20][3], b[10];        /* a: 存放满足条件的三位数              */
                                         /* num:满足条件的三位数分解后得到的数字 */
                                         /* b: 临时工作区                        */
    int i, j, k, m, n, t, flag;
    printf ("The 3 squares with 3 different digits each are:\n");
    for (j=0,i=11; i<=31; i++)                    /* 求出是平方数的三位数 */
        if ( i%10!=0 ) {                          /* 若不是10的倍数,则分解三位数 */
        k=i*i;                                    /* 分解该三位数中的每一个数字 */
        num[j+1][0]=k/100;                                    /* 百位 */
```

```
        num[j+1][1]=k/10%10;                                    /* 十位 */
        num[j+1][2]=k%10;                                       /* 个位 */
        if ( !(num[j+1][0]==num[j+1][1] || num[j+1][0]==num[j+1][2]||
            num[j+1][1]==num[j+1][2]) )          /* 若分解的三位数字均不相等 */
          a[++j]=k;                    /* j:计数器,统计已找到的满足要求的三位数 */
      }
    for (i=1; i<=j-2; ++i) {          /* 从满足条件的三位数中选出三个进行组合 */
      b[1]=num[i][0];                                  /* 取第i个数的三位数字 */
      b[2]=num[i][1];
      b[3]=num[i][2];
      for (t=i+1; t<=j-1; ++t) {
        b[4]=num[t][0];                                /* 取第t个数的三位数字 */
        b[5]=num[t][1];
        b[6]=num[t][2];
        for (flag=0, m=1; !flag&&m<=3; m++)    /* flag:出现数字重复的标记 */
          for (n=4; !flag&&n<=6; n++)       /* 判断前两个数的数字是否有重复 */
            if (b[m]==b[n]) flag=1;                    /* flag=1:数字有重复 */
        if (!flag)
          for (k=t+1; k<=j; ++k) {
              b[7]=num[k][0];                          /* 取第k个数的三位数字 */
              b[8]=num[k][1];
              b[9]=num[k][2];
              for (flag=0, m=1; !flag&&m<=6; m++)
                                                  /* 判断前两个数的数字是否 */
                for (n=7; !flag&&n<=9; n++)       /* 与第三个数的数字重复   */
                  if (b[m]==b[n]) flag=1;
              if (!flag)                                /* 若均不重复则打印结果 */
                printf("%d, %d, %d\n", a[i], a[t], a[k]);
          }
      }
    }
}
```

* 运行结果

```
The 3 squares with 3 different digits each are:
  361, 529, 784
```

* 思考题

将1、2、3、4、5、6、7、8、9九个数字分成二组，每个数字只能用一次，一组形成一个5位数，另一组形成一个4位数，使得前者为后者的 n 倍。求所有满足条件的5位数和4位数。(注意：N_{max}=68，68以内的某些 N 也是不可能的。不可能的 N 值包括：1、10、11、20、21、25、30、31等共32个。)

62. 由8个整数形成奇特的立方体

任意给出8个整数，将这8个数分别放在一个立方体的八个顶点上，要求每个面上的四个数之和皆相等。

* 问题分析与算法设计

简化问题：将8个顶点对应数组中的8个元素，将“每个面上的四个数之和皆相等”转换为数组元素之间和的相等关系。这里的关键在于正确地将立方体的8个顶点与数组的8个元素对应。

可利用简单的穷举方法建立8个数的全部排列。

* 程序与程序注释

```
main()
{   int a[9],ii=0,i,a1,a2,a3,a4,b1,b2,b3,b4, flag;
   for (i=1; i<=8; i++) {                                    /* 输入8个整数 */
     printf ("Please enter [%d] number:", i);
     scanf("%d",&a[i]);
     ii+=a[i];
}
printf(" * * * * * * * * * * * * * * * * * * * *\n");
if ( ii%2 ) {                                    /* 和为奇数则输入的8个数不可用 */
   printf("Sorry they can't be constructed required cube!\n");
   exit(0);
}
for (flag=0,a1=1; a1<=8; a1++)              /* flag:完成标记 flag=1:表示完成 */
  for (a2=1; a2<=8; a2++)                  /* 采用八重循环建立八个整数的全排列 */
    if (a2!=a1)                                          /* 前两个数不能相同 */
      for (a3=1; a3<=8; a3++)
        if (a3!=a2 && a3!=a1)                            /* 前三个数不能相同 */
          for (a4=1; a4<=8; a4++)
            if (a4!=a3 && a4!=a2 && a4!=a1)              /* 前四个数不能相同 */
              for (b1=1; b1<=8; b1++)
                if (b1!=a4 && b1!=a3 && b1!=a2 && b1!=a1)        /* 不能相同 */
                for (b2=1; b2<=8; b2++)
                  if (b2!=b1 && b2!=a4 && b2!=a3 && b2!=a2 && b2!=a1)
                    for (b3=1; b3<=8; b3++)
                    if (b3!=b2   && b3!=b1 && b3!=a4 && b3!=a3
                                 && b3!=a2 && b3!=a1)        /* 不能取相同的数 */
                      for (b4=1; b4<=8; b4++)
            if (b4!=b2 && b4!=b1 && b4!=b3 && b4!=a4 &&
                          b4!=a3 && b4!=a2 && b4!=a1)
              if (a[b1]+a[b2]+a[b3]+a[b4]==ii/2
```

```
                && a[a1]+a[a2]+a[b1]+a[b2]==ii/2
                && a[a1]+a[a4]+a[b1]+a[b4]==ii/2)
            {   flag=1; goto out; }
                                        /* 满足条件则将 flag 置1后退出 */
out:
  if (flag) {
    printf("They can be constructed required cube as follow:\n");
    printf("        /%2d ·····/%2d\n", a[a4], a[a3]);
    printf("      %2d/·····%2d/ | \n", a[a1], a[a2]);
    printf("        |  |        |  |\n");
    printf("        |  |        |  |\n");
    printf("        |  %2d|        |  |%2d\n", a[b4], a[b3]);
    printf("        /·····-/\n");
    printf("      %2d/·····%2d/\n", a[b1], a[b2]);
  }
    else printf("Sorry they can't be constructed required cube!\n");
}
```

* 运行结果

1. Please enter [1] number:20
   Please enter [2] number:45
   Please enter [3] number:39
   Please enter [4] number:25
   Please enter [5] number:29
   Please enter [6] number: 7
   Please enter [7] number: 3
   Please enter [8] number: 2
   Sorry they can't be constructed required cube!

2. Please enter [1] number:20
   Please enter [2] number:21
   Please enter [3] number:22
   Please enter [4] number:23
   Please enter [5] number:24
   Please enter [6] number:25
   Please enter [7] number:26
   Please enter [8] number:27
   They can be constructed required cube as follow:

```
    27/----------/24
   20/----------/23|
    |   |      |   |
    |   |      |   |
    | 22|      |   |21
    |  /-------|--/
   25/----------/28
```

* 思考题

程序中建立全排列的方法效率太低，算法虽然简单但程序过于冗余。请读者设计新的算法完成同样的工作。

63. 减式还原

编写程序求解下式中各字母所代表的数字，不同的字母代表不同的数字。

```
  PEAR
-  ARA
------
  PEA
```

* 问题分析与算法设计

类似的问题从计算机算法的角度来说是比较简单的，可以采用最常见的穷举法解决。程序中采用循环穷举每个字母所可能代表的数字，然后将字母代表的数字转换为相应的整数，代入算式后验证算式是否成立即可解决问题。

*程序与程序注释

```
main()
{   int p, e, a, r;
  for (p=1; p<=9; p++)                    /* 从1到9穷举字母 p 的全部可能取值 */
    for (e=0; e<=9; e++)                  /* 从0到9穷举字母 e 的全部可能取值 */
      if (p!=e)                           /* p!=e */
        for (a=1; a<=9; a++)              /* 从0到9穷举字母 a 的全部可能取值 */
          if (a!=p && a!=e)               /* a!=p and a!=e */
            for (r=0; r<=9; r++)          /* 从0到9穷举字母 r 的全部可能取值 */
              if (r!=p && r!=e && r!=a    /* 四个字母互不相同且满足算式 */
                  && p*1000+e*100+a*10+r-(a*100+r*10+a)
                                          == p*100+e*10+a ) {
                printf("  PEAR    %d%d%d%d\n", p, e, a, r);
                printf(" -ARA  -  %d%d%d\n", a, r, a);
                printf("------  -----\n");
                printf("   PEA      %d%d%d\n", p, e, a);
              }
}
```

* 运行结果

```
  PEAR          1098
-  ARA        -  989
------        -------
   PEA          109
```

* 思考题

请复原下面的和式。不同的字母代表不同的数字。

```
   SEVEN                          82524                  82526
   THREE                          19722                  19722
+    TWO        答案:          +    106          +         104
-------                        -------                -------
  TWELVE                       102352                 102352
```

64. 乘式还原(1)

A 代表数字0到9中的前五个数字,Z 代表后五个数字,请还原下列乘式。

```
        A Z A
×       A A Z
-------------
    A A A A
  A A Z Z
  Z A A
-------------
  Z A Z A A
```

* 问题分析与算法设计

问题本身并不复杂,可以对乘式中的每一位使用穷举法,最终可以得到结果。本题的关键在于怎样有效的判断每个部分积的每一位是否满足题意,这一问题处理不好,编写的程序会很长。程序实现中采用了一个判断函数,通过传入函数的标志字符串对所有的数进行统一的判断处理。

* 程序与程序注释

```
#include <stdio.h>
main()
{   long i, j, k, l, m, n, term, t1, t2, t3;
  int flag;
  for ( i = 0 ; i <= 4 ; ++i )                          /* 被乘数的第一位 */
   for ( j = 5 ; j <= 9 ; ++j )                         /* 被乘数的第二位 */
     for ( k = 0 ; k<= 4 ; ++k ) {                      /* 被乘数的第三位 */
       term = 100 * i + 10 * j + k;                          /* 被乘数 */
       for ( flag=0, n=0 ; n<4 && !flag ; )              /* 乘数的第一位 */
        flag = jud( (t3=++n * 100 * term)/100, "001");
                                                        /* 判断第三个部分积 */
      if ( flag ) {
        for ( flag=0, m=0; m<4 && !flag; )              /* 乘数的第二位 */
          flag = jud( (t2=++m * 10 * term)/10, "1100" );
                                                        /* 判断第二个部分积 */
       if ( flag ) {
         for ( flag=0, l=5; l<9 && !flag; )             /* 乘数的第三位 */
           flag = jud( t1=++l * term, "0000" );
                                                        /* 判断第一个部分积 */
       if (flag && jud(t1+t2+t3, "00101"))              /* 判断乘式的积 */
            print( term, n * 100+m * 10+l, t1, t2, t3 );
```

```
            }
        }
    }
}
print( a, b, s1, s2, s3 )                                            /* 打印结果 */
   long a, b, s1, s2, s3;
{  printf( "\n      %ld\n", a );
   printf( " *) %ld\n", b );
   printf( "………………………………\n");
   printf( "   %ld\n   %ld\n   %ld\n",s1,s2/10,s3/100 );
   printf( "………………………………\n" );
   printf( " %ld\n", a * b );
}
jud (q, pflag)                        /* 判断一个数的每一位是否满足要求的判断函数 */
  long q;                                                    /* 需要判断的数 */
   char *pflag;     /* 标志字符串:A 用1表示,Z 用0表示,标志串排列顺序:个十百… */
{ while ( q!=0 && *pflag != NULL )       /* 循环判断对应位的取值范围是否正确 */
      if ( *pflag-'0' != (q%10>=5 ? 1 : 0) )    /* 标志位与对应的位不符,返回0 */
         return (0);
      else {   q /= 10; ++pflag; }                /* 若相符则取下一位进行判断 */
   if ( q==0 && *pflag==NULL ) /* q 的位数与标志字符串的长度相同时,返回1 */
         return (1);
   else return (0);
}
```

* 运行结果

```
          3 7 2
  ×)      2 4 6
  ---------------
        2 2 3 2
      1 4 8 8
      7 4 4
  ---------------
      9 1 5 1 2
```

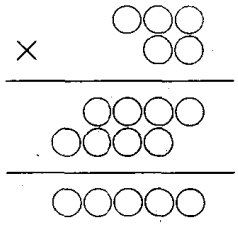
* 思考题

E 代表数字0到9中的偶数数字,O 代表奇数数字,请还原下列乘式。

```
       E E O                   2 8 5
  ×      O O              ×      3 9
  -----------     答案:   -----------
     E O E O                 2 5 6 5
     E O O                   8 5 5
  -----------             -----------
   O O O O O               1 1 1 1 5
```

65. 乘式还原(2)

有乘法算式如下：



18个○的位置上全部是素数(2、3、5或7)，请还原此算式。

＊ 问题分析与算法设计

问题中虽然有18个数位，但只要确定乘数和被乘数后经过计算就可确定其它的数位。

乘数和被乘数共有5个数位，要求每个数都是质数。完全可以采用穷举的方法对乘数和被乘数进行穷举，经过判断后找出答案。但是这种方法给人的感觉是“太笨了”，因为组成的数字只是质数(4个)，完全没有必要在那么大的范围内进行穷举。只需要试探每一位数字为质数时的情况即可。

采用五重循环的方法实现对于5个数字的穷举，在前面的许多例题中都已见过。循环实现简单易行，但嵌套的层次太多，需要穷举的变量的数量直接影响到循环嵌套的层数，这种简单的实现方法缺少技巧性。本例的程序中给出了另一种实现同样功能的算法，该算法的实现思想请阅读程序，这里不再赘述。

程序中并没有直接对质数进行穷举，而是将每个质数与1到4顺序一一对应，在穷举时为处理简单仅对1到4进行穷举处理，待要判断产生的乘积是否满足条件时再利用一个数组完成向对应质数的转换。请体会程序中的处理方法。程序中使用的算法实际上是回溯法。

＊ 程序与程序注释

```
#define NUM 5                                          /* 需要穷举的变量数目 */
#define C_NUM 4                                        /* 每个变量的值的变化范围 */
int a[NUM+1];                                          /* 为需要穷举的变量开辟的数组 */
    /* a[1]:被乘数的百位,a[2]:十位,a[3]:个位.a[4]:被乘数的十位,a[5]:个位 */
int b[] = { 0, 2, 3, 5, 7} ;                           /* 存放质数数字的数组.不使用第0号元素 */
main()
{   int i, not_finish=1;
    i=2;                                               /* i:将要进行处理的元素的指针下标.设置初始值 */
    a[1]=1;                                            /* 为第1号元素设置初始值 */
    while (not_finish) {                               /* not_finish:程序运行没结束标记 */
      while ( not_finish && i<=NUM )
                                                       /* 处理包括第i个元素在内的后续元素,
                                                          找出当前条件下的一种各个变量的一种可能的取值方法 */
        if (a[i] >= C_NUM)                             /* 当要处理的元素取值超过规定的C_NUM时 */
          if (i==1 && a[1]==C_NUM)
            not_finish=0;                              /* 若1号元素已经到C_NUM,则处理全部结束 */
```

```
            else a[i--]=0;          /* 将要处理的元素置0,下标-1(回退一个元素) */
          else a[i++]++;                    /* 当前元素值加1后下标指针加1 */
        if (not_finish) {
          long int sum1, sum2,sum3,sum4;                    /* 定义临时变量 */
          sum1=b[a[1]]*100+b[a[2]]*10+b[a[3]];              /* 计算被乘数 */
                  /* 利用数组的下标与质数的对应关系完成序号1到4向质数的转换 */
          sum2=sum1*b[a[5]];              /* 计算乘数个位与被乘数的部分积 */
          sum3=sum1*b[a[4]];              /* 计算乘数十位与被乘数的部分积 */
          if (sum2>=2222 && sum2<=7777 && f(sum2) &&
            sum3>=2222 && sum3<=7777 && f(sum3))
                                          /* 判断两部分积是否满足题目条件 */
            if ((sum4=sum2+sum3*10)>=22222 && sum4<=77777 && f(sum4)) {
                                          /* 判断乘式的积是否满足题目条件 */
              printf(" %d\n",sum1);              /* 若满足题意,则打印结果 */
              printf(" * %d%d\n", b[a[4]],b[a[5]]);
              printf("-----------------------------------------\n");
              printf(" %d\n",sum2);
              printf(" %d\n",sum3);
              printf("-----------------------------------------\n");
              printf(" %d\n",sum4);
            }
          i=NUM;                          /* 为穷举下一个可能取值作准备 */
        }
      }
}
f(sum)                  /* 判断 sum 的每一位数字是否是质数,若不是返回0,若是返回1 */
   long sum;
{   int i, k, flag;                          /* flag=1:数字是质数的标记 */
   while (sum>0) {
      i=sum%10;                                        /* 取个位的数字 */
      for (flag=0, k=1; ! flag && k<=C_NUM; k++)
         if (b[k]==i) {   flag=1; break; }
      if (!flag) return (0);
      else sum=sum/10;
    }
    return (1);
}
```

* 运行结果

