# 算法一

## 数值计算

算法（Algorithm）：计算机解题的基本思想方法和步骤。算法的描述：是对要解决一个问题或要完成一项任务所采取的方法和步骤的描述，包括需要什么数据（输入什么数据、输出什么结果）、采用什么结构、使用什么语句以及如何安排这些语句等。通常使用自然语言、结构化流程图、伪代码等来描述算法。

## 统计数字

此类问题都要使用循环，要注意根据问题确定循环变量的初值、终值或结束条件，更要注意用来表示计数、和、阶乘的变量的初值。

**例 1**：用随机函数产生 100 个[0，99]范围内的随机整数，统计个位上的数字分别为 1，2，3，4，5，6，7，8，9，0 的数的个数并打印出来。

**算法分析：**

本题使用数组来处理，用数组 a[100]存放产生的 100 个随机整数，数组 x[10]来存放个位上的数字分别为 1，2，3，4，5，6，7，8，9，0 的数的个数。即个位是 1 的个数存放在 x[1]中，个位是 2 的个数存放在 x[2]中，……，个位是 0 的个数存放在 x[10]。

程序清单：

```
#include "stdio.h"
main()
{
  int a[101],x[11],i,p;
  for(i=0;i<=11;i++) x[i]=0;   // 数组初如化
  for(i=1;i<=100;i++)    // 产生 100 个随机整数存放数组 a 中
  {
    a[i]=rand() % 100;
    printf("%4d",a[i]);  // 输出每次产生的随机数
    if(i%10==0) printf("\n");  // 每行输出 10 个随机数
  }
  for(i=1;i<=100;i++)
  {
    p=a[i]%10;   // 取此整数的个数数
    if(p==0) p=10; // 个位是 0，单独处理
    x[p]=x[p]+1;   // 统计次数
  }
  for(i=1;i<=10;i++)
  {
    p=i;
```

```
        if(i==10) p=0;
        printf("%d,%d\n",p,x[i]);
      }
    printf("\n");
    getch();
  }
```

# 斐波纳契数列

有一种数列：0，1，12，3，5，8，13，21，34，……它以 0 和 1 开头，接下来每个数是其前面两个数之和。数字家斐波纳契（Fibonacci）首先发现并研究这种数列的性质与应用，该数列因此得名。

斐波纳契数列的迭代形式如下：

$$F_n=\begin{cases}0 & (n=0)\\ 1 & (n=1)\\ F_{n-1}+F_{n-2} & (n\ge 2)\end{cases}$$

自然界本身就存在这样的数列，人们在描述呈螺旋上升形式的数据时，通常要用到它。该数列有一个令人称奇的特性：对于连续的两个数来说，前一个数与后一个数之比趋于常量 0.618，后一个数与前一个数之比趋于常量 1.618。由于 0.618 或者说 1.618（处理方式不同）在自然界的许多方得到体现，符合人类的审美标准，因此被称为“黄金分割率”。

# 兔子繁殖问题

有一个有趣的古典数学问题是这样说的：将刚出生的一对兔子（雌雄各一只）放到一个孤岛上去，这对兔子从第 3 个月起，每个月生一对兔子（雌雄各一只）；同样，每对兔子自出生以后的第 3 个月起，也是每个月生一对兔子（雌雄各一只）。假设所有兔子都不死，问各月份的兔子总数是多少？

初看起来，这个问题并不好解答。但是，通过列举并分析开始几个月的兔子数变化情况，就可以发现兔子繁殖的规律。将不满 1 个月的兔子称为小兔子，满 1 个月但不满 3 个月的兔子称为中兔子，满 3 个月的兔子称为老兔子。那么，兔子繁殖情况可以列表如下：

| 月份 | 小兔子数(对) | 中兔子数(对) | 老兔子数(对) | 兔子总数(对) |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 1 | 0 | 1 | 2 |
| 4 | 1 | 1 | 1 | 3 |
| 5 | 2 | 1 | 2 | 5 |
| 6 | 3 | 2 | 3 | 8 |
| 7 | 5 | 3 | 5 | 13 |
| 8 | 8 | 5 | 8 | 21 |
| …… | …… | …… | …… | …… |

**程序清单：**

```
#include "stdio.h"
#include "conio.h"
main()
{

  long fn1,fn2,fn3;
  int i,n;

  printf("Please enter Month Number(less than 40): ");
  scanf("%d",&n);
  if(n<1) n=1;

  printf("\n");
  fn1=fn2=1;
  printf(" Month. 1:%10ld\n",fn1);
  if(n>1)
    printf(" Month. 2:%10ld\n",fn2);
  for(i=3;i<=n;i++)
  {
    fn3=fn2+fn1;
    printf(" Month.%2d:%10ld\n",i,fn3);
    fn1=fn2;
    fn2=fn3;
  }
  getch();
}
```

