第八章 常用算法

郎大鹏

第八章 常用算法

8.1　格式控制类

8.2　排序和查找

8.3　一些常见编程算法

8.4　应用题型

8.5　其它常用数学方法

8.6　随机数分析

8.7　习题

第八章 常用算法

8.1 格式控制类

例8.1 输入一个整数行数，根据输入的行数输出杨辉三角。

算法分析：杨辉三角形，又称贾宪三角形，帕斯卡三角形，是二项式系数在三角形中的一种几何排列。杨辉三角最本质的特征是：它的两条斜边都是由数字1组成的，而其余的数则是等于它肩上的两个数之和，根据上述分析，可设计算法如下。

（1）定义一个二维数组a[N]用于存储数据，N为宏定义；

（2）输入一个整数，用于记录需要计算的行数；

（3）将数组第一列的所有元素置1，即a[i][0]=1,i为数组行下标；

（4）从第二行第二列开始，每个数字等于它肩上两个数字之和；

（5）控制打印格式，打印输出

源程序：

```
#include  <stdio.h>
#define N 15  /*宏定义，用来表示二维数组下标范围*/
void main()
{  int i,j,n=0,a[N][N]={0}; /*定义变量和二维数组，并将数组进行初始化*/
   printf("please enter lines (1<N<=15):"); /*输入杨辉三角形的行数*/
      scanf("%d",&n);
   if (n<1 || n>15)
     printf("error!");/*如果输入非法，进行提示，程序结束*/
   else
   { for(i=0;i<n;i++) /*注意循环变量范围*/
        a[i][0]=1;   /*利用一重循环将二维数组第一列置1 */
      for(i=1;i<n;i++)
        for(j=1;j<=i;j++) /*循环变量j随i进行变化*/
           a[i][j]=a[i-1][j-1]+a[i-1][j];
/*每个数字等于它肩上两个数字之和*/
      for(i=0;i<n;i++)  /*利用二重循环输出结果*/
      {for(j=0;j<=i;j++)
         printf("%5d",a[i][j]);
        printf("\n");
      }
}
}
```

8.1　格式控制类

例8.2　输出九九乘法表。

算法分析：该程序主要考核学生如何控制屏幕输出格式，使用格式控制符实现。可设计算法如下。

定义一个9行9列二维数组用于存储数据；

利用二重循环输出结果，在输出效果方面，使用c语言的格式控制符控制输出效果。

源程序:

```
#include <stdio.h>
void main()
{
int a[9][9],i,j;                    /*定义存储数据的数组*/
for(i=0;i<9;i++)                    /*计算数组个元素的值*/
  for(j=0;j<9;j++)
    a[i][j]=(i+1)*(j+1);
    for(i=0;i<9;i++)                    /*输出*/
    {
        for(j=0;j<=i;j++)
            printf("%d*%d=%2d ",i+1,j+1,a[i][j]);
        printf("\n");
    }
}
```

第八章 常用算法

8.1 格式控制类

8.2 排序和查找

8.3 一些常见编程算法

8.4 应用题型

8.5 其它常用数学方法

8.6 随机数分析

8.7 习题

8.2 排序和查找

8.2.1 排序

所谓排序，就是使一串记录或者数字，按照其中的某个或某些关键字的大小，递增或递减的排列起来的操作。排序的算法有很多，对空间的要求及其时间效率也不尽相同。下面列出了一些常见的排序算法：插入排序、冒泡排序、选择排序、快速排序、堆排序、归并排序、基数排序、希尔排序。插入排序和冒泡排序又被称作简单排序，它们对空间的要求不高，但是时间效率却不稳定，而后面三种排序相对于简单排序对空间的要求稍高一点，但时间效率却能稳定在很高的水平。