```
      7 7 5
  ×     3 3
-------------
    2 3 2 5
  2 3 2 5
-------------
  2 5 5 7 5
```

* *思考题*

以下乘式中，A、B、C 代表一确定的数字，○代表任意数字，请复原。

```
           A B C                      2 8 6
  ×        B A C        答案：  ×     8 2 6
  ------------------           ---------------
        ○○○○                      1 7 1 6
        ○○A                      5 7 2
      ○○○B                    2 2 8 8
  ------------------           ---------------
      ○○○○○○                  2 3 6 2 3 6
```

66. 除式还原(1)

给定下列除式，其中包含5个7，其它打×的位置上是任意数字，请加以还原。

```
                           ×7×     ———————— 商
除数 ———————— ××| ×××××    ———————— 被除数
                     ×77
                     ------
                      ×7×
                      ×7×
                     ------
                        ××
                        ××
                     ------
                         0
```

* *题目分析与算法设计*

首先分析题目，由除式本身尽可能多地推出已知条件。由除式本身已知：

1. 被除数的范围是10000到99999，除数的范围是10到99，且可以整除；
2. 商为100到999之间，且十位数字为7；
3. 商的第一位与除数的积为三位数，且后两位为77；
4. 被除数的第三位一定为4；
5. 7乘以除数的积为一个三位数，且第二位为7；
6. 商的最后一位不能为0，且与除数的积为一个二位数。

由已知条件就可以采用穷举法找出结果。

* *程序与程序注释*

```
main()
{  long int i;
int j, k, l;
for (i=10000; i<=99999; i++)                              /* 1. i:被除数 */
  if ( i%1000-i%100==400 )                    /* 4. 被除数的第三位一定为4 */
    for (j=10; j<=99; j++)                                 /* 1. j:除数 */
      if (i%j==0 && (l=i/j)%100>=70 && l%100<80 && l%10!=0
                                                   && l>100 && l<=999)
   /* 1. 可以整除 && 2. 商l在100到999之间且十位数字为7 && 6. 商的个位不能为0 */
```

```
    if (j*(l%10))<100 && j*(l%10)>10)  /* 6.商的个位与除数的积为二位数 */
      if( j*7%100>=70 && j*7%100 < 80)
                                        /* 5. 7乘以除数的积的第二位为7 */
        if ( j*(l/100)%100 == 77 && j*(l/100)>100 )
                                        /* 3. 商的第一位与除数的积的后两位为77 */
          printf("%ld/%d=%d\n", i, j, l);
}
```

* 运行结果

```
        9 7 1
    ------------
53 | 5 1 4 6 3
     4 7 7
   ---------
       3 7 6
       3 7 1
     -------
           5 3
           5 3
         -----
             0
```

* 问题的进一步讨论

在推出的已知条件中，几乎所有的条件都是十分明显的，换句话说，推出的已知条件就是对题目的一种平铺直叙。这种推已知条件的方法十分简单，并且行之有效。

* 思考题

下列除式中仅在商中给定了一个8，其它打×的位置上是任意数字，请还原。

```
                  × 8×   ————— 商
除数 ————— ××× | ××××××  ————— 被除数
               •××××
             ---------
                 ×××
                 ×××
               -------
                 ××××
                 ××××
               -------
                    0
```

67. 除式还原(2)

下列除式中仅在商中给定了一个7，其它打×的位置全部是任意数字，请还原。

```
                  × 7×××  ————— 商
除数 ————— ××× | ××××××××  ————— 被除数
               ××××       ————— ①
             ---------
                 ×××      ————— ②
                 ×××      ————— ③
               -------
                 ××××     ————— ④
                  ×××     ————— ⑤
               -------
                   ××××   ————— ⑥
                   ××××   ————— ⑦
                 -------
                     0
```

* 题目分析与算法设计

这道题是不可能使用单纯的穷举方法求解的，一则计算时间太长，二则难于求出除式中各部分的值。

对除式进行分析，尽可能多地推出限制条件：

由③可以看出，商的第二位7乘除数得一个三位数，所以除数<=142。

由除数乘商的第一位为一个四位数可知，商的第一位只能为8或9且除数>=112。同时商的第五位也为8或9；被除数的前四位一定<=142*9+99且>=1000+10。

由④⑤⑥可以看出，④的前两位一定为“10”；⑤的第一位一定为“9”；⑥的前两位一定在10到99之间；商的第四位一定为0。

由⑤的第一位一定是“9”和“112<=除数<=142”可知：商的第三位可能为7或8。

由除式本身可知：商的第四位为0。

由①可知：除数×商的第一位应当为一个四位数。

由⑤可知：除数×商的第三位应当为一个三位数。

编程时为了方便，将被除数分解：前四位用 a[0]表示，第五位用 a[1]，第六位用 a[2]，第七八两位用 a[3]；除数用变量 b 表示；分解商：第一位用 c[0]，第三位用 c[1]，第五位用 c[2]；其他的部分商分别表示为：②的前两位为 d[0]，④的前三位为 d[1]，⑥的前两位为 d[2]。将上述分析用数学的方法综合起来可以表示为：

被除数：　1010 <= a[0] <= 1377　0 <= a[1] <= 9

　　0 <= a[2] <= 9　　0 <= a[3] <=99

除数：　112 <= b <= 142

商：　8 <= c[0] <= 9　　7 <= c[1] <= 8　8 <= c[2] <= 9

②的前两位：　10 <= d[0] <= 99

④的前三位：　100 <= d[1] <b

⑥的前两位：　10 <= d[2] <= 99

①式部分积：　b * c[0] > 1000

⑤式部分积：　100 < b * c[1] < 1000

* 程序与程序注释

```
main()
{  int a[4],b,c[3],x,d[4],i=1;
for (a[0]=1010; a[0]<=1377; a[0]++)
   for (b=112; b<=142; b++)
   for (c[0]=8; c[0]<=9; c[0]++)
    if ( b*c[0]>1000 && (d[0]=a[0]-b*c[0])>=10 && d[0]<100 )
     for (a[1]=0; a[1]<=9; a[1]++)
      if ((d[1]=d[0]*10+a[1]-b*7)>=100 && d[1]<b)
        for (a[2]=0; a[2]<=9; a[2]++)
        for (c[1]=7; c[1]<=8; c[1]++)
         if (b*c[1]<1000 && (d[2]=d[1]*10+a[2]-b*c[1])>=10 && d[2]<100)
           for (a[3]=0; a[3]<=99; a[3]++)
```

```
        for (c[2]=8; c[2]<=9; c[2]++)
          if (d[2]*100+a[3]-b*c[2]==0) {
            printf("No%2d:",i++);
            printf("%d%d%d%d%d/",a[0],a[1],a[2],a[3]/10,a[3]%10);
            printf("%d=",b);
            printf("%d%d%d%d%d\n",c[0],7,c[1],0,c[2]);
          }
}
```

* 运行结果

No 1:12128316/124=97809

* 思考题

下列除式中"×"所在的位置全部是任意数字，请还原。

```
          ×××××
      ---------
×××  | ××××××××
       ××××
      ---------
        ××××
         ×××
      ---------
          ×××
          ×××
      ---------
          ××××
          ××××
      ---------
             0
```

68. 九位累进可除数

求九位累进可除数。 所谓九位累进可除数是这样一个数：这个数用1到9这九个数字组成，每个数字刚好只出现一次。这个九位数的前两位能被2整除，前三位能被3整除……前 N 位能被 N 整除，整个九位数能被9整除。

* 问题分析与算法设计

问题本身可以简化为一个穷举问题：只要穷举每位数字的各种可能的取值，按照题目的要求对穷举的结果进行判断就一定可以得到正确结果。

问题中给出了"累进可除"这一条件，就使得我们可以在穷举法中加入条件判断。在穷举的过程中，当确定部分位的值后，马上就判断产生的该部分是否符合"累进可除"条件，若符合，则继续穷举下一位数字；否则刚产生的那一位数字就是错误的。这样将条件判断引入到穷举法之中，可以尽可能早的发现矛盾，尽早地放弃不必要穷举的值，从而提高程序的执行效率。

为了达到早期发现矛盾的目的，不能采用多重循环的方法实行穷举，那样编出的程序质量较差。程序中使用的算法已不再是穷举法，而是回溯法。

* 程序与程序注释

```
#define NUM 9
int a[NUM+1];
```

```
main()
{   int number, i, k, flag, not_finish=1;
    long sum;
    i=1;
     /* i:正在处理的数组元素,表示前i-1个元素已经满足要求,正处理的是第i个元素 */
    a[1]=1;                                          /* 为元素a[1]设置初值 */
    while (not_finish) {                          /* not_finish=1:处理没有结束 */
     while (not_finish && i<=NUM) {
      for (flag=1,k=1; flag && k<i; k++)
       if (a[k]==a[i]) flag=0;          /* 判断第i个元素是否与前i-1个元素重复 */
      for (sum=0, k=1; flag&&k<=i; k++) {
       sum=10*sum+a[k];
       if ( sum%k ) flag=0;              /* 判断前k位组成的整数是否能被k整除 */
      }
      if (! flag) {                    /* flag=0:表示第i位不满足要求,需要重新设置 */
       if (a[i]==a[i-1]) {               /* 若a[i]的值已经经过一圈追上a[i-1] */
         i--;                                  /* i值减1,退回处理前一个元素 */
         if (i>1 && a[i]==NUM )
           a[i]=1;                    /* 当第i位的值达到NUM时,第i位的值取1 */
         else if (i==1 && a[i]==NUM)       /* 当第1位的值达到NUM时结束 */
           not_finish=0;                                  /* 置程序结束标记 */
         else a[i]++;                                /* 第i位的值取下一个,加1 */
       }
       else if (a[i]==NUM) a[i]=1;
         else a[i]++;
      }
      else                                        /* 第i位已经满足要求,处理第i+1位 */
          if ( ++i<=NUM )              /* i+1处理下一元素,当i没有处理完毕时 */
            if (a[i-1]==NUM) a[i]=1;  /* 若i-1的值已为NUM,则a[i]的值为1 */
            else a[i]=a[i-1]+1;              /* 否则,a[i]的初值为a[i-1]值的 */
     }                                                        /* "下一个"值 */
     if (not_finish) {
       printf ("\nThe progressire divisiable number is: ");
       for (k=1; k<=NUM; k++)                                 /* 输出计算结果 */
         printf("%d", a[k]);
       if (a[NUM-1]<NUM) a[NUM-1]++;
       else a[NUM-1]=1;
       i=NUM-1;
     }
```

```
    }
}
```

* 运行结果

```
The progressire divisiable number is: 381654729
```

* 思考题

求N位累进可除数。用1到9这九个数字组成一个N(3<=N<=9)位数，每位数字的组成不限，使得该N位数的前两位能被2整除，前三位能被3整除，……，前N位能被N整除。求满足条件的N位数。

第九章　数的变幻

69. 魔术师的猜牌术(1)

魔术师利用一副牌中的13张黑桃，预先将它们排好后迭在一起，牌面朝下。对观众说：我不看牌，只数数就可以猜到每张牌是什么，我大声数数，你们听，不信？你们就看。魔术师将最上面的那张牌数为1，把它翻过来正好是黑桃A，将黑桃A放在桌子上，然后按顺序从上到下数手中的余牌，第二次数1、2，将第一张牌放在这迭牌的下面，将第二张牌翻过来，正好是黑桃2，也将它放在桌子上。第三次数1、2、3，将前面两张依次放在这迭牌的下面，再翻第三张牌正好是黑桃3。这样依次进行将13张牌全翻出来，准确无误。问魔术师手中的牌原始次序是怎样安排的？

* 问题分析与算法设计

题目已经将魔术师出牌的过程描述清楚，我们可以利用倒推的方法，很容易地推出原来牌的排列顺序。

人工倒推的方法是：在桌子上放13个空盒子排成一圈，从1开始顺序编号，将黑桃A放入1号盒子中，从下一个空盒子开始对空的盒子计数，当数到第二个空盒子时，将黑桃2放入空盒子中，然后再从下一个空盒子开始对空盒子计数，顺序放入3、4、5…，直到放入全部13张牌。注意在计数时要跳过非空的盒子，只对空盒子计数。最后牌在盒子中的顺序，就是魔术师手中原来牌的顺序。

这种人工的方法是行之有效的，计算机可以模拟求解。

* 程序与程序注释

```
int a[14];
main()
{   int i,n,j=1;                          /* j:数组(盒子)下标,初始时为1号元素 */
   printf("The original order of cards is: ");
   for (i=1; i<=13; i++) {                 /* i:要放入盒子中的牌的序号 */
     n=1;                                  /* n:空盒计数器 */
     do {
        if (j>13) j=1;    /* 由于盒子构成一个圈,j超过最后一个元素则指向1号元素 */
        if ( a[j]) j++;                    /* 跳过非空的盒子,不进行计数 */
        else {   if (n==i) a[j]=i;         /* 若数到第i个空盒子,则将牌放入空盒中 */
              j++; n++;                    /* 对空盒计数,数组下标指向下一个盒子 */
             }
     } while (n<=i);                       /* 控制空盒计数为i */
   }
   for (i=1; i<=13; i++)                   /* 输出牌的排列顺序 */
     printf("%d ", a[i]);
```

```
    printf("\n");
}
```

* 运行结果

The original order of cards is: 1 8 2 5 10 3 12 11 9 4 7 6 13

70. 魔术师的猜牌术(2)

魔术师再次表演,他将红桃和黑桃全部迭在一起,牌面朝下放在手中,对观众说:最上面一张是黑桃 A,翻开后放在桌上。以后,从上至下每数两张全依次放在最底下,第三张给观众看,便是黑桃2,放在桌上后再数两张依次放在最底下,第三张给观众看,是黑桃3。如此下去,观众看到放在桌子上的牌的顺序是:

黑桃 A 2 3 4 5 6 7 8 9 10 J Q K

红桃 A 2 3 4 5 6 7 8 9 10 J Q K

问魔术师此次手中牌的原始顺序是什么?

* 问题分析与算法设计

本题可在上题基础上进行编程,不同的在于计数的方法和牌的张数。这些并不影响我们求解题目的思路,仍可按照倒推的方法,得到原来魔术师手中的牌的顺序。

* 程序与程序注释

```
int a[27];
main()
{   int i, n, j=1;
    a[1]=1;                                          /* 初始化第一张牌 */
    printf("The original order of cards is (r:rad b:block) :\n");
    for (i=2; i<=26; i++) {
      n=1;
      do {   if (j>26) j = 1;                        /* j超过最后一个元素则指向1号元素 */
         if (a[j]) j++;                              /* 跳过非空的盒子,不进行计数 */
         else {   if (n==3) a[j]=i;                  /* 若数到第3个空盒子,则将牌放入空盒中 */
                  j++; n++;                          /* 对空盒计数,数组下标指向下一个盒子 */
              }
      }   while (n<=3);                              /* 控制空盒计数为3 */
    }
    for(i=1; i<=26; i++) {                           /* 输出牌的排列顺序 */
        printf("%c", a[i]>13 ? 'r':'b');
        printf("%2d ", a[i]>13 ? a[i]-13 : a[i]);
    }
    printf("\n");
}
```

* 运行结果

The original order of cards is (r:rad b:block) :

b1 r6 b10 b2 r12 r3 b3 b11 r9 b4 r7 b12 b5
r4 r13 b6 b13 r11 b7 r5 r1 b8 r8 r10 b9 r2

71. 约瑟夫问题

这是17世纪的法国数学家加斯帕在《数目的游戏问题》中讲的一个故事：15个教徒和15个非教徒在海上遇险，必须将一半的人投入海中，其余的人才能幸免于难，于是想了一个办法：30个人围成一个圆圈，从第一个人开始依次报数，每数到第9个人就将他扔入大海，如此循环进行直到仅余15个人为止。问怎样排法，才能使每次投入大海的都是非教徒。

* 问题分析与算法设计

约瑟夫问题本身并不难，但求解的方法很多；题目的变化形式也很多。这里给出一种实现方法。

题目中30个人围成一圈，因而启发我们用一个循环的链来表示。可以使用结构数组构成一个环形链。结构中有两个成员，其一为指向下一个人的指针，以构成环形的链；其二为该人是否被扔下海的标记，为1表示还在船上。从第一个人开始对尚未扔下海的人进行计数，每数到9时，将结构中的标记改为0，表示该人已被扔下海了。这样循环计数直到有15个人被扔下海时为止。

* 程序与程序注释

