第 4 章　数组与指针

通过本章的学习，理解数组和指针的概念、功能；掌握数组和指针的定义和引用格式；理解数组和指针做函数参数时，参数值的传递过程；掌握用指针表示数组元素的多种方法。了解变量的引用的概念和引用做函数参数的功能。

- 数组和指针的概念、功能；
- 用指针表示数组元素的多种方法；
- 数组和指针做函数参数时，参数值的传递过程。

本章主要介绍数组、指针的概念及它们的简单应用；数组的定义、引用及做函数参数的功能；指针的含义，指针变量的定义、指针运算；指针做函数参数和返回值的功能；用指针表示数组元素的多种方法；字符串的概念、表示方法及其处理函数；指针数组与多级指针；变量引用的概念及引用做函数参数和返回值的功能。

4.1　数组及其应用

当程序处理的数据个数较少时，用少数几个变量来描述它们就够了。但在有的应用中，需要程序处理大批量的数据，数据之间甚至有某种逻辑上的联系，这时可以采用数组。数组适合表示大批量的同类型数据。数组作函数参数时，形参数组与实参数组对应同一存储空间，被调函数对形参数组的处理就是对实参数组的处理，因此被调函数可以改变实参数组，也能返回多个值。

4.1.1　数组的概念

某些程序在运行过程中要处理大批量的数据，而且这批数据中数据之间有某种逻辑上的联系，从而构成逻辑上的整体。如成绩统计分析程序处理一个班 60 名学生的数学成绩时，这些成绩构成一批数据，解方程组程序求解一个多元一次方程组时，方程组所有系数也构成一批数据。设计程序时，为批量数据中的每个数据定义一个变量显然非常麻烦，也难以反映出数据之间的逻辑联系和实现对数据有规律的运算。此时可将这种批量数据作为一个整体处理，用数组来描述和存储。

具有相同类型的一批数据所构成的整体称为数组。一个变量用来表示一个数据，一个数组则用来表示一批数据，因此数组也可理解为一批变量。数组的名字简称为数组名。数组中的

数据称为数组元素（或数组分量），用数组名和下标（顺序号）来标识；可将一个数组元素看成一个变量。例如，一个班 60 名学生的数学成绩可以用一个数组 g 来表示，各个学生的成绩分别表示为：

g[0]，g[1]，g[2]，…，g[i]，…，g[59]

又如二元一次方程组：

$$\begin{cases} a_{0,0}x_0+a_{0,1}x_1=b_0 \\ a_{1,0}x_0+a_{1,1}x_1=b_1 \end{cases}$$

未知数 x 的系数可以用数组 a 表示，其元素为：

a[0][0]，a[0][1]

a[1][0]，a[1][1]

在这里，区分 g 数组的元素需要一个顺序号，故称为一维数组；而区分 a 数组的元素需要两个顺序号，故称为二维数组。

引入数组的概念后，我们可以用循环语句控制下标的变化，从而实现对数组元素有规律的访问。例如，输入 60 名学生的成绩，可写为：

```
for(i=0;i<60;i++)
cin>>g[i];
```

此处利用一条简短的语句，就可输入各个数据。一旦各个数据项存于数组中，将能随时引用其中任一数据，而不必重新输入该数据。

4.1.2　一维数组

如同简单变量一样，数组在使用之前也要定义，即确定数组的名字、类型、大小和维数。

1．一维数组的定义

一维数组的定义形式为：

```
类型符　数组名[常量表达式];
```

其中，方括号中的常量表达式的值表示数组元素的个数，即数组的大小或长度。常量表达式可以包括字面常量和符号常量以及由它们组成的常量表达式，但必须是整型。方括号之前的数组名是一个标识符。类型符指定数组（数组元素）的类型。