8.2.1　排序

例8.3　输入10个数字，使用冒泡排序将其由小到大排序并输出排序后的数字。

算法分析：冒泡排序的平均时间复杂度是平方级的，但是非常容易实现，算法思路比较简单，具体算法如下。

将所有待排序的数字放入工作列表中，此时采用数组进行存储；

输入待排序的数字；

从数组的第一个数字开始检查，如果大于它的下一位数字，则进行互换；

如此循环直至倒数第二个数字；

输出数组元素。

源程序：

```
#include <stdio.h>
#define  N 10  /*宏定义*/
void main()
{      int a[N] , i, j, t;
       printf("input %d numbers :\n",N ); /*提示输入数字*/
       for (i=0 ; i<N ; i++)
          scanf("%d",&a[i]); /*输入数字并存入数组中*/
       printf("\n");
       for(j=0 ; j<N-1 ; j++) /*对数组中的数字从第一个循环到倒数第二个*/
          for(i=0 ; i<N-j ; i++) /*逐个比较 */
            if (a[i]>a[i+1])  /*交换数字*/
            {
                t = a[i];
                a[i] = a[i+1];
                a[i+1] = t;
             }
/*通过上述循环后，每个数字均进行了比较*/
       printf("the sorted numbers:\n");
       for(i=0;i<N; i++)   /*输出*/
           printf("%d  ", a[i]);
       printf("\n");
}
```

8.2.2 查找

查找的方法取决于查找表的结构。静态查找算法常见有以下两种：

（1）顺序查找：按查找表的自然顺序依次查找。

（2）折半查找：前提必须是有序表，即数据按升序或降序存储。查找时用关键字与中间位置元素比较。两值相等，查找成功，否则仅需在左侧区域或右侧区域进行查找，可以大大加快查找速度。

8.2.2 查找

例8.4 输入10个数，然后输入一个关键数，使用顺序查找法找到该数字并输出其位置。

算法分析：顺序查找可以从数组的第一个元素开始，直到找到待查找元素的位置。具体算法如下。

- 输入一组数字并存入数组中；
- 输入一个关键字，用于查找；
- 使用一重循环，从数组第一个元素开始直至最后一个元素对关键字进行比对；
- 如果比对成功，输出成功提示，输出元素的位置，程序结束；
- 如果比对不成功，输出失败提示，程序结束；

源程序：

```
#include "stdio.h"
#define N 10  /*宏定义*/
void main()
{  int a[N],i,key;
   printf("input %d numbers :\n",N );
   for (i=0 ; i<N ; i++)
      scanf("%d",&a[i]); /*输入一组数字*/
   printf("\n");
   printf("input key number :",key );
   scanf("%d",&key); /*输入一个关键字*/
   for(i=0;i<N;i++) /*使用一重循环进行顺序查找*/
      if (a[i]==key)
      {  printf("\n success! coordinates is %d\n",i);
         break; /*如果查找成功，输出成功提示和元素位置*/
      }
   if(i==N)
      printf("failed!\n");/*如果查找不成功，输出失败提示*/
}
```

例8.5　给定10个有序数字和一个关键数，使用折半查找法找到该数并输出其位置。

算法分析：折半查找是将待查找的数组元素不断的分为两部分，每次淘汰二分之一，但是有个大前提是，元素必须是有序的，如果是无序的则要先进行排序操作。具体算法如下：

- 定义一个一维数组并初始化一组有序数字；
- 定义一个关键字待查找；
- 定义两个变量low和high，分别表示查找区域的第一个元素和最后一个元素的下标，初始化时的待查找区域是整个数组；
- 定义变量mid，表示待查找区域的中间元素下标，mid=(low+high)/2；
- 定义变量flag表示查找是否成功；
- 如果下标为mid的元素等于关键字key，查找成功，flag置1，程序结束；
- 如果下标为mid的元素大于关键字key，因为数组是有序的，所以可以确定key位于下标为low的元素和下标为mid-1的元素之间，此时重新置high的值为mid-1；
- 如果下标为mid的元素小于关键字key，操作与步骤（7）相反；
- 如此循环步骤（7）和（8），如果high〉low，说明整个数组中没有找到与key相同的数字，查找失败，flag置0；
- 根据flag的值进行输出。