## 函数多项式级数

一般中国人记起祖冲之，可能不是因为他组织制定了当时精确度是高的历法，规定一年为 365.24281481 天，与现代天文学家测得的数值相比仅差 50 秒，而是他把圆周率$\pi$推算到小数点以后第 7 位，即 3.1415926～3.1415927 之间，相关成就比 16 世纪中叶德国的渥脱和荷兰的安托尼兹早 1000 多年。站在历史的视角，理应为古代数学家的伟大感到自豪，因为这在当时是了不起的。

然而，如果在今天，利用函数的多项式级数表示方法，别说推算圆周率第 7 位，就是第 70 位、第 700 位、第 7000 位乃至更多，都不在话下。比如说，

$$e^x=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots+\frac{x^n}{n!}+\cdots$$

就是利用多项式级数表示函数的一个例子。

利用多项式级数表示函数是重要的计算方法，有很多函数都可以表示成自变量的多项式，从而比较容

易地计算出许多函数乃至常数的任意精确值。

# 自然常量数 e

自然常数 2.71828182845904520536…用在求自然对数等许多数值计算场合。这个无理数之所以称为 e 常数，是为了纪念天才数学家欧拉（Euler）而用他的姓名首字母 e 命名，同时也称为欧拉常数。

e 常数的计算很简单，其公式如下：

$$e = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!} + \cdots$$

程序清单：

```
#include "stdio.h"
#include "math.h"
#include "conio.h"

main()
{
  int n;
  float delta,t,x,e;

  e=1.0;
  x=1.0;
  n=1;
  t=1.0;
  while(t>1e-6)
  {
   t=t*x/n;
   e=e+t;
   n=n+1;
  }
  printf("E is %f",e);
  getch();
}
```

# 完　数

完数意即“完美的数”。如果一个数恰好是小于它的各个不同因子之和，那么就称该数为完数。比如说，6 的因子 1、2、3，而 6=1+2+3，因此 6 就是一个完数。

显然，要判断一个数是否为完数的关键在于，对它进行适当的因子分解，以得到小于它本身的所有因子。顺乎自然的因子分解方法是遍历试验。也就是说，假设要判断数 n 是否为完数，那么就从 1 开始到数 n-1，逐个看它是否为 n 的因子(能整除 n)。将各个因子累加起来，最后与这个数进行比较，如果相等该数就是完数。

**程序清单：**

```
#include "stdio.h"
#include "conio.h"
int perfect(int n)
{
  int i,sum=0;
  for(i=1;i<n;i++)
    if(n%i==0) sum+=i;
  if(sum==n) return sum;
  return 0;
}

main()
{
  int i,j,n;
  printf(" Perfect numbers between 1 and 1000:\n");
  j=0;
  for(i=2;i<=1000;i++)
  {
    if((n=perfect(i))>0)
    {
      printf("\n%5d=",i);
      for(j=1;j<i;j++)
        if(i%j==0)
         {
           if(j>1) printf(" +");
           printf("%5d",j);
         }
    }
  }
  getch();
}
```

## 级数求和（NOIP 2002 复赛试题 存盘名：NOIPC1.C）

【问题描述】

已知：Sn= 1＋1／2＋1／3＋…＋1／n。显然对于任意一个整数 K，当 n 足够大的时候，Sn 大于 K。现给出一个整数 K（1<=K<=15），要求计算出一个最小的 n；使得 Sn＞K。

【输　　入】键盘输入 k

【输　　出】屏幕输出 n

【输入输出样例】

输入：1

输出：2

【算法分析】

此题是一道基本的数学题。级数 $S_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$，当 n→∞时，Sn→∞。即对给出任一个整数 k，总存在 n，使得 Sn>k。所谓满足条件的最小的 n，指的是 $S_{n-1} \le k$，且 Sn>k。

如：k=2 时，满足条件的最小的 n=4，即 $1 + \frac{1}{2} + \frac{1}{3} \le 2$，$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} > 2$

对于给定的整数 k，如何求得最小的整数 n 呢？可以采用穷举法，初始时 Sn=0，n=0，然后每次循环 n←n+1，Sn←Sn+$\frac{1}{n}$，直到 Sn 大于 k，最后输出 k，程序就完成了。

由于 k 为 1～15 之间的整数，n 的数据类型需定义为 long，将 Sn 定义为 double 类型。

【程序清单】

```
main()
{
  int k;
  double sn=0;
  long n=0;
  printf("Input k:");
  scanf("%d",&k);
  do
  { n++;
    sn+=1.0/n;
  }while(sn<=k);
  printf("%ld",n);
}
```

【测试数据】

| 序号 | 输入 | 输出 |
|---|---|---|
| 1 | 3 | 11 |
| 2 | 7 | 616 |
| 3 | 14 | 675214 |
| 4 | 10 | 12367 |
| 5 | 12 | 91380 |