```
struct node
   {   int nextp;                          /* 指向下一个人的指针(下一个人的数组下标) */
       int no_out;              /* 是否被扔下海的标记。1:没有被扔下海 0:已被扔下海 */
   }   link[31];                                  /* 30个人,0号元素没有使用 */
main ()
{   int i, j, k;
   printf("The original circle is (+:pagandom, @:christian):\n");
   for (i=1; i<=30; i++) {                                    /* 初始化结构数组 */
    link[i].nextp = i+1;                        /* 指针指向下一个人(数组元素下标)*/
    link[i].no_out = 1;                           /* 标志置为1,表示人都在船上 */
   }
   link[30].nextp = 1;                        /* 第30个人的指针指向第1个人以构成环 */
   j = 30;            /* j:指向已经处理完毕的数组元素,从link[j]指向的人开始计数 */
   for (i=0; i<15; i++) {                              /* i:已扔下海的人数计数器 */
    for (k=0; ; )                              /* k:决定哪个人被扔下海的计数器 */
      if (k<15) {                                      /* 计数不到15则继续计数 */
        j = link[j].nextp;                             /* 修改指针,取下一个人 */
        k += link[j].no_out;                /* 进行计数。因已扔下海的人标记为0, */
      }                                          /* 故这样计数不会影响正常结果 */
      else break;                                    /* 计数到15则停止计数 */
    link[j].no_out = 0;                            /* 将标记置0,表示该人被扔下海 */
   }
   for (i=1; i<=30; i++)                                          /* 输出结果 */
```

```
    printf("%c ", link[i].no_out ? '@' : '+');          /* +:被扔下海 @:在船上 */
  printf("\n");
}
```

* 运行结果

The original circle is (+:pagandom, @:christian):

+++@@+@+@@@@+@+++@@+@@@+++@+@@+

(+:表示被扔下海的非教徒　　@:留在船上活命的教徒)

* 思考题

有N个小孩围成一圈并依次编号,教师指定从第M个小孩开始报数,报到第S个小孩即令其出列,然后从下一个孩子开始继续报数,数到第S个小孩又令其出列,如此直到所有的孩子都出列。求小孩出列的先后顺序。

72. 邮票组合

某人有四张3分的邮票和三张5分的邮票,用这些邮票中的一张或若干张可以得到多少种不同的邮资?

* 问题分析与算法设计

将问题进行数学分析,不同张数和面值的邮票组成的邮资可用下列公式计算:

$$S = 3 * i + 5 * j$$

其中i为3分邮票的张数,j为5分邮票的张数。

按题目要求,3分的邮票可以取0、1、2、3、4张,5分的邮票可以取0、1、2、3张。采用穷举法进行组合,可以求出这些不同面值不同张数的邮票组合后的邮资。

* 程序与程序注释

```
int a[27];
main()
{   int i,j,k,s,n=0;
  for (i=0; i<=4; i++)                              /* i:取三分邮票的张数 */
   for (j=0; j<=3; j++) {                           /* j:取五分邮票的张数 */
      s=i*3+j*5;                                    /* 计算组成的邮票面值 */
      for (k=0; a[k]; k++)                          /* 查找是否有相同的邮资 */
          if(s==a[k]) break;
      if (!a[k] && s)                   /* 没找到相同的邮资则满足要求存入数组 */
        {   a[k]=s; n++; }
   }
  printf("%d kinds:",n);                                /* 输出结果 */
  for (k=0; a[k]; k++)
   printf("2%d ",a[k]);
}
```

* 运行结果

19 kinds: 5 10 15 3 8 13 18 6 11 16 21 9 14 19 24 12 17 22 27

73. 和数能表示1～23的5个正整数

已知五个互不相同的正整数之和为23，且从这五个数中挑选若干个加起来可以表示从1到23之内的全部自然数，问这五个数都是什么？

* 问题分析与算法设计

从计算机程序设计的角度来说，可以用穷举法分解23，然后判断所分解的五个数是否可以表示1到23之间的全部整数。

* 程序与程序注释

```
main()
{  int a, b, c, d, e, i, j, k, l, m, x, count=0, f=0;
                                        /* f:分解的5个数可以表示出1～23的标记 */
printf("There are following possible result:\n");
for(a=1; a<=23; a++)                          /* 将23分解为a,b,c,d,e五个数 */
   for(b=1+a; b<=23-a; b++)
     for(c=1+b; c<=23-a-b; c++)
       for(d=1+c; d<=23-a-b-c; d++) {
         f=1;
         if ( (e=23-a-b-c-d) > d )
           for (f=0,x=1; x<24&&!f; x++)          /* 判断5个数可否表示1～23 */
             for (f=1,i=0; i<2&&f; i++)          /* 穷举5个数的全部取舍 */
               for (j=0; j<2&&f; j++)
                 for (k=0; k<2&&f; k++)
                   for (l=0; l<2&&f; l++)
                     for (m=0; m<2&&f; m++)
                       if (x==a*i+b*j+c*k+d*l+e*m) f=0;
         if ( !f ) printf(" [%d]: %d %d %d %d %d \n", ++count, a,b,c,d,e);
       }
}
```

* 运行结果

```
There are following possible result:
 [1]: 1 2 3 5 12
 [2]: 1 2 3 6 11
 [3]: 1 2 3 7 10
 [4]: 1 2 4 5 11
 [5]: 1 2 4 6 10
 [6]: 1 2 4 7 9
```

74. 可称1～40磅的4块砝码

法国数学家梅齐亚克在他著名的《数字组合游戏》(1962)中提出了一个问题：一位商人有

一个重40镑的砝码，一天不小心将砝码摔成了四块。后来商人称得每块的重量都是整镑数，而且发现用这四块碎片可以在天平上称1至40镑之间的任意重量。请问这四块碎片各重多少？

* 问题分析与算法设计

本题是上一题的发展。题目中给出的条件是“在天平上”，这意味着：同一砝码既可以放在天平的左侧，也可以放在天平的右侧。若规定重物只能放在天平的左侧，则当天平平衡时有：

重物重量＋左侧砝码重量总和＝右侧砝码重量总和

由此可得：

重物重量＝右侧砝码重量总和－左侧砝码重量总和

编程时只要根据以上公式，使“右侧砝码重量总和－左侧砝码重量总和”可以表示1到40之间的全部重量即可。编程中要注意的是：怎样采用一种简单的方法来表示一个砝码是在天平的“左侧”，还是在“右侧”，或是根本没有使用。

以下程序采用1、－1和0分别表示上述三种情况，请注意理解。

* 程序与程序注释

```
#include "math.h"
main()
{  int weight1, weight2, weight3, weight4, d1, d2, d3, d4, x, flag;
                                                    /* flag:满足题意的标记 */
   printf("The weight is broke up as following 4 pieces:");
   for (weight1=1; weight1<=40; weight1++)              /* 将40分解为4份 */
    for (weight2=weight1+1; weight2<=40-weight1; weight2++)
      for (weight3=weight2+1; weight3<=40-weight1-weight2; weight3++)
        if ( (weight4=40-weight1-weight2-weight3) >=weight3 ) {
          for (flag=1,x=1; x<41 && y; x++)
                                          /* 判断可否称出1～40之间的全部重量 */
            for (flag=0,d1=1; d1>-2; d1--)        /* 将重物放在天平的左边 */
              for (d2=1; d2>-2&&!flag; d2--)          /* 1:砝码在天平右边 */
                for (d3=1; d3>-2&&!flag; d3--)           /* 0:不用该砝码 */
                  for (d4=1; d4>-2&&!flag; d4--)  /* -1:砝码在天平左边 */
                    if (x==weight1*d1+weight2*d2+weight3*d3+weight4*d4)
                      flag=1;
          if(flag) printf("%d %d %d %d\n",weight1, weight2, weight3, weight4);
        }
}
```

* 运行结果

```
The weight is broke up as following 4 pieces: 1  3  9  27
```

75. 10个小孩分糖果

十个小孩围成一圈分糖果，老师分给第一个小孩10块，第二个小孩2块，第三个小孩8块，第四个小孩22块，第五个小孩16块，第六个小孩4块，第七个小孩10块，第八个小孩6块，第九个小

孩14块，第十个小孩20块。然后所有的小孩同时将自己手中的糖分一半给右边的小孩；糖块数为奇数的人可向老师要一块。问经过这样几次调整后大家手中的糖的块数都一样？每人各有多少块糖？

* 问题分析与算法设计

题目描述的分糖过程是一个机械的重复过程，编程算法完全可以按照描述的过程进行模拟。

* 程序与程序注释

```
int j=0;
main()
{   static int sweet[10]={  10,2,8,22,16,4,10,6,14,20}  ;     /* 初始化数组数据 */
   int i, t[10], l;
   printf("                  child\n");
   printf(" round 1 2 3 4 5 6 7 8 9 10\n");
   printf("-------------------------------------------\n");
   print(sweet);                                          /* 输出每个人手中糖的块数 */
   while ( judge(sweet) ) {                               /* 若不满足要求则继续循环 */
    for (i=0; i<10; i++)                                  /* 将每个人手中的糖分成一半 */
       if ( sweet[i]%2==0)                                /* 若为偶数则直接分出一半 */
         t[i] = sweet[i] = sweet[i]/2;
       else                                               /* 若为奇数则加1后再分出一半 */
         t[i] = sweet[i] = (sweet[i]+1)/2;
    for (l=0; l<9; l++)                                   /* 将分出的一半糖给右(后)边的孩子 */
      sweet[l+1] = sweet[l+1] + t[l];
    sweet[0] += t[9];
    print(sweet);                                         /* 输出当前每个孩子手中的糖块数 */
   }
}
judge(c)
   int c[];
{   int i;
   for (i=0; i<10; i++)                                   /* 判断每个孩子手中的糖是否相同 */
    if ( c[0] != c[i] ) return(1);                        /* 不相同返回1 */
   return(0);                                             /* 全相同返回0 */
}
print(s)                                                  /* 输出数组中每个元素的值 */
   int s[];
{   int k;
   printf(" %2d ",j++);
   for (k=0; k<10; k++) printf("%4d",s[k]);
```

```
    printf("\n");
}
```

* 运行结果

	child									
round	1	2	3	4	5	6	7	8	9	10
0	10	2	8	22	16	4	10	6	14	20
1	15	6	5	15	19	10	7	8	10	17
2	17	11	6	11	18	15	9	8	9	14
3	16	15	9	9	15	17	13	9	9	12
4	14	16	13	10	13	17	16	12	10	11
5	13	15	15	12	12	16	17	14	11	11
6	13	15	16	14	12	14	17	16	13	12
7	13	15	16	15	13	13	16	17	15	13
8	14	15	16	16	15	14	15	17	17	15
9	15	15	16	16	16	15	15	17	18	17
10	17	16	16	16	16	16	16	17	18	18
11	18	17	16	16	16	16	16	17	18	18
12	18	18	17	16	16	16	16	17	18	18
13	18	18	18	17	16	16	16	17	18	18
14	18	18	18	18	17	16	16	17	18	18
15	18	18	18	18	18	17	16	17	18	18
16	18	18	18	18	18	18	17	17	18	18
17	18	18	18	18	18	18	18	18	18	18

76. 小明买书

小明假期同爸爸一起去书店，他选中了六本书，每本书的单价分别为：3.1，1.7，2，5.3，0.9和7.2。不巧的是，爸爸兜里只带了十几块钱，为了让小明过一个愉快的假期，爸爸仍然同意买书，但提出一个要求，要小明从六本中选出若干本，使得单价相加所得的和同10最接近。你能够帮助小明解决这个问题吗？

* 问题分析与算法设计

分析题意，可将题目简化为：从六个数中选出若干个求和，使得和与10的差值最小。

题目中隐含两个问题。其一是怎样从六个数中选出若干个数；其二是求与10的差。

从六个数中选出若干个数实质是从六个数中选出若干个进行组合。每个数在组合过程中只有两种情况：要么是选中参加求和，要么是没选中不参加求和。这样就可以使用六重循环对每个数是否参加求和进行全部可能的情况组合。

关于求与10的差值应当注意的是：差值的含义是指差的绝对值。例如："9－10＝－1"和"11－10＝1"，但9和11这两者与10的差值都是"1"。若认为"9"与10的差值为"－1"就错了。

* 程序与程序注释

```
#include <math.h>
```

```
main()
{   int d[6],m,i,j;
    long b[63], flag;
    float c[6], min, x;
    printf ("Pleace enter the prices of 6 books: ");
    for ( i=0; i<6; i++) scanf("%f", &c[i]);                    /* 输入六个浮点数 */
    for ( i=0, min=-1, d[0]=0; d[0]<2; d[0]++)
                                                    /* 建立六个数的全部组合并处理 */
     for (d[1]=0; d[1]<2; d[1]++)                    /* i:差值具有 min 组合的数量 */
       for (d[2]=0; d[2]<2; d[2]++)                  /* min:与10的最小差值 */
         for (d[3]=0; d[3]<2; d[3]++)                /* d[]:组合时是否取该数的标志 */
          for (d[4]=0; d[4]<2; d[4]++)
           for (d[5]=0; d[5]<2; d[5]++) {
             for (flag=0, x=0., j=5; j>=0; j--)
                                /* flag:将六个数的组合用对应的一个十进制位表示 */
                                                    /* x:对应六个数组合的和 */
                 {   x += c[j]*d[j]; flag = flag*10+d[j]; }
            x = ((x-10>0) ? x-10 : 10-x );          /* x:组合的和与10的差 */
             if ( min<0 ) {
                 min=x;                         /* 对第一次计算出的差 min 进行处理 */
                 b[i++]=flag;                   /* b[]:记录有相同 min 的 flag 的数组 */
             }                                          /* i:b[]数组的下标 */
             else if ( min-x>1.e-6) {                   /* 对新的 min 的处理 */
              min=x; b[0]=flag; i=1;
             }
                 else if ( fabs( (double)x-min ) <1.e-6 )
                   b[i++]=flag;                         /* 对相等 min 的处理 */
         }
   for (m=0; m<i; m++) {              /* 输出全部 i 个与10的差值均为 min 的组合 */
       printf (" 10 (+ -) %.2f=", min);
       for (flag=b[m],j=0; flag>0; j++, flag/=10)
         if (flag%10)                   /* 将 b[]中存的标记 flag 还原为各个数的组合 */
          if ( flag>1) printf("%.2f+", c[j]);
          else         printf("%.2\n",c[j]);
   }
}
```

* 运行结果

Pleace enter the prices of 6 books: 3.1 1.7 2.0 5.3 0.9 7.2

10（＋ －）0.10＝2.00＋0.90＋7.20

10（＋ －）0.10＝1.70＋2.00＋5.30＋0.90

10（＋ －）0.10＝3.10＋1.70＋5.30

＊ 思考题

可以看出，程序中求六个数所能产生全部组合的算法并不好，使用六重循环进行处理使程序显得不够简洁。可以设计出更通用、优化的算法产生全部组合。

77. 波瓦松的分酒趣题

法国著名数学家波瓦松在青年时代研究过一个有趣的数学问题：某人有12品脱的啤酒一瓶，想从中倒出6品脱，但他没有6品脱的容器，仅有一个8品脱和一个5品脱的容器，怎样倒才能将啤酒分为两个6品脱呢？

＊ 问题分析与算法设计

将12品脱酒用8品脱和5品脱的空瓶平分，可以抽象为解不定方程：

$$8x-5y=6$$

其意义是：从12品脱的瓶中向8品脱的瓶中倒 x 次，并且将5品脱瓶中的酒向12品脱的瓶中倒 y 次，最后在12品脱的瓶中剩余6品脱的酒。

用 a、b、c 代表12品脱、8品脱和5品脱的瓶子，求出不定方程的整数解，按照不定方程的意义则倒法为：

$$a \rightarrow b \rightarrow c \rightarrow a$$
$$x \quad y$$

倒酒的规则如下：

①按 a→b→c→a 的顺序；

②b 倒空后才能从 a 中取；

③c 装满之后才能向 a 中倒。

按以上规则可以编写出程序如下。

＊ 程序与程序注释

```
int i;                                                    /* 最后需要分出的重量 */
main()
{   int a, y, z;          /* a:满瓶的容量  y:第一个空瓶的容量  z:第二个空瓶的容量 */
    printf ("Input Full a, Empty b,c, Get i:");
    scanf("%d%d%d%d", &a, &y, &z, &i);
    getti(a, y, z);                                       /* 按 a→y→z→a 操作步骤 */
    getti(a, z, y);                                       /* 按 a→z→y→a 操作步骤 */
}
getti (a, y, z)
    int a, y, z;          /* a:满瓶的容量  y:第一个空瓶的容量  z:第二个空瓶的容量 */
{   int b=0, c=0;                    /* b:第一瓶实际的重量  c:第二瓶实际的重量 */
    printf(" a%d b%d c%d\n%4d%4d%4d\n" , a, y, z, a, b, c);
    while ( a!=i || b!=i&&c!=i ) {                        /* 当满瓶!=i 或 另两瓶都!=i */
     if (!b)
```

```
        {   a-=y; b=y; }                    /* 如果第一瓶为空,则将满瓶倒入第一瓶中 */
    else if ( c==z )
        {   a+=z; c=0; }                    /* 如果第二瓶装满,则将第二瓶倒入满瓶中 */
    else if ( b>z-c )                       /* 如果第一瓶的重量>第二瓶的剩余空间 */
        {   b-=(z-c); c=z; }                /* 则将装满第二瓶,第一瓶中保留剩余部分 */
    else {   c+=b; b=0; }                   /* 否则,将第一瓶全部倒入第二瓶中 */
    printf ("%4d%4d%4d\n",a,b,c);
  }
}
```

* 运行结果