8.2.2 查找

源程序：

```
#include <stdio.h>
#define N 10
void main()
{   int i, a[]={2,10,20,28,45,64,69,80,100,2010},key=80;
    int low=0,high=N-1,mid,flag=0; /*初始化*/
    while (low<=high) /*开始循环*/
    {mid=(low+high)/2;
      if (a[mid]==key)
      {flag=1;
          break; /*如果成功，退出*/
      }
      if (a[mid]>key)
         high=mid-1;
      else
         low=mid+1;  /*根据a[mid]与key的比较进行折半*/
    }
 printf("number:");
    for (i=0 ; i<N ; i++)
       printf("%d ",a[i]);
    printf("\nkey is:%d\n",key);
    if (flag==0)   /*根据flag的值进行不同结果输出
*/
       printf("failed!");
    else
       printf("sucess! R=%d\n",mid);
}
```

8.3 一些常见编程算法

例**8.6** 编程读取一段英文文本，输出其中的每个单词并统计其中有多少个单词，单词之间使用空格分隔。

算法分析：这是一个单词分割的例子，具体算法如下。

- 定义一个字符数组用于存储英文文本；
- 定义一个变量**num**作为计数器，用于存储单词数。定义一个变量**flag**用于判断是否出现单词；
- 从第一个字符开始顺序读取数组中的字母，直到出现字符串结束标志**'\0'**，如果出现空格，说明出现了一个单词，将**flag**置为**0**；
- 如果**flag**为**0**，表示刚刚出现了一个单词，此时将计数器加**1**，将**flag**置为**1**以开始下一个判断；
- 输出结果**num**。

源程序：

```
#include <stdio.h>
#include <string.h>
void main()
{  char read[100];/*定义一个数组，用于存储英文文本*/
   int i,num,flag;
   char c;
   num = 0;
   flag = 0;  /*初始化变量*/
   gets(read); /*读入一段英文*/
   puts(read);
   for(i=0; (c=read[i]) != '\0'; i++) /*逐个字母进行判断*/
   {  if (c ==' ')
         flag=0;   /*读入空格时候表示出现了新的单词*/
      else if (flag == 0）
         {  flag=1;
                num++;
         }
   }
   printf("There are %d words .\n",num);
}
```

第八章 常用算法

8.3 一些常见编程算法

例8.7 输入一个十进制数字和欲转换的进制（10以下），输出转换后的结果。

算法分析：这是一个进制转换问题，计算机中常用的数的进制主要有：二进制、八进制、十六进制。算法如下。

(1)定义变量x，存储一个十进制数字；

(2)定义变量y，表示需要转化的进制；

(3)令t=x/y；

(4)如果t不为零，说明此时t大于等于y，转到（3）；

(5)如果t等于零，说明此时t小于y，可以直接将t向y进制转化。

(6)递归调用(3)(4)(5)；

源程序：

```
#include <stdio.h>
void convert(int x,int y)
{
   int t;
   if ((t = x/y)!= 0)  /*判断是否可以直接转化，这是递归调用的出口*/
   {
      convert(t,y); /*如果不能跳出，则继续递归调用*/
   }
   printf("%d", x%y); /*每次递归打印余数，直至最后递归结束*/
}
void main()
{
   int x,y;
   printf("input  numbers:",x );    /*输入*/
   scanf("%d", &x);
   printf("convert:",y );
   scanf("%d", &y);
   convert(x,y);   /*调用*/
}
```

8.3 一些常见编程算法

例8.8 将一个二维数组的行和列元素互换，存到另一个二维数组中。

算法分析：这是一个矩阵转置问题。问题关键就是a数组元素的行下标和列下标分别是b数组的列下标和行下标。具体算法如下。

定义一个二维数组a并进行数据初始化；

定义一个与a中数组行数、列数互换的数组b，用于存储转置后的数组数据；

根据第一个数组的行列建立一个二重循环，将a中的每一个元素复制到b中，复制原则是a数组元素的行下标和列下标分别是b数组的列下标和行下标；

输出数组b。

源程序：

```
#include <stdio.h>
void main()
{  int a[2][3]={{1,2,3},{4,5,6}};/*定义数组a并初始化*/
   int b[3][2], i, j;   /*根据a的行列下标定义数组b的列行下标*/
   printf("array a:\n");
   for (i=0; i<=1; i++)
   {for (j=0; j<=2; j++)
      {     printf("%5d",a[i][j]);
            b[j][i] = a[i][j];   /*行列互换*/
            }
            printf("\n");
            }
            printf("array b:\n");
            for (i=0; i<=2; i++) /*输出结果*/
            {
      for (j=0; j<=1; j++)
                        printf("%5d",b[i][j]);
            printf("\n");
            }
}
```

