

第 5 章

利用数组处理批量数据

5.1 用筛选法求 100 之内的素数。

解：解题思路。所谓“筛法”指的是“埃拉托色尼(Eratosthenes)筛法”。埃拉托色尼是古希腊的著名数学家。他采取的方法是,在一张纸上写上 1~1000 的全部整数,然后逐个判断它们是否是素数,找出一个非素数,就把它挖掉,最后剩下的就是素数,见图 5-1。

① 2 3 ④ 5 ⑥ 7 ⑧ ⑨ ⑩ 11 ⑫ 13 ⑭ ⑮ ⑯ 17 ⑰ 19 ⑱ ⑲ 21 ⑳ 23 ㉔ ㉕ ㉖ ㉗ ㉘ 29 ㉚ 31 ㉜ ㉝ ㉞ ㉟ ㊱
37 ㊳ ㊴ ㊵ 41 ㊷ 43 ㊹ ㊺ ㊻ 47 ㊽ ㊾ ㊿ ...

图 5-1

具体做法如下。

(1) 先将 1 挖掉(因为 1 不是素数)。

(2) 用 2 去除它后面的各个数,把能被 2 整除的数挖掉,即把 2 的倍数挖掉。

(3) 用 3 去除它后面各数,把 3 的倍数挖掉。

(4) 分别用 4,5...各数作为除数去除这些数以后的各数。这个过程一直进行到在除数后面的数已全被挖掉为止。例如在图 5-1 中找 1~50 的素数,要一直进行到除数为 47 为止。事实上,可以简化,如果需要找 1~n 的素数表,只须进行到除数为 $\sqrt{n}$ (取其整数)即可。例如对 1~50,只须进行到将 7(即 $\sqrt{50}$ 的整数部分)作为除数即可。请读者思考为什么?

上面的算法可表示如下。

(1) 挖去 1。

(2) 用下一个未被挖去的数 p 去除 p 后面各数,把 p 的倍数挖掉。

(3) 检查 p 是否小于 $\sqrt{n}$ 的整数部分(如果 $n=1000$,则检查 $p<31?$),如果是,则返回(2)继续执行,否则就结束。

(4) 剩下的数就是素数。

用计算机解此题,可以定义一个数组 a。数组元素 $a[1] \sim a[n]$ 分别代表 1~n 这 n 个数。如果检查出数组 a 的某一元素的值是非素数,就使它变为 0,最后剩下不为 0 的就是素数。

编写程序如下。

```

#include<stdio.h>
#include<math.h>                                //程序中用到求平方根函数 sqrt
int main()
{ int i,j,n,a[101];                             //定义 a 数组包含 101 个元素
  for (i=1;i<=100;i++)                          //a[0]不用,只用 a[1]~a[100]
    a[i]=i;                                       //使 a[1]~a[100] 的值为 1~100
  a[1]=0;                                         //先"挖掉"a[1]
  for (i=2;i<sqrt(100);i++)
    for (j=i+1;j<=100;j++)
      { if(a[i]!=0 && a[j]!=0)
        { if (a[j]%a[i]==0)
          a[j]=0;                                //把非素数"挖掉"
        }
      }
  printf("\n");
  for (i=2,n=0;i<=100;i++)
    { if (a[i]!=0)                               //选出值不为 0 的数组元素,即素数
      { printf("%5d",a[i]);                      //输出素数,宽度为 5 列
        n++;                                     //累计本行已输出的数据个数
      }
    }
  if(n==10)
    { printf("\n");
      n=0;
    }
  }
  printf("\n");
  return 0;
}

```

运行结果:

```

 2   3   5   7  11  13  17  19  23  29
31  37  41  43  47  53  59  61  67  71
73   7  83  89  97

```

5.2 用选择法对 10 个整数排序。

解: 解题思路。选择法的思路如下: 设有 10 个元素 $a[1] \sim a[10]$, 将 $a[1]$ 与 $a[2] \sim a[10]$ 比较, 若 $a[1]$ 比 $a[2] \sim a[10]$ 都小, 则不进行交换, 即无任何操作。若 $a[2] \sim a[10]$ 中有一个以上比 $a[1]$ 小, 则将其中最大的一个 (假设为 $a[i]$) 与 $a[1]$ 交换, 此时 $a[1]$ 中存放了 10 个数中最小的数。第 2 轮将 $a[2]$ 与 $a[3] \sim a[10]$ 比较, 将剩下 9 个数中的最小者 $a[i]$ 与 $a[2]$ 对换, 此时 $a[2]$ 中存放的是 10 个中第二小的数。以此类推, 共进行 9 轮比较, $a[1] \sim a[10]$ 就已按由小到大的顺序存放了。N-S 图如图 5-2 所示。