```
Input Full a, Empty b,c, Get i: 12 8 5 6
```

No. 1	a12	b8	c5
	12	0	0
	4	8	0
	4	3	5
	9	3	0
	9	0	3
	1	8	3
	1	6	5
	6	6	0

No. 2	a12	b5	c8
	12	0	0
	7	5	0
	7	0	5
	2	5	5
	2	2	8
	10	2	0
	10	0	2
	5	5	2
	5	0	7
	0	5	7
	0	4	8
	8	4	0
	8	0	4
	3	5	4
	3	1	8
	11	1	0
	11	0	1
	6	5	1

* 思考题

上面的程序中仅找出了两种分酒的方法,并没有找出全部的分法。请设计新的算法,找出全部的分酒方法,并找出一种倒酒次数最少的方法。

第十章　定理与猜想

78. 求 π 的近似值(1)

请利用“正多边形逼近”的方法求出 π 的近似值。

＊ 问题分析与算法设计

利用“正多边形逼近”的方法求 π 值在很早以前就存在，我们的先人祖冲之就是用这种方法在世界上第一个得到精确度达小数点后第 6 位的 π 值的。

利用圆内接正六边形边长等于半径的特点，将边数翻番，作出正十二边形，求出边长。重复这一过程，就可获得所需精度的 π 的近似值。

假设单位圆内接多边形的边长为 2b，边数为 i，则边数加倍后新的正多边形的边长为：

$$x = \frac{\sqrt{2 - 2 * \sqrt{1 - b * b}}}{2}$$

周长为：

$$y = 2 * i * x$$

i：为加倍前的正多边形的边数

＊ 程序与程序注释

```
#include "math.h"
main()
{ double e=0.1, b=0.5, c, d;
    long int i;                                          /* i:正多边形边数 */
    for (i=6; ;i*=2) {                                   /* 正多边形边数加倍 */
       d = 1.0 - sqrt(1.0-b*b);                          /* 计算圆内接正多边形的边长 */
       b = 0.5 * sqrt(b*b+d*d);
       if ( 2*i*b - i*e<1e-15) break;                    /* 精度达 1e-15 则停止计算 */
       e=b;                    /* 保存本次正多边形的边长作为下一次精度控制的依据 */
    }
    printf("pai=%.15lf\n", 2*i*b);                       /* 输出 π 值和正多边形的边数 */
    printf("The number of edges of required polygon: %ld\n", i);
}
```

＊ 运行结果

```
pai=3.141592653589793
The number of edges of required polygon: 100663296
```

＊ 思考题

请用外切正多边形逼近的方法求 π 的近似值。

79. 求 π 的近似值(2)

利用随机数法求 π 的近似值。

* 问题分析与算法设计

随机数法求π近似值的思路是:在一个单位边长的正方形中,以边长为半径,以一个顶点为圆心,在正方形上作四分之一圆。随机的向正方形内扔点,若落入四分之一圆内则计数。重复向正方形内扔足够多的点,将落入四分之一圆内的计数除以总的点数,其值就是π值四分之一的近似值。

按此方法可直接进行编程。注意:本方法求出的π值只有统计次数足够多时才可能准确。

* 程序与程序注释

```
#include <time.h>
#include <stdlib.h>
#define N 30000
main()
{ float x, y;
  int c=0, d=0;
  randomize();
  while(c++<=N) {
    x=random(101);              /* x:坐标。产生 0 到 100 之间共 101 个的随机数 */
    y=random(101);              /* y:坐标。产生 0 到 100 之间共 101 个的随机数 */
    if ( x*x+y*y<=10000 )                    /* 利用圆方程判断点是否落在圆内 */
      d++;                                   /* 若落在圆内则计数 */
  }
  printf(" pi=%f\n", 4.*d/N);                /* 输出求出的π值 */
}
```

* 运行结果

多次运行程序,可能得到多个不同的结果,这是因为采用的是统计规律求π的近似值,只有当统计的次数足够大时,才可能逼近π值。运行四次,可能的结果是:

3.122267

3.139733

3.133733

3.106800

80. 奇数平方的一个有趣性质

编程验证"大于 1000 的奇数其平方与 1 的差是 8 的倍数"。

* 问题分析与算法设计

本题是一个很容易证明的数学定理,我们可以编写程序验证之。

题目中给出的处理过程很清楚,算法不需特殊设计,可按照题目的叙述直接进行验证(程序中仅验证到 3000)。

* 程序与程序注释

```
main()
{ long int a;
```

```
  for (a=1001; a<=3000; a+=2) {
    printf("%ld: ", a);                                  /* 输出奇数本身 */
    printf("( %ld * %ld - 1 ) / 8 ", a, a);              /* 输出(奇数的平方-1)/8 */
    printf("= %ld ", (a*a-1)/8);                        /* 输出被8除后的商 */
    printf("+ %ld\n", (a*a-1)%8);                       /* 输出被8除后的余数 */
  }
}
```

* 运行结果

最后十个数的运算结果如下：

```
2981: ( 2981 * 2981 - 1 ) / 8 = 1110795 + 0
2983: ( 2983 * 2983 - 1 ) / 8 = 1112286 + 0
2985: ( 2985 * 2985 - 1 ) / 8 = 1113778 + 0
2987: ( 2987 * 2987 - 1 ) / 8 = 1115271 + 0
2989: ( 2989 * 2989 - 1 ) / 8 = 1116765 + 0
2991: ( 2991 * 2991 - 1 ) / 8 = 1118260 + 0
2993: ( 2993 * 2993 - 1 ) / 8 = 1119756 + 0
2995: ( 2995 * 2995 - 1 ) / 8 = 1121253 + 0
2997: ( 2997 * 2997 - 1 ) / 8 = 1122751 + 0
2999: ( 2999 * 2999 - 1 ) / 8 = 1124250 + 0
```

81. 角谷猜想

日本一位中学生发现一个奇妙的“定理”，请角谷教授证明，而教授无能为力，于是产生角谷猜想。猜想的内容是：任给一个自然数，若为偶数除以2，若为奇数则乘3加1，得到一个新的自然数后按照上面的法则继续演算，若干次后得到的结果必然为1。请编程验证之。

* 问题分析与算法设计

本题是一个尚未获得一般证明的猜想，但屡试不爽，可用程序验证之。

题目中给出的处理过程很清楚，算法不需特殊设计，可按照题目的叙述直接进行验证。

* 程序与程序注释

```
#include "stdio.h"
main()
{ int n, count=0;
  printf("Please enter number:");
  scanf ("%d", &n);                                      /* 输入任一整数 */
  do {
    if (n%2) {
      n=n*3+1;                                           /* 若为奇数,n乘3加1 */
     printf("[%d]: %d*3+1=%d\n", ++count, (n-1)/3,n);
    }
    else {
```

```
        n/=2;                                          /* 若为偶数,n 除以 2 */
        printf("[%d]: %d/2=%d\n", ++count, 2*n,n);
      }
    } while (n! =1);                                   /* n 不等于 1 则继续以上过程 */
}
```

* 运行结果

① Please enter number:10

[1]: 10/2=5
[2]: 5*3+1=16
[3]: 16/2=8
[4]: 8/2=4
[5]: 4/2=2
[6]: 2/2=1

② Please enter number:21

[1]: 21*3+1=64
[2]: 64/2=32
[3]: 32/2=16
[4]: 16/2=8
[5]: 8/2=4
[6]: 4/2=2
[7]: 2/2=1

③ Please enter number:110

[1]: 110/2=55
[2]: 55*3+1=166
[3]: 166/2=83
.....
[111]: 8/2=4
[112]: 4/2=2
[113]: 2/2=1

82. 四方定理

数论中著名的"四方定理"讲的是:所有的自然数至多只要用四个数的平方和就可以表示。请编程验证此定理。

* 问题分析与算法设计

本题是一个定理,我们不去证明它而是编程序验证。

对四个变量采用试探的方法进行计算,满足要求时输出计算结果。

* 程序与程序注释

```
#include <stdio.h>
main ()
```

```
{ int number, i, j, k, l;
  printf("Please enter a number=");
  scanf("%d", &number);                                    /* 输入整数 */
  for(i=1; i<number/2; i++)                    /* 试探法。试探 i、j、k、l 的不同取值 */
    for(j=0; j<=i; j++)
      for(k=0; k<=j; k++)
        for(l=0; l<=k; l++)
          if ( number==i*i+j*j+k*k+l*l ) { /* 若满足定理要求则输出结果 */
            printf(" %d=%d*%d+%d*%d+%d*%d+%d*%d\n",
                                  number,i,i,j,j,k,k,l,l);
            exit(0);                                       /* 退出循环 */
          }
}
```

* 运行结果

① Please enter a number=110

110=7*7+6*6+4*4+3*3

② Please enter a number=211

211=8*8+7*7+7*7+7*7

③ Please enter a number=99

99=7*7+5*5+4*4+3*3

83. 卡布列克常数

验证卡布列克运算。任意一个四位数，只要它们各个位上的数字是不全相同的，就有这样的规律：

①将组成该四位数的四个数字由大到小排列，形成由这四个数字构成的最大的四位数；

②将组成该四位数的四个数字由小到大排列，形成由这四个数字构成的最小的四位数(如果四个数字中含有 0，则得到的数不足四位)；

③求两个数的差，得到一个新的四位数(高位零保留)。

重复以上过程，最后得到的结果总是 6174。这个数被称为卡布列克常数。

* 问题分析与算法设计

题目中给出的处理过程很清楚，算法不需特殊设计，可按照题目的叙述直接进行验证。

* 程序与程序注释

```
int count=0;                                               /* 计数器 */
main()
{ int n;
  printf("Enter a number:");
  scanf("%d",&n);                                      /* 输入任意正整数 */
  vr6174(n);                                         /* 调用函数进行验证 */
}
```

```
vr6174(num)
   int num;
{ int each[4],max,min;
   if ( num! =6174 && num) {   /* 若不等于 6174 且 不等于 0 则进行卡布列克运算 */
     parse _ sort(num, each);                    /* 将整数分解,数字存入 each 数组中 */
     max _ min(each,&max,&min);                  /* 求数字组成的最大值和最小值 */
     num=max _ min;                              /* 求最大值和最小值的差 */
     printf(" [%d]: %d-%d=%d\n", ++count, max, min, num);
                                                 /* 输出该步计算过程 */
     vr6174(num);                                /* 递归调用自身继续进行卡布列克运算 */
   }
}
parse _ sort(num, each)                          /* 函数将整数分解,数字存入 each 数组中 */
   int num, * each;
{  int i, * j, * k,temp;
   for (i=0;i<=4;i++) {                          /* 将 NUM 分解为数字 */
     j=each+3-i;
     * j=num%10;
     num/=10;
   }
   for(i=0;i<3;i++)                              /* 对各个数字从小到大进行排序 */
     for(j=each,k=each+1; j<each+3-i; j++,k++)
       if ( * j> * k) { temp= * j; * j= * k; * k=temp; }
   return;
}
max _ min(each,max,min)                          /* 将分解的数字还原为最大整数和最小整数 */
   int * each, * max, * min;
{ int * i;
   * min=0;
   for (i=each; i<each+4; i++)                   /* 还原为最小的整数 */
     * min= * min * 10+ * i;
   * max=0;
   for (i=each+3; i>=each; i--)                  /* 还原为最大的整数 */
     * max= * max * 10+ * i;
   return;
}
```

* 运行结果

① Enter a number:4321

[1]: 4321-1234=3087

[2]: 8730－378＝8352

[3]: 8532－2358＝6174

② Enter a number:8720

[1]: 8720－278＝8442

[2]: 8442－2448＝5994

[3]: 9954－4599＝5355

[4]: 5553－3555＝1998

[5]: 9981－1899＝8082

[6]: 8820－288＝8532

[7]: 8532－2358＝6174

③ Enter a number:9643

[1]: 9643－3469＝6174

84. 尼科彻斯定理

验证尼科彻斯定理，即：任何一个整数的立方都可以写成一串连续奇数的和。

＊ 问题分析与算法设计

本题是一个定理，我们先来证明它是成立的。

对于任一正整数 a，不论 a 是奇数还是偶数，整数(a×a－a+1)必然为奇数。

构造一个等差数列，数列的首项为(a×a－a+1)，等差数列的差值为 2(奇数数列)，则前 a 项的和为：

$$\begin{aligned} & a \times ((a \times a - a + 1)) + 2 \times a(a-1)/2 \\ &= a \times a \times a - a \times a + a + a \times a - a \\ &= a \times a \times a \end{aligned}$$

定理成立。证毕。

通过定理的证明过程可知：所要求的奇数数列的首项为(a×a－a+1)，长度为 a。编程的算法不需特殊设计，可按照定理的证明过程直接进行验证。

＊ 程序与程序注释

```
main()
{ int a,b,c,d;
  printf("Please enter a number:");
  scanf("%d", &a);                                    /* 输入整数 */
  b = a*a*a;                                          /* 求整数的三次方 */
  printf("%d*%d*%d=%d=", a, a, a, b);
  for (d=0, c=0; c<a; c++) {          /* 输出数列，首项为 a*a-a+1，等差值为 2 */
    d += a*a-a+1 + c*2;                               /* 求数列的前 a 项的和 */
    printf (c ?"+%d" :"%d", a*a-a+1+c*2);
  }
  if (d==b) printf(" Y\n");                           /* 若满足条件则输出"Y" */
  else printf(" N\n");                                /* 否则输出"N" */
```

}

* 运行结果

① Please enter a number:13

13 * 13 * 13=2197=157+159+161+163+165+167+169+171+173+175+177+179+181 Y

② Please enter a number:14

14 * 14 * 14=2744=183+185+187+189+191+193+195+197+199+201+203+205+207+209 Y

* 思考题

本题的求解方法是先证明,在证明的过程中找到编程的算法,然后编程实现。实际上我们也可以不进行证明,直接使用编程中常用的试探的方法来找出该数列,验证该定理。请读者自己设计算法。当然这样得到的数列可能与用定理方法得到的数列不一样。

85. 回文数的形成

任取一个十进制整数,将其倒过来后与原来的整数相加,得到一个新的整数后重复以上步骤,则最终可得到一个回文数。请编程验证。

* 问题分析与算法设计

回文数的这一形成规则目前还属于一个猜想,尚未获得数学上的证明,有些回文数要经历上百个步骤才能获得。这里通过编程验证。

题目中给出的处理过程很清楚,算法不需特殊设计,可按照题目的叙述直接进行验证。

* 程序与程序注释

```
#define MAX 2147483647                                  /* 限定 M+N 的范围 */
main()
{ long int n,m, re();
  int count=0;
  printf("please enter a number optionaly:");
  scanf ("%ld",&n);
  printf("The generation process of palindrome:\n");
  while ( ! nonre((m=re(n))+n) ) {  /* 判断整数与其反序数相加后是否为回文数 */
    if ( m+n >= MAX ) {                                 /* 超过界限输出提示信息 */
        printf(" input error, break.\n");
        break;
    }
    else {
        printf(" [%d]: %ld+%ld=%ld\n", ++count, n, m, m+n);
        n += m;                                                    /* 累加 */
    }
  }
  printf(" [%d]: %ld+%ld=%ld\n", ++count, n, m, m+n);
                                                    /* 输出最后得到的回文数 */
  printf("Here we reached the aim at last ! \n");
```

```
}
long re(a)                                          /* 求输入整数的反序数 */
   long int a;
{ long int t;
   for (t=0; a>0; a/=10)                            /* 将整数反序 */
      t = t*10 + a%10;
   return(t);                                       /* 返回反序后的整数值 */
}
nonre(s)                                            /* 判断给定的整数是否是回文数 */
   long int s;
{ if ( re(s)==s ) return(1);                        /* 若是回文数返回 1 */
   else           return(0);                        /* 否则返回 0 */
}
```

* 运行结果

1. please enter a number optionaly: 1993
   The generation process of palindrome:
   [1]: 1993+3991=5984
   [2]: 5984+4895=10879
   [3]: 10879+97801=108680
   [4]: 108680+86801=195481
   [5]: 195481+184591=380072
   [6]: 380072+270083=650155
   [7]: 650155+551056=1201211
   [8]: 1201211+1121021=2322232
   Here we reached the aim at last !
2. please enter a number optionaly:1994
   The generation process of palindrome:
   [1]: 1994+4991=6985
   [2]: 6985+5896=12881
   [3]: 12881+18821=31702
   [4]: 31702+20713=52415
   [5]: 52415+51425=103840
   [6]: 103840+48301=152141
   [7]: 152141+141251=293392
   Here we reached the aim at last !

第十一章　智力游戏

86. 自动发牌

一副扑克有52张牌，打桥牌时应将牌分给四个人。请设计一个程序完成自动发牌的工作。要求：黑桃用S(Spaces)表示；红桃用H(Hearts)表示；方块用D(Diamonds)表示；梅花用C(Clubs)表示。

* 问题分析与算法设计

按照打桥牌的规定，每人应当有13张牌。在人工发牌时，先进行"洗牌"，然后将洗好的牌按一定的顺序发给每一个人。为了便于计算机模拟，可将人工方式的发牌过程加以修改：先确定好发牌顺序：1、2、3、4；将52张牌顺序编号：黑桃2对应数字0，红桃2对应数字1，方块2对应数字2，梅花2对应数字3，黑桃3对应数字4，红桃3对应数字5，……；然后从52张牌中随机的为每个人抽牌。

这里采用C语言库函数中的随机函数，生成0至51之间的共52个随机数，以产生洗牌后随机发牌的效果。

* 程序与程序注释