8.3 一些常见编程算法

例8.9 输入两个整数，求它们之间的素数并输出。

算法分析：首先需要明确素数的定义，如果一个整数N，能被2到根号下N之间的整数整除，那么N不是素数，反之N是素数，1不是素数，2和3是素数。据此，可依如下思路编写程序：

（1）定义两个变量n1和n2用于存储输入的两个整数，且n1不能大于n2。

（2）定义整数n，使n从n1循环到n2。

（3）对于每一个n，取其整数平方根，从2到其整数平方根进行循环，判定其是否能够被n整除，如果能够整除说明n不是素数，反之n是素数，打印之。

源程序：

```
#include <stdio.h>
#include <math.h> /*引入头文件*/
void main()
{
   int n1,n2;
   int n,sqrt_n,flag,i=0;
   printf("please input n1:");
   scanf("%d",&n1);
   printf("please input n2:");
   scanf("%d",&n2);  /*输入两个数字*/
   if(n1==1)
      n1++; /*如果输入n1为1，则加1*/
   if(n1>=n2)
      printf("input error!"); /*如果n1小于n2，程序报错，此处也可修改程序为将n1和n2互换*/
   else
   {
for(n=n1;n<n2;n++)
      {  /*逐个判断*/
         for(sqrt_n=2,flag=1;sqrt_n <= sqrt(n);sqrt_n++)
/*该for循环功能是判断n是否为素数*/
         {
            if(n%sqrt_n==0)
            {
               flag=0;
               break;
            }
         }
         if(flag)
         { /*如果是素数，打印，注意此打印是在最外层循环的里面*/
            printf("%d ",n);
            i++;
            if(i%8==0)
               printf("\n");
         }
      }
   }
}
```

8.3 一些常见编程算法

例8.10 输入两个数，求它们的最大公约数和最小公倍数

算法分析：可以利用辗转相除法求最大公约数，关于辗转相除法，在我国古代的《九章算术》中就有如下记载：约分术曰：“可半者半之，不可半者，副置分母、子之数，以少减多，更相减损，求其等也。以等数约之。”其中所说的“等数”，就是最大公约数。求“等数”的办法是“更相减损”法，实际上就是辗转相除法。最小公倍数 = 两个数的积 / 最大公约数。具体算法如下。

- 输入两个数m和n;
- m对n求余为a，若a不等于0，则 m = n，n = a，继续求余。否则 n 为最大公约数。
- 最小公倍数 = 两个数的积 / 最大公约数;
- 输出结果。

源程序：

```
#include <stdio.h>
void main()
{
   int m, n;
   int m_cup, n_cup, res;
   printf("Enter two integer:\n");
   scanf("%d %d", &m, &n);  /*输入*/
   if (m > 0 && n >0) /* 判断输入合法性*/
   {  m_cup = m;
      n_cup = n;
      res = m_cup % n_cup; /*初始化*/
      while (res != 0) /*开始辗转相除，直至能够整除*/
      {  m_cup = n_cup;
         n_cup = res;
         res = m_cup % n_cup;
      }
      printf("Greatest common divisor: %d\n", n_cup); /*打印结果*/
      printf("Lease common multiple : %d\n", m*n/n_cup);
   }
   else
      printf("Error!\n");/*若输入错误，打印错误信息后退出*/
}
```

8.3 一些常见编程算法

例8.11 输入两个数，求它们之间的水仙花数并打印。

算法分析：水仙花数是指一个n位数（n>=3）的各个位的n次方的和等于这个数，那这个数就是水仙花数，因此，首先需要将这个数的各个位分解开来，然后分别进行n次方后求和，与原数比较。具体算法如下描述。