例如数组定义

```
int a[10];
```

表示定义了一个数组名为 a 的一维数组，它有 10 个元素，每个元素都是整型。

要注意的是方括号中不能含有变量，下面的定义错误的：

```
int N;
cin>>N;                    //输入数组的长度
int a[N];                  //企图根据 N 的临时输入值定义数组的长度
```

如果 N 是已经定义的符号常量则合法，例如：

```
#define N 10               //或者  const int N=10;
int a[N];
```

2．一维数组元素的引用

一维数组元素的引用形式为：

```
数组名[下标]
```

一个数组元素的引用就代表一个数据，它和简单变量等同使用。

C++规定，数组元素的下标从 0 开始。在引用数组元素时要注意下标的取值范围。当

所定义数组的元素个数为 N 时，下标值取 0 到 N-1 之间的整数。例如上面定义的数组 a 有 10 个元素，即 a[0]，a[1]，…a[9]，如程序引用数组元素 a[i]，就要保证程序运行时 i 在 0 到 9 之间。

下标可以是整型常量、整型变量或整型表达式。要给上面的数组 a 中数组元素输入数据可表示如下（假设 i 是已定义的整型变量）：

```
for(i=0;i<10;i++)
cin>>a[i];              //输出则改为：cout<<a[i];
```

其功能相当于语句：

```
cin>>a[0]>>a[1]>>a[2]>>a[3]>>a[4]>>a[5]>>a[6]>>a[7]>>a[8]>>a[9];
```

3．一维数组的存储结构

程序运行期间，每个变量对应一个特定的存储单元，一维数组在内存中则占据一片连续的存储单元，数组元素按下标从小到大连续排列，每个元素占用相同的字节数。

例如，定义数组 a 如下：

```
int a[5];
```

则数组 a 的存储结构如图 4-1 所示。数组名 a 代表数组所占内存的起始地址，即第一个元素的地址（该元素首字节的编号）。图中数字 1000 表示内存地址，可认为 a 代表地址常量 1000。

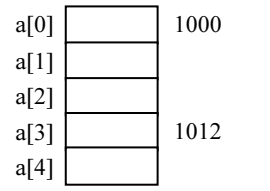

图 4-1　一维数组存储结构

4．一维数组的初始化

如同变量初始化一样，在定义数组的同时，可设定数组全部或部分元素的初值。数组的初始化可用以下几种方法实现。

（1）顺序列出数组全部元素的初值。

数组初始化时，将数组元素的初值依次写在一对花括号内。例如：

```
int x1[5]={0,1,2,3,4};
```

经上面定义和初始化之后，使得 x1[0]、x1[1]、x1[2]、x1[3]、x1[4]的初值分别为 0、1、2、3 和 4。

注意：如提供的初值个数超过了数组元素个数，编译源程序时会出现语法错误。

（2）仅对数组的前面一部分元素设定初值。

```
int x2[10] ={0,1,2,3};
```

对 x2 前四个元素设定了初值，依次为 0，1，2，3。编译系统默认后六个元素值为 0。

（3）对全部数组元素设定初值时，可以不指定数组元素的个数。

```
int x3[]={0,1,2,3,4,5,6,7,8,9};
```

编译系统根据花括号中数据的个数确定数组的元素个数。所以数组 x3 有 10 个元素。但若提供的初值个数小于数组应有的元素个数时，则方括号中的数组元素个数值不能省略。

5．一维数组简单应用举例

【例 4.1】已知数列

$$\begin{cases} f_1 = 1,\ f_2 = 0,\ f_3 = 1 \\ f_n = f_{n-1} - 2f_{n-2} + f_{n-3} \qquad (n > 3) \end{cases}$$