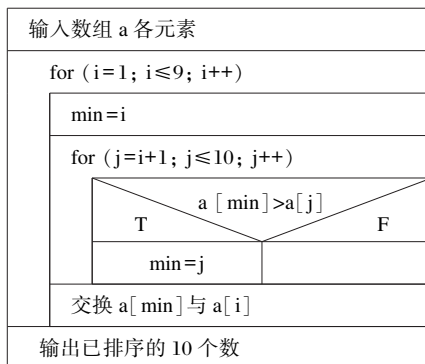


图 5-2

编写程序如下。

```
#include<stdio.h>
int main()
{ int i,j,min,temp,a[11];
  printf("enter data: \n");
  for (i=1;i<=10;i++)
  {printf("a[%d]=",i);
    scanf("%d",&a[i]);          //输入 10 个数
  }
  printf("\n");
  printf("The original numbers: \n");
  for (i=1;i<=10;i++)
    printf("%5d",a[i]);          //输出这 10 个数
  printf("\n");
  for (i=1;i<=9;i++)             //以下 8 行是对 10 个数排序
  {min=i;
    for (j=i+1;j<=10;j++)
      if (a[min]>a[j]) min=j;
    temp=a[i];                   //以下 3 行将 a[i+1]~a[10]中最小者与 a[i]对换
    a[i]=a[min];
    a[min]=temp;
  }
  printf("\nThe sorted numbers: \n");    //输出已排好序的 10 个数
  for (i=1;i<=10;i++)
    printf("%5d",a[i]);
  printf("\n");
  return 0;
}
```

运行结果:

enter data:

a[1]=1
a[2]=16
a[3]=5
a[4]=98
a[5]=23
a[6]=119
a[7]=18
a[8]=75
a[9]=65
a[10]=81

The original numbers:

1 16 5 98 23 119 18 75 65 81

The sorted numbers:

1 5 16 18 23 65 75 81 98 119

5.3 求一个 3×3 的整型二维数组对角线元素之和。

解：编写程序如下。

```
#include<stdio.h>
int main()
{ int a[3][3],sum=0;
  int i,j;
  printf("enter data:\n");
  for (i=0;i<3;i++)
    for (j=0;j<3;j++)
      scanf("%d",&a[i][j]);
  for (i=0;i<3;i++)
    sum=sum+a[i][i];
  printf("sum=%6d\n",sum);
  return 0;
}
```

运行结果：

enter data:

1 ✓

2 ✓

3 ✓

4 ✓

5 ✓

6 ✓

7 ✓

8 ✓

9 ✓

sum= 15

关于输入数据方式的讨论：

在程序的 `scanf` 语句中用 `%d` 作为输入格式控制,上面输入数据的方式显然是可行的。其实也可以在一行中连续输入 9 个数据,如:

1 2 3 4 5 6 7 8 9 ✓

结果也一样。在输入 9 个数据并按回车键后,这 9 个数据被送到内存中的输入缓冲区中,然后逐个送到各个数组元素中。下面的输入方式也是正确的:

1 2 3 ✓

4 5 6 ✓

7 8 9 ✓

或者:

```

1 2 ✓
3 4 5 6 ✓
7 8 9 ✓

```

都是可以的。

请考虑,如果将程序第 7~9 行改为

```

for (j=0;j<3;j++)
    scanf("%d %d %d",&a[0][j],&a[1][j],&a[2][j]);

```

应如何输入? 是否必须一行输入 3 个数据,如:

```

1 2 3 ✓
4 5 6 ✓
7 8 9 ✓

```

答案是可以按此方式输入,也可以不按此方式输入,而采用前面介绍的方式输入,不论分多少行、每行包括几个数据,只要求最后输入 9 个数据即可。

程序中用的是整型数组,运行结果是正确的。如果用的是实型数组,只须将程序第 4 行的 `int` 改为 `float` 或 `double` 即可,并且在 `scanf` 函数中使用 `%f` 或 `%lf` 格式声明。

5.4 已有一个已排好序的数组,要求输入一个数后,按原来排序的规律将它插入数组中。

解: 解题思路。设数组 `a` 有 `n` 个元素,而且已按升序排列,在插入一个数时按下面的方法处理:

(1) 如果插入的数 `num` 比 `a` 数组最后一个数大,则将插入的数放在 `a` 数组末尾。

(2) 如果插入的数 `num` 不比 `a` 数组最后一个数大,则将它依次和 `a[0]~a[n-1]` 比较,直到出现 `a[i]>num` 为止,这时表示 `a[0]~a[i-1]` 各元素的值比 `num` 小,`a[i]~a[n-1]` 各元素的值比 `num` 大。`num` 理应插到 `a[i-1]` 之后、`a[i]` 之前。怎样才能实现此目的呢? 将 `a[i]~a[n-1]` 各元素向后移一个位置(即 `a[i]` 变成 `a[i+1]`, $\dots$,`a[n-1]` 变成 `a[n]`)。然后将 `num` 放在 `a[i]` 中。N-S 图见图 5-3。