（1）定义两个整型变量n1，n2；

（2）定义一个整型变量n，使得n进行n1到n2的循环；

（3）对每一个n，将其各位进行分解并存入定义好的数组中，同时判断该数的位数并存入整型变量x中；

（4）将数组中的各个元素分别进行x次方后求和，如果和等于n，则n是水仙花数；

（5）打印之。

源程序：

```
#include <stdio.h>
#include <math.h> /*引入头文件*/
void main()
{  int n1, n2;
   int n,sum,a[10],i,x;
   double temp_sum;  /*求n次方函数的运行结果是浮点型*/
   printf("Enter two integer:\n");
   scanf("%d,%d", &n1, &n2); /*输入*/
   for(sum=n1;sum<=n2;sum++)  /*对n1和n2之间的数进行逐个判断*/
   {  n=sum;         /*每次循环开始将sum赋给n，用于分解各个位*/
      temp_sum=0;    /*每次循环开始将temp_sum复位*/
      x=0;                                        /*用来记录该数一共有多少位*/
      while(n%10>0)   /*该循环用来进行各个位的分解，存入数组，并记录位数*/
      {       a[x]=n%10;
               x++;
               n=n/10;
      }   for(i=x-1;i>=0;i--)   /*用上面记录的位数来控制循环，求x次方后之和*/
          temp_sum = temp_sum + pow(a[i],x);/*pow函数的运行结果是浮点型*/
      if(sum == int(temp_sum)) /*强制转换后进行比较*/
           printf("%d ",sum);
   }
}
```

8.3 一些常见编程算法

例8.12 求1000以内的完数。

算法分析：所谓完数就是因子之和等于该数本身，例如6=1+2+3。据此，判断一个数是否为完数应先将其进行因式分解，然后求因子之和进行比较。具体算法描述如下。

（1）定义整型变量sum，用于存储因子之和；

（2）定义整型变量n，使n从1循环至1000；

（3）对于每一个n进行循环，从1至n-1进行逐个判断是否能够被n整除，如果能够整除，则说明该数是n的因子，执行sum=sum+i语句；

（4）如果因子之和等于n，则输出n。

（5）重复（2）。

源程序：

```
#include <stdio.h>
void main()
{
  int i,n,sum;
  for(n=2;n<=1000;n++)  /*进行循环*/
  {
     sum=0;
     for(i=1;i<n;i++) /*此循环用来找出所有的因子*/
     {
        if(n % i == 0)  /*如果是因子，加入到因子之和sum中*/
        {
           sum=sum+i;
        }
     }
     if(sum == n)   /*判断是否是完数*/
        printf("%6d is a Perfect num\n",n);
  }
}
```

8.4 应用题型

例8.13 猴子吃桃，猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个。第2天早上将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第n天早上想再吃时，见只剩下一个桃子了。求第1天共摘了多少桃子？

算法分析：某一天吃的是前一天的一半还多一个，假设今天剩下为x1，昨天共有x2个桃子，它们的关系是：x1=x2/2-1,即x2=(x1+1)*2，如果知道今天剩下的桃子，那么就可以知道昨天的，如果知道昨天的，那么前天的就知道了……。如果在第n天只剩下一个桃子，那么向前计算n-1天就可以计算出第一天的桃子。具体算法描述如下。

（1）定义一个变量day，表示一共吃的天数；

（2）定义两个变量x1和x2，表示前一天吃的桃子数和今天吃的桃子数；

（3）初始化；

（4）利用算法分析中的公式进行计算，每次计算后应将x1和x2进行重新初始化，每计算一次将天数减1，直至天数为零；

（5）输出结果.

源程序：

```
#include <stdio.h>
void main()
{
  int day,x1,x2;      /*定义变量*/
  printf("\n Enter day:");
  scanf("%d",&day);  /*输入天数*/
  day--;
  x2=1;                              /*初始化*/
  while(day>0)                       /*循环，直至天数为零*/
  {
    x1=(x2+1)*2;
    x2=x1;                           /*根据公式进行计算，每次计算后将x1和x2重新初始化*/
    day--;
  }
  printf(" The number is %d\n",x1);  /*输出结果*/
}
```