编程实现：

（1）按每行 5 个元素的格式输出 f_1～f_{20}；

（2）输出 f_1～f_{20} 中最大、最小元素及它们的下标。

分析：数列即有次序的一系列数，在程序中正好可以将它的各项存于数组 f 的元素中，如

将数列的 f_1 存于 f[1]中，f_i 存于 f[i]中，先根据上述数列的递推公式求出 f_i 存于 f[i]中，然后按格式要求输出 f 数组的各元素。

至于求 f_1～f_{20} 中最大、最小元素，就是找出 f[1]～f[20]中最大、最小元素。用变量 max、min 分别保存当前找到的最大、最小值，首先假定第一个元素既是最大的，也是最小的，即用 f[1]对 max、min 赋值。然后用 max、min 与（后面）所有元素（f[i]）一一比较，把比当前 max 更大的元素值赋给 max，比当前 min 更小的元素值赋给 min，同时用变量 j、k 分别记录当前最大、最小元素的下标。这种有规律的比较过程正好可用 for 语句控制，比较完后，输出 max、min、j、k 即可。

程序如下：

```
//*****ex4_1.cpp*****
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{   int i, f[21]={0,1,0,1};                    //f[0]闲置不用，可随意设定其初值
    int j,k,max,min;
    for(i=4;i<21;i++)                          //求各个元素的值
        f[i]=f[i-1]-2*f[i-2]+f[i-3];
    cout<<"The   results:"<<endl;
    for(i=1;i<21;i++)
        {   if(i%5==1)   cout<<endl;           //输出 5 个元素换行
            cout<<setw(6)<<f[i];
        }
    cout<<endl;
    //下面解答（2）问，求最大、最小元素
    max=min=f[1];                              //假定第一个元素既是最大的，也是最小的
    j=k=1;                                     //将最初最大，最小元素下标值赋给变量 j，k
    for (i=2;i<21;i++)
    {   if (max<f[i]) { max=f[i];j=i;}         //把当前找到的最大值送 max，下标送 j
        if (min>f[i]) { min=f[i];k=i;}         //把当前找到的最小值送 min，下标送 k
    }
    cout<<"\nmax:f["<<j<<"]="<<max<<", min:f["<<k<<"]="<<min<<'\n';
    return 0;
}
```

其实也可不用 max、min 变量，而仅找最大元素、最小元素的下标，将上面程序中的 max<f[i] 改成 f[j]<f[i]，将 min>f[i]改成 f[k]>f[i]，最后输出最大值 f[j]、最小值 f[k]。

思考：若 f_1 存于 f[0]中，f_i 存于 f[i-1]中，上述程序如何修改。

【例 4.2】将从键盘输入的 N 个数按从小到大顺序排列，然后输出。

分析：

解答此题分为三个步骤：

第一步：将需要排序的 N 个数存放到一个数组中（设 x 数组）；

第二步：将 x 数组中的元素从小到大排序，即 x[0]最小、x[1]次之、…、x[N-1]最大。

第三步：将排序后的 x 数组输出。

其中第二步是关键，实现这一步的排序算法非常多，如选择法、冒泡法、插入法、归并法等。此处采用选择排序法。

选择排序法的基本思路是：设有 N 个数要排序，则从中选择最小的数与第一个数交换，然后从后面的 N-1 个数中找最小的数，将它与第二个数互换，这样重复 N-1 次后即可将 N 个数按由小到大排序。

具体过程是：要找出 x[0]、x[1]、…、x[j]、…、x[N-1]中最小值元素，则找出最小元素的下标即可。用变量 k 保存当前已找到最小元素的下标，先假定最前面元素是最小者，因此将 0 赋给 k（k=0;），然后用 x[k]与 x[0]后面的元素逐个比较，若发现某个 x[j]小于 x[k]，则将 k 修改为 j（k=j;），继续用新的 x[k]与 x[j]后面的元素逐个比较，最后最小元素下标便存于 k；这种逐个比较的过程可用循环控制。然后交换 x[0]、x[k]的值，这样第一轮选择交换就将 N 个元素中的最小值放入 x[0]中。一般地，第 i+1 轮选择交换将找出 x[i]、x[i+1]、…、x[j]、…、x[N-1]中最小元素并交换到 x[i]中。依此类推，经过 N-1 轮处理后就完成了将 N 个数由小到大排序。i 从 0 递增到 N-2 也用循环控制。该过程的流程图如图 4-2 所示。