```
#include "stdlib.h"
#include "stdio.h"
main()
{ static char n[]={'2','3','4','5','6','7','8','9','T','J','Q','K','A'};
  int a[53], b1[13], b2[13], b3[13], b4[13];
  int b11=0, b22=0, b33=0, b44=0, t=1, m, flag, i;
  while (t<=52) {                                          /* 控制发52张牌 */
    m = random(52);                                        /* 产生0到51之间的随机数 */
    for (flag=1, i=1; i<=t && flag; i++)                   /* 查找新产生的随机数是否已经存在 */
      if (m==a[i]) flag=0;                                 /* flag=1:产生的是新的随机数
                                                              flag=0:新产生的随机数已经存在 */
    if (flag) {
    a[t++]=m;                                              /* 如果产生了新的随机数,则存入数组 */
    if (t%4==0) b1[b11++]=a[t-1];                          /* 根据t的模值,判断当前 */
    else if (t%4==1) b2[b22++]=a[t-1];                     /* 的牌应存入哪个数组中 */
    else if (t%4==2) b3[b33++]=a[t-1];
    else if (t%4==3) b4[b44++]=a[t-1];
    }
  }
```

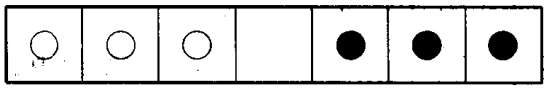
```
    qsort(b1, 13, sizeof(int), comp);                    /* 将每个人的牌进行排序 */
    qsort(b2, 13, sizeof(int), comp);
    qsort(b3, 13, sizeof(int), comp);
    qsort(b4, 13, sizeof(int), comp);
    p(b1,n); p(b2,n); p(b3,n); p(b4,n);                  /* 分别打印每个人的牌 */
}
p(b,n)
    char n[];int b[];
{ int i;
    printf("\n\006");                                    /* 打印黑桃标记 */
    for (i=0; i<13; i++)                                 /* 将数组中的值转换为相应的花色 */
      if(b[i]/13==0) printf(" %c ",n[b[i]%13]);          /* 输出该花色对应的牌 */
    printf("\n\003");                                    /* 打印红桃标记 */
    for (i=0; i<13; i++)
      if((b[i]/13)==1) printf(" %c ",n[b[i]%13]);
    printf("\n\004");                                    /* 打印方块标记 */
    for (i=0; i<13; i++)
      if(b[i]/13==2) printf(" %c ",n[b[i]%13]);
    printf("\n\005");                                    /* 打印梅花标记 */
    for (i=0; i<13; i++)
      if(b[i]/13==3 || b[i]/13==4) printf(" %c ",n[b[i]%13]);
    printf("\n");
}
comp (int *j, int *i)                                   /* qsort 调用的排序函数 */
{ return (*i-*j);}
* 运行示例
    S K J 8
    H A J 5 3
    D Q 8
    C K J 7 5

    S A T 6 4 2
    H 4 2
    D 7 6 4
    C Q T 9

    S 9 7 5 3
    H K Q T 9
    D J 3 2
```

C 8 2

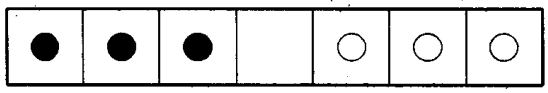
S Q
H 8 7 6
D A K T 9 5
C A 6 4 3

87. 黑白子交换

有三个白子和三个黑子如下图布置：



游戏的目的是用最少的步数将上图中白子和黑子的位置进行交换：



游戏的规则是：(1)一次只能移动一个棋子；(2)棋子可以向空格中移动，也可以跳过一个对方的棋子进入空格，但不能向后跳，也不能跳过两个子。请用计算机实现上述游戏。

* 问题分析与算法设计

计算机解决这类问题的关键是要找出问题的规律，或者说是要制定一套计算机行动的规则。分析本题，先用人来解决问题，可总结出以下规则：

①黑子向左跳过白子落入空格，转⑤；

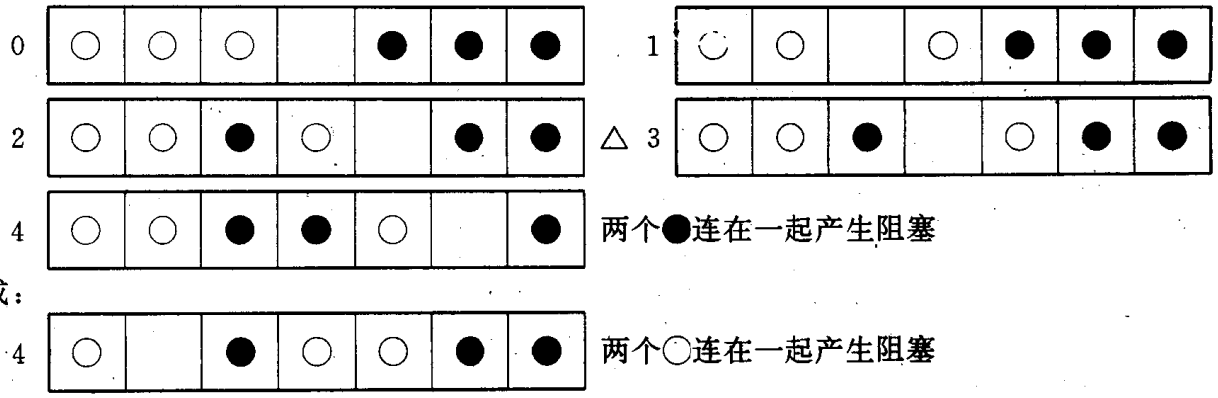
②白子向右跳过黑子落入空格，转⑤；

③黑子向左移动一格落入空格(但不应产生棋子阻塞现象)，转⑤；

④白子向右移动一格落入空格(但不应产生棋子阻塞现象)，转⑤；

⑤判断游戏是否结束，若没有结束，则转①继续。

所谓的“阻塞”现象就是：在移动棋子的过程中，两个尚未到位的同色棋子连接在一起，使棋盘中的其它棋子无法继续移动。例如按下列方法移动棋子：



产生“阻塞”现象的原因是在第 2 步(△状态)时，棋子○不能向右移动，只能将●向左移动。

总结产生“阻塞”的原因，当棋盘出现“黑、白、空、黑”或“白、空、黑、白”状态时，不能向“左”或向“右”移动中间的棋子，只移动两边的子。

按照上述规则，可以保证在移动棋子的过程中，不会出现棋子无法移动的现象，且可以用最少的步数完成白子和黑子的位置交换。

* 程序与程序注释

```
int number;
print(a)                                                    /* 打印棋盘的状态 */
   int a[];
{ int i;
   printf("No. %2d:------------------------------------------------\n", number++);
   printf(" ");
   for (i=0; i<=6; i++)
     printf ("| %c ", a[i]==1 ? '*' : (a[i]==2 ? '@' : ' ') );
   printf("|\n ------------------------------------------------\n");
}
main()
{ int t[7]={1, 1, 1, 0, 2, 2, 2};
                                                /* 初始化数组 1:白子 2:黑子 0:空格 */
   int i, flag;
   print(t);
   while ( t[0]+t[1]+t[2]! =6 || t[4]+t[5]+t[6]! =3 ) {
                                                        /* 判断游戏是否结束 */
                                            /* 若还没有完成棋子的交换则继续进行循环 */
     flag=1;          /* flag 为棋子移动一步的标记 1:尚未移动棋子 0:已经移动棋子 */
     for(i=0; flag && i<5; i++)          /* 若白子可以向右跳过黑子，则白子向右跳 */
       if (t[i]==1 && t[i+1]==2 && t[i+2]==0)
         { change(&t[i], &t[i+2]); print(t); flag=0;}
     for(i=0; flag && i<5; i++)          /* 若黑子可以向左跳过白子，则黑子向左跳 */
       if (t[i]==0 && t[i+1]==1 && t[i+2]==2)
         { change(&t[i],&t[i+2]); print(t); flag=0;}
     for(i=0; flag && i<6; i++)
                                        /* 若向右移动白子不会产生阻塞，则白子向右移动 */
       if (t[i]==1 && t[i+1]==0 && ( i==0 || t[i-1]! =t[i+2]) )
         { change(&t[i], &t[i+1]); print(t); flag=0;}
       for(i=0; flag && i<6; i++)
                                        /* 若向左移动黑子不会产生阻塞，则黑子向左移动 */
       if (t[i]==0 && t[i+1]==2 && ( i==5 || t[i-1]! =t[i+2]) )
         { change(&t[i], &t[i+1]); print(t); flag=0; }
   }
```

```
}
change(n,m)                                    /* 完成交换两个整数的功能 */
   int *n, *m;
{ int term;
   term=*n; *n=*m; *m=term;
}
```

* 运行结果

No. 0:	*	*	*		@	@	@
No. 1:	*	*		*	@	@	@
No. 2:	*	*	@	*		@	@
No. 3:	*	*	@	*	@		@
No. 4:	*	*	@		@	*	@
No. 5:	*		@	*	@	*	@
No. 6:		*	@	*	@	*	@
No. 7:	@	*		*	@	*	@
No. 8:	@	*	@	*		*	@
No. 9:	@	*	@	*	@	*	
No. 10:	@	*	@	*	@		*
No. 11:	@	*	@		@	*	*
No. 12:	@		@	*	@	*	*
No. 13:	@	@		*	@	*	*
No. 14:	@	@	@	*		*	*
No. 15:	@	@	@		*	*	*

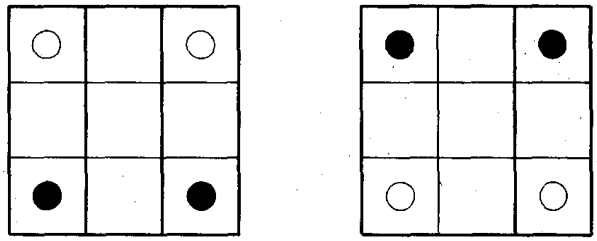
* 问题的进一步讨论

本题中的规则不仅适用于三个棋子的情况，而且可以推而广之，适用于任意 N 个棋子的情况。读者可以编程验证，按照本规则得到的棋子移动步数是最少的。

事实上，制定规则是解决这类问题的关键。一个游戏程序“思考”水平的高低，完全取决于使用规则的好坏。

* 思考题

有两个白子和两个黑子如下左图布置：



棋盘中的棋子按“马步”规则行走，要求用最少的步数将图中白子和黑子的位置进行交换，最终结果如右图所示。

88. 常胜将军

现有 21 根火柴，两人轮流取，每人每次可以取 1 至 4 根，不可多取，也不能不取，谁取最后一根火柴谁输。请编写一个程序进行人机对弈，要求人先取，计算机后取；计算机一方为“常胜将军”。

* 问题分析与算法设计

在计算机后走的情况下，要想使计算机成为“常胜将军”，必须找出取胜的关键。根据本题的要求可以总结出：后走一方取子的数量与对方刚才一步取子的数量之和等于 5，就可以保证最后一个子是留给先取子的那个人的。

据此分析进行算法设计就是很简单的工作，编程实现亦十分容易。

* 程序与程序注释

```
main()
{ int a=21, i;
  printf("Game begin:\n");
  while(a>0) {
    do {
      printf(" How many sticks do you wish to take (1~%d) ? ", a>4 ? 4: a );
      scanf("%d", &i);
    } while ( i>4 || i<1 || i>a );                    /* 接收正确的输入 */
    if (a-i>0) printf(" %d sticks left in the pile. \n", a-i);
    if ((a-i)<=0) {
      printf(" You have taken the last sticks. \n");
      printf(" * * * You lose! \nGame Over. \n");     /* 输出取胜标记 */
      break;
    }
    else
      printf(" Computer take %d sticks. \n",5-i);     /* 输出计算机取的子数 */
    a -= 5;
```

```
        printf(" %d sticks left in the pile. \n", a);
    }
}
```

* 运行示例

```
Game begin:
How many sticks do you wish to take (1～4) ? 2
    19 sticks left in the pile.
Computer take 3 sticks.
    16 sticks left in the pile.
How many sticks do you wish to take (1～4) ? 3
    13 sticks left in the pile.
Computer take 2 sticks.
    11 sticks left in the pile.
How many sticks do you wish to take (1～4) ? 4
    7 sticks left in the pile.
Computer take 1 sticks.
    6 sticks left in the pile.
How many sticks do you wish to take (1～4) ? 1
    5 sticks left in the pile.
Computer take 4 sticks.
    1 sticks left in the pile.
How many sticks do you wish to take (1～1) ? 1
    You have taken the last sticks.
    * * * You lose!
Game Over.
```

* 思考题

改变题目中火柴的数量(如为 22 根),则后走的一方就不一定能够保持常胜了,很可能变成"常败"。此时,后走一方的胜负就与火柴的初始数量和每次允许取的火柴数量的最大值有直接关系,请编写程序解决这一问题。

89. 抢 30

这是中国民间的一个游戏。两人从 1 开始轮流报数,每人每次可报一个数或两个连续的数,谁先报到 30,谁就为胜方。

* 问题分析与算法设计

本题与上题类似,算法也类似。所不同的是,本题谁先走第一步是可选择的。若计算机走第一步,那么计算机一定是赢家。若人先走第一步,那么计算机只好等待人犯错误,如果人先走第一步且不犯错误,那么人就会取胜;否则,计算机会抓住人的一次错误使自己成为胜利者。

* 程序与程序注释

```
#include "time. h"
```

```
#include "stdlib.h"
main()
{ int tol = 0 ;
  printf("\n * * * * * * * catch thirty * * * * * * *\n");
  printf("Game Begin\n");
  randomize();                                          /* 初始化随机数 */
  if ( random(2) == 1 )                          /* 取随机数决定机器和人谁先走第一步 */
    tol = input(tol);                                  /* 若为1,则人先走第一步 */
  while( tol! =30 )                                     /* 游戏结束的条件 */
    if ( (tol=copu(tol)) == 30 )                 /* 计算机取一个数,若为30则机器胜利 */
      printf("You lose! \n");
    else
      if ( (tol=input(tol)) == 30 )                    /* 人取一个数,若为30则人胜利 */
        printf("I lose! \n");
  printf(" * * * * * * * * * * * Game Over * * * * * * *\n");
}
input(t)                                                /* 控制人取一个正确的数 */
  int t;
{ int a;
  do {
      printf(" Please count:");
      scanf("%d",&a);
      if ( a > 2 || a < 1 || t+a>30 )
        printf(" Error input, again! ");
      else
        printf(" You count: %d \n",t+a);
  } while ( a > 2 || a < 1 || t+a>30 );
  return (t + a);                              /* 返回当前的已经取走的数的累加和 */
}
copu(s)                                                 /* 控制机器取一个正确的数 */
  int s;
{ int c;
  printf(" Computer count:");
  if( ( s + 1 ) % 3 == 0 )                          /* 若剩余的数的模为1,则取1 */
      printf(" %d\n", ++s);
  else if( ( s + 2 ) % 3 == 0 ) {
      s += 2;                                       /* 若剩余的数的模为2,则取2 */
      printf(" %d\n", s);
      }
```