8.4 应用题型

例8.14 猴子分桃，海滩上有一堆桃子，五只猴子来分。第一只猴子把这堆桃子凭据分为五份，多了一个，这只猴子把多的一个扔入海中，拿走了一份。第二只猴子把剩下的桃子又平均分成五份，又多了一个，它同样把多的一个扔入海中，拿走了一份，第三、第四、第五只猴子都是这样做的，问海滩上原来最少有多少个桃子？

算法分析：首先可以肯定原来桃子的总数不会超过某一个数值，不妨假设该数值为10000。从10000开始，逐个向下检查，总可以找到原来桃子的总数。本题目其它算法公式和算法描述可参考例8.13，源程序也没有给出注释，请同学们进行分析。

源程序：

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
   int t,i,y,temp;
   for(t=10000;t>0;t--)
   {
      temp=t;
      for(i=1;i<=5;i++)
      {
        if((temp-1)%5!=0)
           break;
        else
        temp=temp-(temp-1)/5-1;
      }
      if(i==6)
          printf(" %d\n",t);
   }
}
```

8.4 应用题型

例8.15 百鸡问题，用100元买100只鸡，大公鸡5元1只，母鸡3元1只，小鸡1元3只。问各能买多少只？

算法分析：可以使用穷举法搜索法来完成，穷举搜索法是对可能是解的众多候选解按某种顺序进行逐一枚举和检验，并从众找出那些符合要求的候选解作为问题的解。把公鸡、母鸡、小鸡只数分别设为未知数cock、hen、chick，显然，三个未知数满足如下：cock：1~20。Hen：1~33。chick：1~100，且它们符合以下条件：cock+hen+chick=100，同时cock*5+hen*3+chick/3=100。定义三个循环变量，做三重循环，穷举所有可能性。具体算法描述如下。

- 定义三个变量，cock、hen、chick;
- 考察三个变量的范围，cock：1~20。Hen：1~33。chick：1~100;
- 根据三个变量的范围构造一个三重循环，在循环的最内部进行条件判断：cock+hen+chick=100 且 cock*5+hen*3+chick/3=100;
- 得到结果，输出。

源程序：

```
#include <stdio.h>
void main()
{
   int cock,hen,chick,count=0; /*定义变量*/
   for(cock=0;cock<=20;cock++)/*构造三重循环*/
      for(hen=0;hen<=33;hen++)
         for(chick=0;chick<=100;chick++)
          if((cock+hen+chick)==100 &&(cock*5+hen*3+chick/3)==100)/*判断*/
             {
              printf("\n cock:%-3d hen:%-3d chick:%-3d",cock,hen,chick);
                 count++;/*记录共有几种方法*/
              }
    printf("\n total of %d method!\n",count); /*输出结果*/
}
```

8.5 其它常用数学方法

迭代法也称为辗转法或递推法，是一种不断用变量的旧值递推新值的过程，主要作为一种求方程或方程组近似根的常用算法设计方法。利用迭代算法解决问题，需要做好以下三个方面的工作：

（1）确定迭代变量

在可以用迭代算法解决的问题中，至少存在一个直接或间接地不断由旧值递推出新值的变量，这个变量就是迭代变量。

（2）建立迭代关系式

所谓迭代关系式，指如何从变量的前一个值推出其下一个值的公式（或关系）。迭代关系式的建立是解决迭代问题的关键，通常可以使用递推或倒推的方法来完成。

（3）对迭代过程进行控制

8.5.1　迭代法

例8.16　输入一个整数，用迭代法求其平方根。

算法分析：求平方根的迭代公式是：x1=1/2*(x0+a/x0)。具体算法描述如下。

（1）先自定一个初值x0，作为a的平方根值，在我们的程序中取a/2作为a的初值；利用迭代公式求出一个x1。此时，x1与a的真正的平方根值相比，误差很大。

（2）把新求得的x1代入x0中，准备用此新的x0再去求出一个新的x1。

（3）利用迭代公式再求出一个新的x1的值，也就是用新的x0又求出一个新的平方根值x1，此值将更趋近于真正的平方根值。

（4）比较前后两次求得的平方根值x0和x1，如果它们的差值小于指定的值，即达到要求的精度，则认为x1就是a的平方根值，去执行步骤（5）；否则执行步骤（2），即循环进行迭代。

源程序：