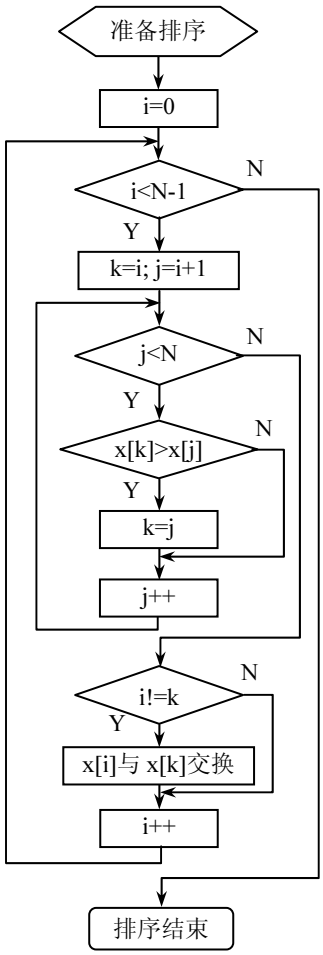

图 4-2　选择排序算法流程图

程序如下：

```
//*****ex4_2.cpp*****
#define N   10
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{   int i,j,t,k;
    int x[N];
    cout<<"please input 10 numbers:"<<endl;
    for (i=0;i<N;i++)
        cin>>x[i];                          //从键盘上输入 10 个元素，x[0]保存第一个数据
    for (i=0;i<N-1;i++)
    {   k=i; //每轮选择开始时假定 x[i]最小，将下标 i 保存在 k 中
      for (j=i+1;j<N;j++)                   //x[k]与 x[i]后面的元素逐个比较
            if (x[k]>x[j]) k=j;             //若某个 x[j]小于 x[k]，则将 k 修改为 j
            if (k!=i)                       //若 k 已不等于其初值，
            { t=x[i]; x[i]=x[k]; x[k]=t; }  //说明 k 至少被修改过 1 次，找到了更小的数，
                                            //则交换 x[i]、x[k]的值
    }
    cout<<"The sorted numbers:"<<'\n';
    for(i=0;i<N;i++)
    cout<<setw(6)<<x[i];
```

```
    cout<<endl;
    return  0;
}
Please input 10 numbers：11  5  6  2  8  7  9  1  150  90↙
The sorted numbers：1  2  5  6  7  8  9  11  90  150
```

思考：要实现由大到小排序的功能该如何修改？

4.1.3　二维数组

数组的维数是指数组元素的下标个数，一维数组的元素只有一个下标，二维数组的元素则有两个下标。或者说二维数组中的元素需要两个下标才能区分。

1．二维数组的定义

二维数组的定义形式为：

类型符　数组名[常量表达式 1][常量表达式 2];

例如

```
int a[3][4];
```

定义二维数组 a，其 12 个元素逻辑上构成如下 3 行 4 列的二维阵列：

```
a[0][0]  a[0][1]  a[0][2]  a[0][3]
a[1][0]  a[1][1]  a[1][2]  a[1][3]
a[2][0]  a[2][1]  a[2][2]  a[2][3]
```

C++把二维数组看作是一种特殊的一维数组，而它的元素本身又是一维数组。C++认为上述数组 a 是特殊一维数组，其 3 个元素 a[0]、a[1]和 a[2]逻辑上分别对应于上面二维阵列的第一、第二、第三行，每个元素又是一个包含 4 个元素的一维数组，如 a[0]包含上面二维阵列的第一行 4 个元素 a[0][0]、a[0][1]、a[0][2]、a[0][3]。通常，一个 n 维数组可看作是一个一维数组，而它的元素是一个 n-1 维的数组。