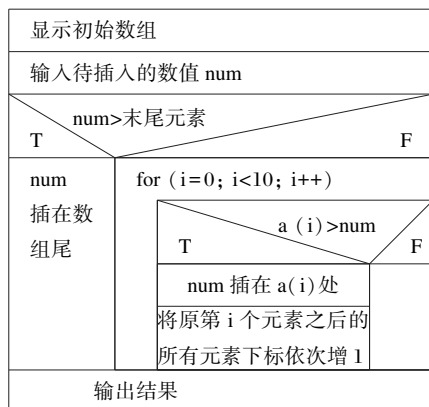


图 5-3

编写程序如下。

```
#include<stdio.h>
int main()
{
    int a[11]={1,4,6,9,13,16,19,28,40,100};
    int temp1,temp2,number,end,i,j;
    printf("array a: \n");
    for (i=0;i<10;i++)
        printf("%5d",a[i]);
    printf(" \n");
    printf("insert data:");
    scanf("%d",&number);
    end=a[9];
    if (number>end)
        a[10]=number;
    else
    {
        for (i=0;i<10;i++)
        {
            if (a[i]>number)
            {
                temp1=a[i];
                a[i]=number;
                for (j=i+1;j<11;j++)
                {
                    temp2=a[j];
                    a[j]=temp1;
                    temp1=temp2;
                }
                break;
            }
        }
    }
    printf("Now, array a: \n");
    for (i=0;i<11;i++)
        printf("%5d",a[i]);
    printf(" \n");
    return 0;
}
```

运行结果:

```
array a:
    1    4    6    9   13   16   19   28   40  100
insert data: 5
Now, array a:
    1    4    5    6    9   13   16   19   28   40  100
```

5.5 将一个数组中的值按逆序重新存放。例如,原来顺序为 8,6,5,4,1,要求改为 1,4,5,6,8。

解: 解题思路。以中间的元素为中心,将其两侧对称的元素的值互换。例如,将 8 和 1 互换,将 6 和 4 互换。N-S 图见图 5-4。

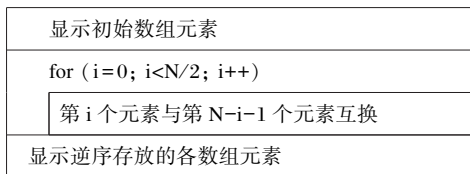


图 5-4

编写程序如下。

```
#include<stdio.h>
#define N 5                //定义 N 代表 5
int main()
{ int a[N],i,temp;
  printf("enter array a: \n");
  for (i=0;i<N;i++)
    scanf("%d",&a[i]);
  printf("array a: \n");
  for (i=0;i<N;i++)
    printf("%4d",a[i]);
  for (i=0;i<N/2;i++)      //循环的作用是将对称的元素的值互换
  { temp=a[i];
    a[i]=a[N-i-1];
    a[N-i-1]=temp;
  }
  printf(" \nNow, array a: \n");
  for (i=0;i<N;i++)
    printf("%4d",a[i]);
  printf(" \n");
  return 0;
}
```

运行结果:

```
enter array a:
8 6 5 4 1 ✓
array a:
8    6    5    4    1
Now, array a:
1    4    5    6    8
```

5.6 输出以下的杨辉三角形(要求输出10行)。

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
:

```

解: 解题思路。杨辉三角形是 $(a+b)^n$ 展开后各项的系数。例如:

$(a+b)^0$ 展开后为 1	系数为 1
$(a+b)^1$ 展开后为 $a+b$	系数为 1, 1
$(a+b)^2$ 展开后为 $a^2+2ab+b^2$	系数为 1, 2, 1
$(a+b)^3$ 展开后为 $a^3+3a^2b+3ab^2+b^3$	系数为 1, 3, 3, 1
$(a+b)^4$ 展开后为 $a^4+4a^3b+6a^2b^2+4ab^3+b^4$	系数为 1, 4, 6, 4, 1

以上就是杨辉三角形的前5行。杨辉三角形各行的系数有以下的规律:

- (1) 各行第一个数都是1。
- (2) 各行最后一个数都是1。

(3) 从第3行起,除上面指出的第一个数和最后一个数外,其余各数是上一行同列和前一列两个数之和。例如,第4行第2个数(3)是第3行第2个数(2)和第3行第1个数(1)之和。可以这样表示: $a[i][j]=a[i-1][j]+a[i-1][j-1]$,其中, i 为行数, j 为列数。

编写程序如下。