```
#include<stdio.h>
#include<math.h>
void main()
{
    double a,x0,x1;
    printf("Input a:\n");
    scanf("%lf",&a);                    /*输入*/
    if(a<0)
       printf("Error!\n");              /*判断*/
    else
    {
       x0=a/2;
       x1=(x0+a/x0)/2;                  /*初始化*/
       do
       {
          x0=x1;
          x1=(x0+a/x0)/2;               /*循环利用迭代公式*/
        }while(fabs(x0-x1)>=1e-6);      /*直至达到指定精度*/
    }
    printf("Result:\n");
    printf("sqrt(%g)=%g\n",a,x1);       /*输出结果*/
}
```

8.5.2 递归法

程序调用自身的编程技巧称为递归。它通常把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。

一般来说，递归需要有边界条件、递归前进段和递归返回段。当边界条件不满足时，递归前进；当边界条件满足时，递归返回。

注意：

（1）递归就是在过程或函数里调用自身；

（2）在使用递增归策略时，必须有一个明确的递归结束条件，称为递归出口。

例8.17　用递归法求两个数的最大公约数。

源程序：

```
#include <stdio.h>
int d( int m, int n) /*递归函数*/
{
   if(m%n==0)                    /*递归出口判断*/
      return n;
   else
      return d(n, m%n);  /*循环调用自身，注意函数参数*/
}
void main()
{
   int m,n;
   printf(" input m,n:");
   scanf("%d,%d",&m,&n);
   printf(" max divisor: %d",d(m,n));
}
```

8.5.3 分治法

分治法的基本思想是：任何一个可以用计算机求解的问题所需的计算时间都与其规模N有关。问题的规模越小，越容易直接求解，解题所需的计算时间也越少。例如，对于n个元素的排序问题，当n=1时，不需任何计算；n=2时，只要作一次比较即可排好序；n=3时只要作3次比较即可，…。而当n较大时，问题就不那么容易处理了。要想直接解决一个规模较大的问题，有时是相当困难的。分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。

8.5.3 分治法

分治法的适用条件：分治法所能解决的问题一般具有以下几个特征：

（1）该问题的规模缩小到一定的程度就可以容易地解决；

（2）该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质；

（3）利用该问题分解出的子问题的解可以合并为该问题的解；

（4）该问题所分解出的各个子问题相互独立，即子问题之间不包含公共的子子问题。

分治法的基本步骤，分治法在每一层递归上都有三个步骤：

（1）分解：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题；

（2）解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题；

（3）合并：将各个子问题的解合并为原问题的解。

8.6 随机数分析

在C语言中,rand（）函数可以用来产生随机数，但是这不是真正意义上的随机数，而是一个伪随机数。它是根据一个数（可以种子）为基准以某个递推公式推算出来的一系数，当这系列数很大的时候就符合正态公布，从而相当于产生了随机数，但当计算机正常开机后，这个种子的值是定了的，因此产生的随机数是伪随机数。为了改变这个种子的值，C提供了srand（）函数，函数原型是void srand(int a)，该函数可以用来提供随机种子。为了产生不可预见的随机序列，通常利用时间函数来帮助产生随机种子，考虑到每一次运行程序的时间是不同的，因此srand((unsigned int)(time(NULL))是一种常用的形式。

8.6　随机数分析

例8. 18　打印10个随机数。

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
                /*使用当前时钟做种子*/
void main(  )
{
      int i;
      srand( (unsigned)time( NULL ) );
            /*初始化随机数*/
      printf("ten rand numbers:\n");
         for( i = 0; i < 10;i++ )
              /*打印出10个随机数*/
      {
      printf( " %6d\n", rand() );
      if (i%5==0)
         printf("\n");
      }
}
```

8.6 随机数分析

例8.19 打印10个0~100之间的随机数。

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
void main( )
{
int i;
srand( (unsigned)time( NULL ) );
printf("ten rand numbers(0~100):\n");
    for( i = 0; i < 10;i++ )
{
   printf( "%3d\n", rand()%100+1);
   if (i%5==0)
       printf("\n");
}
}
```