```
        else {
          c = random.(2) + 1;                          /* 否则随机取 1 或 2 */
          s += c;
          printf (" %d\n", s);
        }
    return (s);                                  /* 返回当前的已经取走的数的累加和 */
}
* 运行示例
    * * * * * * * catch thirty * * * * * * *
    Game Begin
      Please count:2
            You count: 2
    Computer count: 3
    Please count:2
            You count: 4
    Computer count: 6
    Please count:2
            You count: 8
    Computer count: 9
    Please count:2
            You count: 11
    Computer count: 12
    Please count:2
            You count: 14
    Computer count: 15
    Please count:1
            You count: 16
    Computer count: 18
    Please count:1
            You count: 19
    Computer count: 21
    Please count:1
            You count: 22
    Computer count: 24
    Please count:2
            You count: 26
    Computer count: 27
    Please count:2
            You count: 29
```

Computer count: 30

I lose!

* * * * * * * * * * * Game Over * * * * * * *

* 思考题

巧夺偶数。桌子上有 25 颗棋子，游戏双方轮流取子，每人每次最少取走一颗棋子，最多可取走三颗棋子。双方照这样取下去，直到取光全部棋子。于是双方手中必然一方为偶数，一方为奇数，偶数方为胜者。请编程实现人机游戏。

90. 搬山游戏

设有 n 座山，计算机与人为比赛的双方，轮流搬山。规定每次搬山的数目不能超过 k 座，谁搬最后一座谁输。游戏开始时，计算机请人输入山的总数(n)和每次允许搬山的最大数目(k)。然后请人先开始，等人输入了需要搬走的山的数目后，计算机马上打印出它搬多少座山，并提示尚余多少座山。双方轮流搬山直到最后一座山搬完为止。计算机会显示谁是赢家，并问人是否要继续比赛。若人不想玩了，计算机便会统计出共玩了几局，双方胜负如何。

* 问题分析与算法设计

计算机参加游戏时应遵循如下原则：

① 当：

剩余山数目 − 1 <= 可移动的最大数 k

时，计算机要移(剩余山数目−1)座，以便将最后一座山留给人。

② 对于任意正整数 x，y，一定有：

$$0 <= x\%(y+1) <= y$$

在有 n 座山的情况下，计算机为了将最后一座山留给人，而且又要控制每次搬山的数目不超过最大数 k，它应搬山的数目要满足下列关系：

$$(n-1)\%(k+1)$$

如果算出结果为 0，即整除无余数，则规定只搬 1 座山，以防止冒进后发生问题。

按照这样的规律，可编写出游戏程序如下：

* 程序与程序注释

```
#include <stdio.h>
main()
{ int n, k, x, y, cc, pc, g;
  printf(" Move Mountain Game\n");
  printf("Game Begin\n");
  pc=cc=0;
  g=1;
  for ( ; ; ) {
      printf(" No. %2d game \n", g++);
      printf(" ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
      printf(" How many mountains are there ? ");
```

```
        scanf("%d", &n);
        if (! n) break;
        printf(" How many mountains are allowed to each time ? ");
        do {
          scanf ("%d", &k);
          if ( k>n || k<1 ) printf(" Repeat again ! \n");
        } while ( k>n || k<1 );
        do {
          printf(" How many mountains do you wish to move away ?");
          scanf("%d", &x);
          if (x<1 || x>k || x>n) {                    /* 判断搬山数是否符合要求 */
            printf(" Illegal , again please! \n");
            continue;
          }
          n -=x;                                       /* 求出所剩山数 */
          printf(" There are %d mountains left now. \n", n);
          if (! n) {
            printf(" ····· I win. You are failure. ·····\n\n");cc++;
          }
          else {
            y =(n-1) % (k+1);                          /* 求出最佳搬山数 */
            if (! y) y=1;
            n -=y;
            printf(" Computer move %d mountains away. \n", y);
            if (n) printf(" There are %d mountains left now. \n", n);
            else {
              printf(" ····· I am failure. You win. ····· \n\n");
              pc++;
            }
          }
        } while (n);
    }
    printf("Games in total have been played %d. \n", cc+pc);
    printf(" Your score is win %d , lose %d. \n", pc, cc);
    printf(" My score is win %d , lose %d. \n", cc, pc);
}
```

* 运行示例

```
    Move Mountain Game
  Game Begin
```

No. 1 game

~~~~~~

How many mountains are there ? 10
How many mountains are allowed to each time ? 3
How many mountains do you wish to move away ? 1
There are 9 mountains left now.
Computer move 1 mountains away.
There are 8 mountains left now.
How many mountains do you wish to move away ? 3
There are 5 mountains left now.
Computer move 1 mountains away.
There are 4 mountains left now.
How many mountains do you wish to move away ? 3
There are 1 mountains left now.
Computer move 1 mountains away.
..... I am failure. You win. .....

No. 2 game

~~~~~~

How many mountains are there ? 0
Games in total have been played 1.
Your score is win 1 , lose 0.
My score is win 0 , lose 1.

* 思考题

取石子游戏。将石子分成若干堆，每堆有若干粒，参加游戏的甲乙两方轮流从任意一堆中取走任意个石子，甚至可以全部取走，但每次只能在一堆中取，不允许从这堆取一些，再从另一堆中取一些。直到谁取走最后一颗石子谁就获胜。请编程进行人机对弈。

91. 人机猜数游戏(1)

由计算机“想”一个四位数，请人猜这个四位数是多少。人输入四位数字后，计算机首先判断这四位数字中有几位是猜对了，并且在对的数字中又有几位位置也是对的，将结果显示出来，给人以提示，请人再猜，直到人猜出计算机所想的四位数是多少时为止。

例如：计算机“想”了一个“1234”请人猜，可能的提示如下：

人猜的整数	计算机判断有几个数字正确	有几个位置正确
1122	2	1
3344	2	1
3312	3	0

4123	4	0
1243	4	2
1234	4	4

游戏结束

请编程实现该游戏。游戏结束时，显示人猜一个数用了几次。

* 问题分析与算法设计

问题本身清楚明了。判断相同位置上的数字是否相同不需要特殊的算法，只要截取相同位置上的数字进行比较即可。但在判断几位数字正确时，则应当注意：若计算机所想的是“1123”，而人所猜的是“1567”，则正确的数字只有1位。

程序中截取计算机所想的数的每位数字与人所猜的数按位比较。若有两位数字相同，则要记住所猜数中数字的位置，使该位数字只能与一位对应的数字“相同”。当截取下一位数字进行比较时，就不应再与上述位置上的数字进行比较，以避免所猜的数中的一位数字与对应数中多位数字“相同”的错误情况。

* 程序与程序注释

```
#include "time.h"
#include "stdlib.h"
main()
{int stime,a,z,t,i,c,m,g,s,j,k,l[4];        /* j:数字正确的位数  k:位置正确的位数 */
   long ltime;
   ltime =time(NULL);                       /* l:数字相同时,人所猜中数字的正确位置 */
   stime = (unsigned int) ltime/2;
   srand(stime);
   z = random(9999);                                  /* 计算机想一个随机数 */
   printf("I have a number with 4 digits in mind, please guess.\n");
   for (c=1;;c++) {                                   /* c:猜数次数计数器 */
     printf(" Enter a number with 4 digits:");
     scanf("%d",&g);                                  /* 请人猜 */
     a=z; j=0; k=0; l[0]=l[1]=l[2]=l[3]=0;
     for (i=1; i<5; i++) {    /* i:原数中的第i位数。个位为第1位,千位为第4位 */
       s=g; m=1;
       for (t=1; t<5; t++) {                        /* 人所猜数的位数 */
         if ( a%10 == s%10 ) {                      /* 若第i位与人猜的第t位相同 */
           if (m && t! =l[0] && t! =l[1] && t! =l[2] && t! =l[3] ) {
             j++; m=0; l[j-1]=t;    /* 若该位置上的数字尚未与其它数字“相同” */
           }                        /* 记录数字相同时,该数字在所猜数中的位置 */
           if (i==t) k++;                           /* 若位置也相同,则计数器k加1 */
         }
         s/=10;
       }
```

```
        a/=10;
      }
      printf(" You have correctly guessed %d digits,\n" ,j);
      printf(" and correctly guessed %d digits in exact position. \n", k);
      if (k==4) break;                    /* 若位置全部正确,则人猜对了;退出 */
    }
    printf("Now you have correctly guessed the whole number after %d times. \n",c);
}
```

* 运行示例

```
I have a number with 4 digits in mind, please guess.
  Enter a number with 4 digits: 1122
    You have correctly guessed 0 digits,
      and correctly guessed 0 digits in exact position.
  Enter a number with 4 digits: 3344
    You have correctly guessed 1 digits,
      and correctly guessed 0 digits in exact position.
  Enter a number with 4 digits: 5566
    You have correctly guessed 1 digits,
      and correctly guessed 0 digits in exact position.
  Enter a number with 4 digits: 3578
    You have correctly guessed 0 digits,
      and correctly guessed 0 digits in exact position.
  Enter a number with 4 digits: 4690
    You have correctly guessed 3 digits,
      and correctly guessed 3 digits in exact position.
  Enter a number with 4 digits: 4699
    You have correctly guessed 2 digits,
and correctly guessed 2 digits in exact position.
  Enter a number with 4 digits: 4600
  You have correctly guessed 4 digits,
    and correctly guessed 4 digits in exact position.
  Now you have correctly guessed the whole number after 7 times.
```

* 思考题

猜数游戏。由计算机"想"一个数请人猜,人输入猜的数,如果猜对了,则结束游戏,否则计算机给出提示,告诉人猜的数是太大,还是太小。当一个数猜了 20 次还未猜中时,应停止猜数者继续游戏的权力,从程序中退出。

92. 人机猜数游戏(2)

将以上游戏双方倒一下,请人想一个四位的整数,计算机来猜,人给计算机提示信息,最终

看计算机用几次猜出一个人“想”的数。请编程实现。

* 问题分析与算法设计

解决这类问题时，计算机的思考过程不可能象人一样具备较完备的推理能力，关键在于要将推理和判断的过程变成一种机械的过程，找出相应的规则，否则计算机难于完成推理工作。

基于对问题的分析和理解，将问题进行简化，求解分为两步完成：首先确定四位数字的组成，然后再确定四位数字的排列顺序。可以列出如下规则：

①分别显示四个 1，四个 2，……，四个 0，确定四位数字的组成。

②依次产生四位数字的全部排列（依次两两交换全部数字的位置）。

③根据人输入的正确数字及正确位置的数目，进行分别处理：

（注意此时不出现输入 3 的情况，因为在四个数字已经确定的情况下，若有 3 个位置正确，则第四个数字的位置必然也是正确的）。

若输入 4：游戏结束。

判断本次输入与上次输入的差值

若差为 2：说明前一次输入的一定为 0，本次输入的为 2，本次交换的两个数字的位置是正确的，只要交换另外两个没有交换过的数字即可结束游戏。

若差为－2：说明前一次输入的一定为 2，本次输入的一定为 0。说明刚交换的两个数字的位置是错误的，只要将刚交换的两个数字位置还原，并交换另外两个没有交换过的数字即可结束游戏。

否则：若本次输入的正确位置数＜＝上次输入的正确位置数

则恢复上次四位数字的排列，控制转③。

否则：将本次输入的正确位置数作为“上次输入的正确位置数”，控制转③。

* 程序与程序注释

```
int a[4], flag ,count;
main()
{ int b1, b2, i, j, k=0, p, c;
   printf("Game Begin\n");
   printf("Now guess your number in mind is ####.\n");
   for (i=1; i<10 && k<4; i++) {    /* 分别显示四个 1 到 9 确定四个数字的组成 */
     printf("No. %d: your number may be: %d%d%d%d\n",++count, i, i, i, i);
     printf(" How many digits have had correctly guessed :");
     scanf ("%d", &p);                              /* 人输入包含几位数字 */
     for (j=0; j<p; j++)
       a[k+j] = i;                                  /* a[]:存放已确定数字的数组 */
     k += p;                                        /* k:已确定的数字个数 */
   }
   if (k<4)                                         /* 自动算出四位中包含 0 的个数 */
     for(j=k;j<4;j++)
       a[j]=0;
   i=0;
```

```
    printf("No. %d: your number may be: %d%d%d%d\n",
                              ++count,a[0],a[1],a[2],a[3]);
    printf(" How many are in exact positions :");          /* 顺序显示四位数字 */
    scanf ("%d", &b1);                                     /* 人输入有几位位置是正确的 */
    if (b1==4) { prt(); exit(0); }                         /* 四位正确,打印结果,结束游戏 */
    for (flag=1,j=0; j<3 && flag; j++)   /* 实现四个数字的两两(a[j],a[k])交换 */
      for (k=j+1; k<4 && flag; k++)                        /* flag:结束标记 */
        if ( a[j]! =a[k] ) {                 /* 只有两个数字不相同时才进行交换 */
          c=a[j]; a[j]=a[k]; a[k]=c;                       /* 交换(a[j],a[k]) */
          printf("No. %d: your number may be: %d%d%d%d\n",
                                        ++count, a[0], a[1], a[2], a[3]);
          printf(" How many are in exact positions :");
          scanf("%d", &b2);                                /* 输入有几个位置正确 */
          if (b2==4) { prt(); flag=0; }                    /* 若全正确,结束游戏 */
          else if(b2-b1==2) bhdy(j,k);
                              /* 若上次与本次的差为2,则交换另外两个元素即可结束 */
          else if(b2-b1==-2) {
                    /* 若上次与本次的差为-2,则说明已交换的(a[j],a[k])是错误的,
                       将(a[j],a[k])还原后,只要交换另外两个元素即可结束游戏 */
          c=a[j]; a[j]=a[k]; a[k]=c;
          bhdy(j,k);
        }
        else if ( b2<=b1 ) {
              c=a[j]; a[j]=a[k]; a[k]=c;                   /* 恢复交换的两个数字 */
        }
        else b1=b2;          /* 其它情况则将新输入的位置信息作为上次的位置保存 */
      }
    if (flag) printf("You input error! \n");
                                  /* 交换结束仍没结果,只能是人输入的信息错误 */
}
prt()                                                      /* 打印结果,结束游戏 */
{ printf(" Now your number must be %d%d%d%d. \n", a[0], a[1], a[2], a[3]);
    printf("Game Over\n");
}
bhdy(s,b)                  /* 交换除a[s]和a[b]以外的另外两个元素,并结束游戏 */
    int s,b;
{ int i, c=0, d[2];
    for (i=0; i<4; i++)                          /* 查找s和b以外的两个元素下标 */
      if ( i! =s && i! =b ) d[c++]=i;
```

```
    i=a[d[1]]; a[d[1]]=a[d[0]]; a[d[0]]=i;
                                        /* 交换除 a[s]和 a[b]以外的两个元素 */
    prt();                                  /* 打印结果,结束游戏 */
    flag=0;
}
```

* 运行示例

假设人想的四位数为:7215。

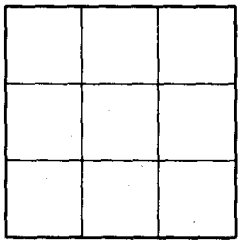
```
Game Begin
Now guess your number in mind is # # # #.
No. 1: your number may be: 1111
        How many digits have had correctly guessed :1
No. 2: your number may be: 2222
        How many digits have had correctly guessed :1
No. 3: your number may be: 3333
        How many digits have had correctly guessed :0
No. 4: your number may be: 4444
        How many digits have had correctly guessed :0
No. 5: your number may be: 5555
        How many digits have had correctly guessed :1
No. 6: your number may be: 6666
        How many digits have had correctly guessed :0
No. 7: your number may be: 7777
        How many digits have had correctly guessed :1
No. 8: your number may be: 1257
        How many are in exact positions :1
No. 9: your number may be: 2157
        How many are in exact positions :0
No. 10: your number may be: 5217
        How many are in exact positions :2
No. 11: your number may be: 7215
        How many are in exact positions :4
Now your number must be 7215.
Game Over
```

* 问题的进一步讨论

本程序具有逻辑结构清晰、算法简单正确的优点。但在接受人的输入信息时缺少必要的出错保护功能,同时在进行第 3 步推理过程中,没有保留每次猜出的数字位置信息及人输入的回答,这样对于每次人输入的信息就无法进行合法性检验,即无法检查人的输入信息是否自相矛盾;同时也无法充分利用前面的结果。

这些缺陷是可以改进的,但最后一个问题改进难度较大,留给读者发挥才干。

＊ 思考题

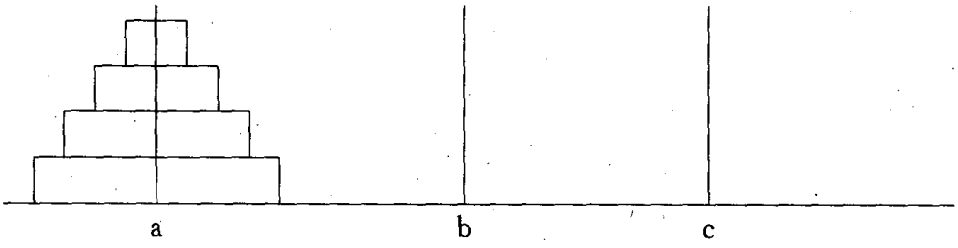


“一条龙游戏”。在一个3×3的棋盘上，甲乙双方进行对弈，双方在棋盘上轮流放入棋子，如果一方的棋子成一直线(横、竖或斜线)，则该方赢。请编写该游戏程序实现人与机器的比赛；比赛结果有三种结果：输、赢或平。

在编程过程中请首先分析比赛中怎样才能取胜，找出第一步走在什么位置就最可能赢。

93. 汉诺塔

约19世纪末，在欧洲的商店中出售一种智力玩具，在一块铜板上有三根杆，最左边的杆上自上而下、由小到大顺序串着由64个圆盘构成的塔，游戏的目的是将最左边杆上的盘全部移到最右边的杆上，条件是一次仅能移动一个盘，且不允许大盘放在小盘的上面。



汉 诺 塔

＊ 问题分析与算法设计

这是一个著名的问题，几乎所有讲述递归算法的教材上都有这个问题。由于条件是一次仅能移动一个盘，且不允许大盘放在小盘的上面，所以64个盘子的移动次数是：

18,446,744,073,709,551,615

这是一个天文数字，若每一微秒可能计算(并不输出)一次移动，那么也需要几乎一百万年。我们仅能找出问题的解决方法并解决较小N值时的汉诺塔，但根本不可能用计算机解决64层的汉诺塔。

分析问题，找出移动盘子的正确算法。

首先考虑a杆最下面的盘子而非杆上最上面的盘子，于是任务就变成了：

- 将上面的63个盘子移到b杆上；
- 将a杆上剩下的盘子移到c杆上；
- 将b杆上的全部盘子移到c杆上。

将这个过程继续下去，就是要先完成移动63个盘子、62个盘子、61个盘子……的工作。

为了更清楚地描述算法，可以定义一个函数movedisc(n，a，b，c)。该函数的功能是：将N个盘子从A杆上借助C杆移动到B杆上。这样移动N个盘子的工作就可以按以下过程进行：

①movedisc (n－1，a，c，b)；

②将一个盘子从a移动到b上；

③movedisc (n－1，c，b，a)；

重复以上过程，直到将全部盘子移动到位时为止。