```

#include<stdio.h>
#define N 10
int main()
{
    int i,j,a[N][N];                //数组为10行10列
    for (i=0;i<N;i++)
        {
            a[i][i]=1;                //使对角线元素的值为1
            a[i][0]=1;                //使第1列元素的值为1
        }
    for (i=2;i<N;i++)                //从第3行开始处理
        for (j=1;j<=i-1;j++)
            a[i][j]=a[i-1][j-1]+a[i-1][j];
    for (i=0;i<N;i++)
        {
            for (j=0;j<=i;j++)
                printf("%6d",a[i][j]);    //输出数组各元素的值
            printf("\n");
        }
    printf("\n");
    return 0;
}

```

 **说明:** 数组元素的序号是从 0 开始算的, 因此数组中 0 行 0 列的元素实际上就是杨辉三角形中第 1 行第 1 列的数据, 以此类推。

运行结果:

```

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

```

5.7 输出“魔方阵”。所谓魔方阵是指这样的方阵, 它的每一行、每一列和对角线之和均相等。例如, 三阶魔方阵为

```

8  1  6
3  5  7
4  9  2

```

要求输出由 $1 \sim n^2$ 的自然数构成的魔方阵。

解: 解题思路。魔方阵的阶数 n 应为奇数。要将 $1 \sim n^2$ 的自然数构成魔方阵, 可按以下规律处理:

- (1) 将 1 放在第 1 行中间一列。
- (2) 从 2 开始直到 $n \times n$, 各数依次按下列规则存放: 每一个数存放的行比前一个数的行数减 1, 列数加 1 (例如上面的三阶魔方阵, 5 在 4 的上一行后一列)。
- (3) 如果上一数的行数为 1, 则下一个数的行数为 n (指最下一行)。例如, 1 在第 1 行, 则 2 应放在最下一行, 列数同样加 1。
- (4) 当上一个数的列数为 n 时, 下一个数的列数应为 1, 行数减 1。例如, 2 在第 3 行最后一列, 则 3 应放在第 2 行第 1 列。
- (5) 如果按上面规则确定的位置上已有数, 或上一个数是第 1 行第 n 列时, 则把下一个数放在上一个数的下面。例如, 按上面的规定, 4 应该放在第 1 行第 2 列, 但该位置已被 1 占据, 所以 4 就放在 3 的下面。由于 6 是第 1 行第 3 列 (即最后一列), 故 7 放在 6 下面。

按此方法可以得到任何阶的魔方阵。

N-S 图见图 5-5。

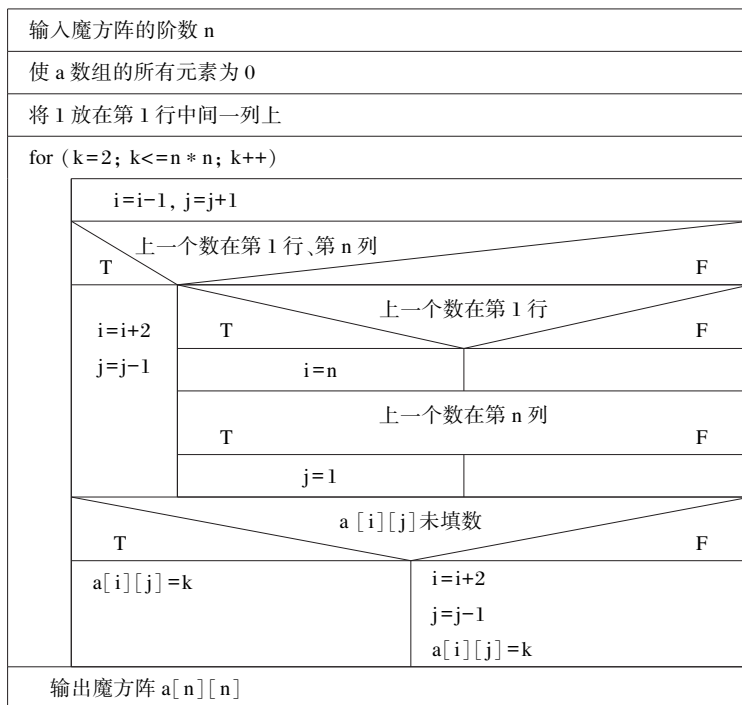


图 5-5

编写程序如下。

```
#include<stdio.h>
int main()
{ int a[15][15], i, j, k, p, n;
  p=1;
  while (p==1)
  { printf("enter n(n=1 to 15):");          //要求阶数为 1~15 的奇数
    scanf("%d", &n);
    if ((n!=0) && (n<=15) && (n%2!=0))      //检查 n 是否为 1~15 的奇数
      p=0;
  }
  //初始化
  for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
      a[i][j]=0;
  //建立魔方阵
  j=n/2+1;
  a[1][j]=1;
  for (k=2; k<=n*n; k++)
  { i=i-1;
    j=j+1;
    if ((i<1) && (j>n))
      { i=i+2;
```