```
* 程序与程序注释
int i=0;
main()
{ unsigned n;
   printf("Please enter the number of discs:");
   scanf("%d", &n);                                          /* 输入N值 */
   printf("\tneedle:\ta\t b\t c\n");
   movedisc (n, 'a', 'c', 'b');          /* 从A上借助B将N个盘子移动到C上 */
   printf("\t Total: %d\n", i);
}
movedisc( n, fromneedle, toneedle, usingneedle)
                /* movedisc 函数完成的功能是:将 fromneedle 杆上的N个盘子借助 */
                                        /* usingneedle 杆移动到 toneedle 杆上 */
   unsigned n;
   char fromneedle, toneedle, usingneedle;
{
   if (n>0) {
      movedisc(n-1, fromneedle, usingneedle, toneedle);
            /* 从fromneedle 上借助 toneedle 将N-1个盘子移动到 usingneedle 上 */
      ++i;
      switch (fromneedle) {          /* 将fromneedle 上的一个盘子移到 toneedle 上 */
      case 'a': switch (toneedle) {
            case 'b':printf("\t[%d]:\t%2d ----------->%2d\n", i, n, n);
                  break;
             case 'c': printf("\t[%d]:\t%2d ----------->%2d\n", i, n, n);
                  break;
            }
            break;
      case 'b': switch (toneedle) {
            case 'a': printf("\t[%d]:\t%2d<----------%2d\n", i, n, n);
                  break;
            case 'c': printf("\t[%d]:\t\t%2d ----------->%2d\n", i, n, n);
                  break;
            }
            break;
      case 'c': switch (toneedle) {
            case 'a': printf("\t[%d]:\t%2d<----------%2d\n", i, n, n);
                  break;
            case 'b': printf("\t[%d]:\t\t%2d<----------%2d\n", i, n, n);
```

```
                    break;
                }
                break;
        }
        movedisc (n-1, usingneedle, toneedle, fromneedle);
                /* 从 usingneedle 上借助 fromneedle 将 N-1 个盘子移动到 toneedle 上 */
    }
}
```

* 运行结果

```
Please enter the number of discs: 4
needle:a        b        c
[1]:  1 ------>1
[2]:  2 ---------------->2
[3]:            1 ------>1
[4]:  3 ------>3
[5]:   1----------------1
[6]:   2<------2
[7]:  1 ------>1
[8]:  4 ---------------->4
[9]:            1 ------> 1
[10]:  2<------2
[11]:  1----------------1
[12]:           3 ------>3
[13]: 1 ------>1
[14]: 2 ---------------->2
[15]:           1 ------>1
Total: 15
```

第十二章　其它趣味程序

94. 兔子产子

从前有一对长寿的兔子，他们每一个月生一对小兔子，新生的小兔子两个月就长大了，在第二个月的月底就开始生他们的下一代小兔子，这样一代一代生下去。求解兔子增长数量的数列。

* 问题分析与算法设计

问题可以抽象成下列数学公式：

$$U_n = U_{n-1} + U_{n-2}$$

其中：

n 是项数(n>=3)。它就是著名的菲波那奇数列，该数列的前几项为：1,1,2,3,5,8,13,21,....。

菲波那奇数列在程序中可以用多种方法进行处理。按照其通项递推公式利用最基本的循环控制就可以实现题目的要求。

* 程序与程序注释

```
main()
{ int n,i,un1,un2,un;
  for (n=2; n<3; ) {
    printf ("Please enter required number of generation:");
    scanf ("%d", &n);
    if ( n < 3 ) printf ("\n Enter error ! \n");
  }                                          /* 控制输入正确的 N 值 */
  un=un2=1;
  printf ("The repid increase of rabbits in first %d generations is as felow:\n",n);
  printf("1\t1\t");
  for ( i=3; i<=n; i++) {
    un1 = un2;
    un2 = un;
    un = un1 + un2;                          /* 利用通项公式求解 N 项的值 */
    printf ( i%10 ? "%d\t" : "%d\n", un);
  }
  printf("\n");
}
```

* 运行结果

```
128
```

Please enter required number of generation: 20

The repid increase of rabbits in first 20 generations is as felow:

1	1	2	3	5	8	13	21	34	55
89	144	233	377	610	987	1597	2584	4181	6765

95. 将阿拉伯数翻译为罗马数字

将大于 0 小于 1000 的阿拉伯数转换为罗马数字。阿拉伯数字与罗马数字对应关系如下：

1	2	3	4	5	6	7	8	9
I	II	III	IV	V	VI	VII	VIII	IX
10	20	30	40	50	60	70	80	90
X	XX	XXX	XL	L	LX	LXX	LXXX	XCC
100	200	300	400	500	600	700	800	900
C	CC	CCC	CD	D	DC	DCC	DCCC	CM

* 问题分析与算法设计

题目中给出了阿拉伯数与罗马数字的对应关系，题中的数字转换实际上就是查表翻译。即将整数的百、十、个位依次从整数中分解出来，查找表中相应的行后输出对应的字符。

* 程序与程序设计

```
main()
{ static char *a[][10]={"","I","II","III","IV","V","VI","VII","VIII","IX",
                        "","X","XX","XXX","XL","L","LX","LXX","LXXX","XCC",
                        "","C","CC","CCC","CD","D","DC","DCC","DCCC","CM"
                        };                                    /* 建立对照表 */
  int n, t, i, m;
  printf ("Please enter number:");
  scanf ("%d",&n);                                            /* 输入整数 */
  printf ("%d=", n);
  for (m=0, i=1000; m<3; m++, i/=10) {
    t = (n%i)/(i/10);                          /* 从高位向低位依次取各位的数字 */
    printf("%s", a[2-m][t]);                          /* 通过对照表翻译输出 */
  }
  printf ("\n");
}
```

* 运行结果

```
1. Please enter number:863
   863=DCCCLXIII
2. Please enter number:256
   256=CCLVI
```

3. Please enter number:355
   355=CCCLV
4. Please enter number:522
   522=DXXII
5. Please enter number:15
   15=XV

* 思考题

输入正整数 N,产生对应的英文字符串并输出。例如:

1 ONE 2 TWO 3 THREE
10 TEN 11 ELEVEN
135 ONE HUNDRED THIRTY FIVE

96. 选美比赛

在选美大奖赛的半决赛现场,有一批选手参加比赛,比赛的规则是最后得分越高,名次越低。当半决赛结束时,要在现场按照选手的出场顺序宣布最后得分和最后名次,获得相同分数的选手具有相同的名次,名次连续编号,不用考虑同名次的选手人数。例如:

选手序号: 1, 2, 3, 4, 5, 6, 7
选手得分: 5, 3, 4, 7, 3, 5, 6
则输出名次为: 3, 1, 2, 5, 1, 3, 4

请编程帮助大奖赛组委会完成半决赛的评分排名工作。

* 问题分析与算法设计

问题用程序设计语言加以表达的话,即为:将数组 A 中的整数从小到大进行连续编号,要求不改变数组中元素的顺序,且相同的整数要具有相同的编号。

普通的排序算法均要改变数组元素原来的顺序,显然不能满足要求。为此,引入一个专门存放名次的数组,再采用通常的算法:在尚未排出名次的元素中找出最小值,并对具有相同值的元素进行处理,重复这一过程,直到全部元素排列好为止。

* 程序与程序注释

```
#define NUM 7                                          /* 定义要处理的人数 */
int a[NUM+1]={0,5,3,4,7,3,5,6};                        /* 为简单直接定义选手的分数 */
int m[NUM+1], l[NUM+1];  /* m:已编名次的标记数组 l:记录同名次元素的下标 */
main()
{ int i, n, smallest, num, k, j;
  num=1;                                               /* 名次 */
  for (i=1; i<=NUM; i++)                  /* 控制扫描整个数组,每次处理一个名次 */
    if (m[i]==0) {          /* 若尚未进行名次处理(即找到第一个尚未处理的元素) */
    smallest=a[i];                        /* 取第一个未处理的元素作为当前的最小值 */
    k=1;                                  /* 数组 l 的下标,同名次人数 */
    l[k]=i;                               /* 记录分值为 smallest 的同名次元素的下标 */
     for (j=i+1; j<=NUM; j++)             /* 从下一个元素开始对余下的元素进行处理 */
```

```
        if (m[j]==0)                                    /* 若为尚未进行处理的元素 */
          if (a[j]<smallest) {                          /* 分数小于当前最小值 */
             smallest=a[j];                             /* 则重新设置当前的最小值 */
             k=0;                                       /* 重新设置同名次人数 */
             l[++k]=j;                                  /* 重新记录同名次元素下标 */
          }
          else if (a[j]==smallest)                      /* 若与当前最低分相同 */
                l[++k]=j;                               /* 记录同名次的元素下标 */
      for (j=1; j<=k; j++)                              /* 对同名次的元素进行名次处理 */
          m[l[j]]=num;
      num++;                                            /* 名次加1 */
      i=0;                                              /* 控制重新开始,找下一个没排名次的元素 */
    }
    printf(" Player-No Score Rank\n");
    for (j=1; j<=NUM; j++)                              /* 控制输出 */
      printf(" %3d    %4d    %4d\n", j, a[j], m[j]);
}
```

* 运行结果

Player-No	Score	Rank
1	5	3
2	3	1
3	4	2
5	4	7
5	3	1
3	6	5
7	6	4

* 思考题

若将原题中的“名次连续编号,不用考虑同名次的选手人数”,改为“根据同名次的选手人数对选手名次进行编号”,那么应怎样修改程序。

97. 满足特异条件的数列

输入 m 和 n(20≥m≥n>0)求出满足以下方程式的正整数数列 i1,i2,…,in,使得:i1+i2+...+in=m,且 i1≥i2≥……≥in。例如:

当 n=4, m=8 时,将得到如下 5 个数列:

5 1 1 1　　4 2 1 1　　3 3 1 1　　3 2 2 1　　2 2 2 2

* 问题分析与算法设计

可将原题抽象为:将 M 分解为 N 个整数,且 N 个整数的和为 M,i1≥i2≥……≥in。分解整数的方法很多,由于题目中有“i1≥i2≥……≥in”的要求,提示我们可先确定最右边 in 元素

的值为1,然后按照条件使前一个元素的值一定大于等于当前元素的值,不断地向前递推就可以解决问题。下面的程序允许用户选定M和N,输出满足条件的所有数列。

* 程序与程序注释

```
#define NUM 10                                  /* 允许分解的最大的元素数量 */
int i[NUM];                                     /* 记录分解出的数值的数组 */
main()
{ int sum, n, total, k, flag, count=0;
   printf("Please enter requried terms (<=10):");
   scanf("%d", &n);
   printf("              their sum:");
   scanf("%d", &total);
   sum=0;                                       /* 当前从后向前k个元素的和 */
   k=n;                                         /* 从后向前正在处理的元素下标 */
   i[n]=1;                                      /* 将最后一个元素的值置为1作为初始值 */
   printf("There are following possible series:\n");
   while (1) {
     if ( sum+i[k]<total )                      /* 若后k位的和小于指定的total */
       if (k<=1)                                /* 若正要处理的是第一个元素 */
        { i[1]=total-sum; flag=1; }             /* 则计算第一个元素的值并置标记 */
       else {
         sum+=i[k--];
         i[k]=i[k+1];                           /* 置第k位的值后k-1 */
         continue;                              /* 继续向前处理其它元素 */
       }
     else if (sum+i[k]>total || k!=1)           /* 若和已超过total或不是第一个元素 */
         { sum-=i[++k]; flag=0; }               /* k向后回退一个元素 */
     else flag=1;                               /* sum+i[k]=total && k=1 则设置flag标记 */
     if (flag) {                                /* flag:计算完毕可以输出的标记 */
       printf(" [%d]:", ++count);
       for (flag=1; flag<=n; ++flag)
           printf ("%d", i[flag]);
       printf("\n");
     }
     if (++k>n)                                 /* k向后回退一个元素后判断是否已退出最后一个元素 */
       break;                                   /* 若已退出最后一个元素则退出循环 */
     sum -= i[k];
     i[k]++;                                    /* 试验下一个分解 */
   }
}
```

* 运行结果

```
Please enter requried terms (<=10):4
                    their sum:8
There are following possible series:
[1]:5111
[2]:4211
[3]:3311
[4]:3221
[5]:2222
```

98. 八后问题

在一个8×8的国际象棋盘上，有八个皇后，每个皇后占一格；要求皇后间不会出现相互“攻击”的现象，即不能有两个皇后处在同一行、同一列或同一对角线上。问共有多少种不同的方法。

* 问题分析与算法设计

这是一个古老的具有代表性的问题，用计算机求解时的算法也很多，这里仅介绍一种。

采用一维数组来进行处理。数组的下标 i 表示棋盘上的第 i 列，a[i]的值表示皇后在第 i 列所放的行位置。如：a[1]=5，表示在棋盘的第一列的第五行放一个皇后。

程序中首先假定 a[1]=1，表示第一个皇后放在棋盘的第一列第一行的位置上，然后试探第二列中皇后可能的位置，找到合适的位置后，再处理后续的各列。这样通过对各列的反复试探，可以最终找出皇后的全部摆放方法。

程序采用回溯法，算法的细节请参见程序。

* 程序与程序注释

```
#define NUM 8                                          /* 定义数组大小 */
int a[NUM+1];
main()
{ int number, i, k, flag, not_finish=1, count=0;
  i=1;  /* 正在处理的元素下标，表示前 i-1个元素已符合要求，正在处理第 i 个元素 */
  a[1]=1;                                              /* 为数组的第一个元素赋初值 */
  printf("The possible configuration of 8 queens are:\n");
  while (not_finish) {                                 /* not_finish=1:处理尚未结束 */
    while (not_finish && i<=NUM) {
                                        /* 处理尚未结束且还没处理到第 NUM 个元素 */
      for (flag=1,k=1; flag && k<i; k++)               /* 判断是否有多个皇后在同一行 */
        if (a[k]==a[i]) flag=0;
      for (k=1; flag && k<i; k++)                      /* 判断是否有多个皇后在同一对角线 */
        if ( (a[i]=[k]-(k-i)) || (a[i]==a[k]+(k-i)) ) flag=0;
          if (! flag) {                    /* 若存在矛盾不满足要求，需要重新设置第 i 个元素 */
        if (a[i]==a[i-1]) {                /* 若 a[i]的值已经经过一圈追上 a[i-1]的值 */
```

```
            i--;                                    /* 退回一步,重新试探处理前一个元素 */
            if (i>1 && a[i]==NUM)
                a[i]=1;                          /* 当a[i]的值为NUM时将a[i]的值置1 */
            else if (i==1 && a[i]==NUM)
                not_finish=0;                    /* 当第一位的值达到NUM时结束 */
            else a[i]++;                                /* 将a[i]的值取下一个值 */
          }
          else if (a[i]==NUM) a[i]=1;
          else a[i]++;                                  /* 将a[i]的值取下一个值 */
        }
        else if ( ++i<=NUM )                    /* 第i位已经满足要求则处理i+1位 */
            if (a[i-1]==NUM) a[i]=1;
                                              /* 若前一个元素的值为NUM则a[i]=1 */
            else a[i]=a[i-1]+1;              /* 否则元素值为前一个元素的下一个值 */
      }
      if (not_finish) {
        ++count;
        printf( (count-1)%3 ? " [%2d]:" : "\n [%2d]:", count);
        for (k=1; k<=NUM; k++)                                  /* 输出结果 */
          printf(" %d",a[k]);
        if (a[NUM-1]<NUM) a[NUM-1]++;              /* 修改倒数第二位的值 */
        else a[NUM-1]=1;
        i=NUM-1;                                  /* 开始寻找下一个满足条件的解 */
      }
    }
}
```

* 运行结果

The possible configurations of 8 queens are:

```
[ 1 ]:  15863724  [ 2 ]:  16837425  [ 3 ]:  17468253
[ 4 ]:  17582463  [ 5 ]:  24683175  [ 6 ]:  25713864
[ 7 ]:  25741863  [ 8 ]:  26831475  [ 9 ]:  26174835
[10]:  27368514  [11]:  27581463  [12]:  28613574
[13]:  35714286  [14]:  35841726  [15]:  35281746
[16]:  35286471  [17]:  36814752  [18]:  36815724
[19]:  36824175  [20]:  36258174  [21]:  36271485
[22]:  36275184  [23]:  36418572  [24]:  36428571
[25]:  37285146  [26]:  37286415  [27]:  38471625
[28]:  31758246  [29]:  46827135  [30]:  46831752
```

[31]: 46152837 [32]: 47185263 [33]: 47382516
[34]: 47526138 [35]: 47531682 [36]: 48136275
[37]: 48157263 [38]: 48531726 [39]: 41582736
[40]: 41586372 [41]: 42586137 [42]: 42736815
[43]: 42736851 [44]: 42751863 [45]: 42857136
[46]: 42861357 [47]: 57138642 [48]: 57142863
[49]: 57248136 [50]: 57263148 [51]: 57263184
[52]: 57413862 [53]: 58413627 [54]: 58417263
[55]: 51468273 [56]: 51842736 [57]: 51863724
[58]: 52468317 [59]: 52473861 [60]: 52617483
[61]: 52814736 [62]: 53847162 [63]: 53168247
[64]: 53172864 [65]: 68241753 [66]: 61528374
[67]: 62713584 [68]: 62714853 [69]: 63571428
[70]: 63581427 [71]: 63724815 [72]: 63728514
[73]: 63741825 [74]: 63175824 [75]: 63184275
[76]: 63185247 [77]: 64713528 [78]: 64718253
[79]: 64158273 [80]: 64285713 [81]: 71386425
[82]: 72418536 [83]: 72631485 [84]: 73825164
[85]: 73168524 [86]: 74258136 [87]: 74286135
[88]: 75316824 [89]: 82417536 [90]: 82531746
[91]: 83162574 [92]: 84136275

* 思考题

一个8×8的国际象棋盘，共有64个格子。最多将五个皇后放入棋盘中，就可以控制住整个的盘面，不论对方的棋子放在哪一格中，都会被吃掉。请编程找出这样的五个“皇后”可能的布局。

99. 超长正整数的加法

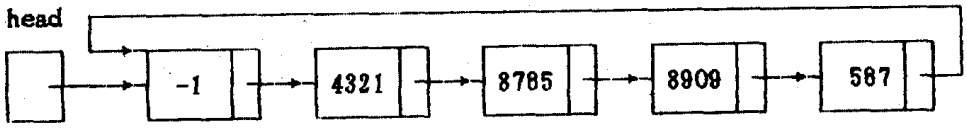
请设计一个算法完成两个超长正整数的加法。

* 问题分析与算法设计

首先要设计一种数据结构来表示一个超长的正整数，然后才能够设计算法。

我们采用一个带有表头结点的环形链来表示一个非负的超大整数。如果从低位开始为每个数字编号，则第一位到第四位、第五位到第八位、…的每四位组成的数字，依次放在链表的第一个、第二个、…结点中，不足4位的最高位存放在链表的最后一个结点中，表头结点的值规定为－1。例如：

大整数“567890987654321”可用如下的带表头结点 head 的链表表示：



按照此数据结构，可以从两个表头结点开始，顺序依次对应相加，求出所需要的进位后代入下面的运算。具体的实现算法请见程序中的注释。

* 程序与程序注释

```
#include <stdio.h>
#define HUNTHOU 10000
typedef struct node { int data;
                      struct node *next;
                    } NODE;                                    /* 定义链表结构 */
NODE *insert_after (u, num)    /* 在u结点之后插入一个新的NODE,其值为num */
  NODE *u;
  int num;
{ NODE *v;
  v = (NODE *)malloc(sizeof(NODE));                            /* 申请一个NODE */
  v->data = num;                                               /* 赋值 */
  u->next = v;                                                 /* 在u结点之后插入一个NODE */
  return(v);
}
NODE *addint(p, q)                  /* 完成加法操作返回指向*p+*q结果的指针 */
  NODE *p, *q;
{ NODE *pp, *qq, *r, *s, *t;
  int total, number, carry;
  pp=p->next; qq=q->next;
  s=(NODE *)malloc(sizeof(NODE));                              /* 建立存放和的链表表头 */
  s->data=-1;
  t=s; carry=0;                                                /* carry:进位 */
  while ( pp->data!=-1 && qq->data!=-1) {                      /* 均不是表头 */
    total = pp->data + qq->data + carry;                       /* 对应位与前次的进位求和 */
    number = total % HUNTHOU;                                  /* 求出存入链中部分的数值 */
    carry = total / HUNTHOU;                                   /* 算出进位 */
    t=insert_after(t,number);                                  /* 将部分和存入s指向的链中 */
    pp=pp->next;                                               /* 分别取后面的加数 */
    qq=qq->next;
  }
  r = (pp->data!=-1) ? pp : qq;                                /* 取尚未处理完毕的链指针 */
  while (r->data!=-1) {                                        /* 处理加数中较大的数 */
    total = r->data+carry;                                     /* 与进位相加 */
    number = total%HUNTHOU;                                    /* 求出存入链中部分的数值 */
    carry = total/HUNTHOU;                                     /* 算出进位 */
```

```
    t = insert_after(t,number);                    /* 将部分和存入 s 指向的链中 */
    r = r->next;                                   /* 取后面的数值 */
  }
  if (carry) t=insert_after(t,1);                  /* 处理最后一次的进位 */
  t->next=s;                                       /* 完成和的链表 */
  return(s);                                       /* 返回指向和的结构指针 */
}

NODE *inputint(void)                               /* 输入超长正整数 */
{ NODE *s, *ps, *qs;
  struct number { int num;                         /* 定义临时的中间结构 */
                  struct number *np;
                } *p, *q;
  int i, j, k;
  long sum;
  char c;
  p=NULL;     /* 指向输入的整数,链首为整数的最低的个位,链尾为整数的最高位 */
  while ( (c=getchar()) != '\n')                   /* 输入整数,按字符接收数字 */
    if (c>='0'&&c<='9') {                          /* 若为数字则存入 */
      q = (struct number *)malloc(sizeof(struct number));   /* 申请空间 */
      q->num=c-'0';                                /* 存入一位整数 */
      q->np=p;                                     /* 建立指针 */
      p=q;
    }
  s=(NODE *)malloc(sizeof(NODE));
  s->data=-1;                                      /* 建立表示超长正整数的链头 */
  ps=s;
  while (p!=NULL) {    /* 将接收的临时数据链中的数据转换为所要求的标准形式 */
      sum=0;i=0; k=1;
      while ( i<4 && p!=NULL) {                    /* 取出低四位 */
        sum = sum + k * (p->num);
        i++; p=p->np; k=k*10;
      }
      qs=(NODE *)malloc(sizeof(NODE));             /* 申请空间 */
      qs->data=sum;                                /* 赋值,建立链表 */
      ps->next=qs;
      ps=qs;
  }
  ps->next=s;
```

```
    return(s);                                    /* 返回指向超长整数结构的指针 */
}
printint(s)
    NODE *s;
{ if (s->next->data!=-1) {                        /* 若不是表头,则输出 */
      printint(s->next);                          /* 递归输出 */
      if (s->next->next->data==-1)
       printf("%d",s->next->data);
       else {
       int i,k=HUNTHOU;
       for (i=1; i<=4; i++,k/=10)
        putchar('0'+s->next->data%(k)/(k/10));
       }
    }
}
main()
{ NODE *s1, *s2, *s;
    NODE *inputint(), *addint(), *insert_after();
    s1=inputint();                                /* 输入被加数 */
    s2=inputint();                                /* 输入加数 */
    printf(" S1="); printint(s1); putchar('\n');  /* 显示被加数 */
    printf(" S2="); printint(s2); putchar('\n');  /* 显示加数 */
    s=addint(s1,s2);                              /* 求和 */
    printf("S1+S2="); printint(s); putchar('\n'); /* 输出结果 */
}
```

* 运行结果

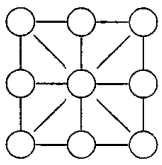
```
 S1= 12345678901234567890123456789012345678901234567890
 S2= 98765432109876543210987654321098765432109876543210
S1+S2=111111111011111111101111111110111111111011111111100
 S1= 99999999999999999999999999999999999999999999999999
 S2=                                                   1
S1+S2=100000000000000000000000000000000000000000000000000
 S1=              6123746732678321786342673746218
 S2=16236723167789326897678936247896789231478 6
S1+S2=162367231684017015709467684265310566061004
 S1=617382678123678416278346123789462137846218736423817642387413264782 3
 S2=                                  3242314321432623176786436782314699 99
S1+S2=617382678123678416278346123789494560989433062655585506755236411782 2
```

* 思考题

请设计完成两个超长正整数的减法、乘法和除法的算法。

100. 数字移动

在图中的九个点上，空出中间的点，其余的点上任意填入数字1至8；1的位置固定不动，然后移动其余的数字，使1到8顺时针从小到大排列。移动的规则是：只能将数字沿线移向空白的点。



请编程显示数字移动过程。

* 问题分析与算法设计

分析题目中的条件，要求利用中间的空白格将数字顺时针方向排列，且排列过程中只能借助空白的点来移动数字。问题的实质就是将矩阵外面的8个格看成一个环，8个数字在环内进行排序。由于受题目要求的限制“只能将数字沿线移向空白的点”，所以要利用中间的空格进行排序，这样要求的排序算法与众不同。

观察中间的点，它是唯一一个与其它8个点有连线的点，即它是中心点。中心点的活动的空间最大，它可以向8个方向移动，充分利用中心点这一特性是算法设计成功与否的关键。

在找到1所在的位置后，其余各个数字的正确位置就是固定的。我们可以按照下列算法从数字2开始，一个一个地来调整各个数字的位置。

- 确定数字 i 应处的位置；
- 从数字 i 应处的位置开始，向后查找数字 i 现在的位置；
- 若数字 i 现在位置不正确，则将数字 i 从现在的位置（沿连线）移向中间的空格，而将原有位置空出；依次将现有空格前的所有元素向后移动；直到将 i 应处的位置空出，把它移入再次空出中间的空格。

从数字2开始使用以上过程，就可以完成全部数字的移动排序。

编程时要将矩阵的外边八个格看成一个环，且环的首元素又是不定的，如果算法设计的不好，程序中就要花很多的精力来处理环中元素的前后顺序问题。将题目中的3×3矩阵用一个一维数组表示，中间的元素（第四号）刚好为空格，设计另一个指针数组，专门记录矩阵外边八个格构成环时的连接关系。指针数组的每个元素依次记录环中数字在原来数组中的对应的元素下标。这样通过指针数组将原来矩阵中复杂的环型关系，表示成了简单的线性关系，从而大大简化了程序设计。

* 程序与程序注释

```
int a[ ]={0,1,2,5,8,7,6,3};        /* 指针数组。依次存入矩阵中构成环的元素的下标 */
int b[9];                                          /* 表示3×3矩阵，b[4]为空格 */
int c[9];                                /* 确定1所在位置后，对环进行调整的指针数组 */
int count=0;                                          /* 数字移动步数计数器 */
main()
{ int i, j, k, t;
   void print();
   printf ("Please enter original order of digits 1~8: ");
   for (i=0; i<8; i++)
      scanf("%d", &b[a[i]]);
```

```
        /* 顺序输入矩阵外边的8个数字,矩阵元素的顺序由指针数组的元素 a[i]控制 */
    printf("The sorting process is as felow:\n");
    print();                                              /* 输出初始矩阵状态 */
    for (t=-1,j=0; j<8 && t==-1; j++)                     /* 确定数字1所在的位置 */
        if ( b[ a[j] ] == 1 ) t=j;                        /* t:记录数字1所在的位置 */
    for (j=0; j<8; j++)       /* 调整环的指针数组,将数字1所在的位置定为环的首 */
        c[j] = a[ (j+t)%8 ];
    for (i=2; i<9; i++)                                   /* 从2开始依次调整数字的位置 */
                          /* i:正在处理的数字,i 对应在环中应当的正确位置就是 i-1 */
        for (j=i-1; j<8; j++)                             /* 从 i 应处的正确位置开始顺序查找 i */
            if (b[c[j]]==i && j!=i-1 ) {                  /* 若 i 不在正确位置 */
                b[4]=i;                                   /* 将 i 移到中心的空格中 */
                b[c[j]]=0; print();                       /* 空出 i 原来所在的位置,输出 */
                for (k=j; k!=i-1; k--) {
                                /* 将空格以前到 i 的正确位置之间的数字依次向后移动一格 */
                    b[ c[k] ] = b[ c[k-1] ];              /* 数字向后移动 */
                    b[ c[k-1] ] = 0;
                    print();
                }
                b[ c[k] ] = i;                            /* 将中间的数字 i 移入正确的位置 */
                b[4] = 0;                                 /* 空出中间的空格 */
                print();
                break;
            }
            else if (b[c[j]]==i) break;                   /* 数字 i 在正确的位置 */
}
void print ( void )                                       /* 按格式要求输出矩阵 */
{ int c;
    for (c=0; c<9; c++)
        if (c%3==2) printf("%2d\n", b[c]);
        else        printf("%2d",b[c]);
    printf("---%2d---\n", count++);
}
```

* 运行结果

```
Please enter original order of digits 1~8: 8 5 6 3 2 1 7 4
The sorting process is as felow:
  8 5 6     8 5 6     8 5 6     8 5 0     8 0 5     0 8 5     4 8 5
  4 0 3     4 2 3     4 2 0     4 2 6     4 2 6     4 2 6     0 2 6
```

7 1 2	7 1 0	7 1 3	7 1 3	7 1 3	7 1 3	7 1 3
— 0—	— 1—	— 2—	— 3—	— 4—	— 5—	— 6—
4 8 5	4 8 5	4 8 5	4 8 5	4 8 0	4 0 8	0 4 8
7 2 6	7 0 6	7 3 6	7 3 0	7 3 5	7 3 5	7 3 5
0 1 3	2 1 3	2 1 0	2 1 6	2 1 6	2 1 6	2 1 6
— 7—	— 8—	— 9—	—10—	—11—	—12—	—13—
7 4 8	7 4 8	7 0 8	0 7 8	4 7 8	4 7 8	4 7 0
0 3 5	3 0 5	3 4 5	3 4 5	3 0 5	3 5 0	3 5 8
2 1 6	2 1 6	2 1 6	2 1 6	2 1 6	2 1 6	2 1 6
—14—	—15—	—16—	—17—	—18—	—19—	—20—
4 0 7	4 5 7	4 5 7	4 5 7	4 5 0	4 5 6	
3 5 8	3 0 8	3 6 8	3 6 0	3 6 7	3 0 7	
2 1 6	2 1 6	2 1 0	2 1 8	2 1 8	2 1 8	
—21—	—22—	—23—	—24—	—25—	—26—	—结束—

* 对问题的进一步讨论

很显然，按照上述算法能解决问题，但移动的步数并不是最少的。

注意算法中的两个问题。其一：数字1的位置自始至终是保持不变的；其二：没有考虑初始情况下，位置原本就已经是正确的数字。如例中的数字5和6，按照算法，当移动其它数字时，5和6也要跟着移动多次，这显然浪费了不少步数。

对于实例，若让数字1参与其它数字的移动排序过程，并充分利用数字5和6初始位置已经正确这一条件，可以大大优化移动排序的过程。例如按以下过程排序，可以减少8步。

8 5 6	8 5 6	8 5 6	8 5 6	8 5 6	8 5 6	8 5 6
4 0 3	4 7 3	4 7 3	4 7 3	4 1 0	4 0 7	4 1 7
7 1 2	0 1 2	1 0 2	1 2 0	1 2 3	1 2 3	0 2 3
— 0—	— 1—	— 2—	— 3—	— 4—	— 5—	— 6—
8 5 6	8 5 6	8 5 6	8 5 6	8 5 6	8 5 6	0 5 6
4 1 7	4 0 7	4 3 7	4 3 7	4 7 1	4 0 1	4 8 7
2 0 3	2 1 3	2 1 0	2 0 3	2 0 1	2 7 1	3 2 1
— 7—	— 8—	— 9—	—10—	—11—	—12—	—13—
4 5 6	4 5 6	4 5 6	4 5 6	4 5 6		
0 8 7	3 8 7	3 8 7	3 8 7	3 0 7		
3 2 1	0 2 1	2 0 1	2 1 0	2 1 8		
—14—	—15—	—16—	—17—	—18—	—结束—	

*思考题

请重新设计算法，编写更优化的程序，尽可能减少移动的步数。