由二维数组可以推广到多维数组。多维数组的定义形式有连续多个“[常量表达式]”。例如：

```
float b[2][2][3];
```

定义了三维数组 b。

2．二维数组元素在内存中的存储顺序

程序运行时，二维数组占据一片连续的内存单元。将二维数组看成二维阵列时，在内存中二维数组元素是按行的顺序存放的，即先顺序存放第一行的各元素，再存放第二行的各元素，依次类推。对于上面二维数组 a，其元素在内存中的排列顺序如图 4-3 所示。

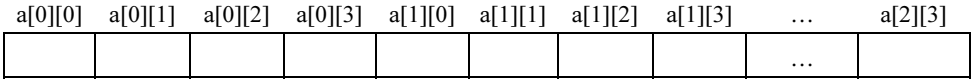

图 4-3　二维数组的存储结构

对于一个多维数组，它的元素在内存中的存放顺序为：第一维的下标变化最慢，最右边的下标变化最快。

3．二维数组的初始化

二维数组的初始化方法有以下几种：

（1）按行给二维数组各元素设定初值。

例如：

```
int y1[2][3]={{1,2,3},{4,5,6}};
```

第一对花括号{1,2,3}内的数据给第一行的元素设定初值，第二对花括号内的数据给第二行的元素设定初值，依次类推。这种设定初值方法比较直观。

（2）按元素的排列顺序初始化

例如：

```
int y2[2][3]={1,2,3,4,5,6};
```

其效果与上一方式相同，但这种设定初值方法结构性差，容易遗漏。

（3）对部分元素设定初值

例如：

```
int y3[3][4]={{1,2},{0,5},{4}};
```

y3 各元素依次为

```
1    2    0    0
0    5    0    0
4    0    0    0
```

显然，其效果是使 y3[0][0]为 1，y3[0][1]为 2，y3[1][0]为 0，y3[1][1]为 5，y3[2][0]为 4，其余元素均为 0。也可对部分行的元素设定初值，例如：

```
int y3[3][4]={{1,2},{0,5}};
```

y3 各元素依次为

```
1    2    0    0
0    5    0    0
0    0    0    0
```

中间的行不设定初值则对应的一对括号不能省，例如：

```
int y3[3][4]={{1,2},{ },{4}};
```

y3 各元素依次为

```
1    2    0    0
0    0    0    0
4    0    0    0
```

（4）如果对数组的全部元素都赋初值，定义数组时，第一维的大小可以不指定，例如：

```
int y4[][3]={1,2,3,4,5,6};
```

编译系统会根据给出的初始数据个数和其他维的元素个数确定第一维的大小。所以数组 y4 有 2 行。

分行设定元素初值时，也可省略第一维的大小，例如：

```
int y5[][3]={{0,2},{ }};
```

编译系统也能确定数组 y5 共有 2 行。

4. 二维数组元素的引用

二维数组元素的引用形式为：

```
数组名[下标 1][下标 2]
```

下标 1 称第一维下标，下标 2 称第二维下标。因二维数组逻辑上对应一张表格，故下标 1 又称行标，下标 2 又称列标。如同一维数组一样，下标可以是整型常量、变量或表达式。各维下标的下界都是 0。

n 维数组元素的引用形式为数组名之后紧接连续 n 个“[下标]”。

5. 二维数组应用举例

二维数组比较适合用来描述逻辑上构成二维表格的一批数据，如矩阵、线性方程组的系数、组成元素具有相同数据类型的各种表格。

【例 4.3】找出矩阵 $A_{M\times N}$ 中第一个最大元素以及它的行号和列号（从 0 开始计算）。

分析：可以将 $A_{M\times N}$ 存于二维数组 a 中，然后在 a 中找第一个最大元素以及它的行号和列号，此题与例 4.1 类似，不同之处是查找范围从一维数组变成二维数组。此处用变量 max 保存数组中的最大元素值，用变量 row、colum 分别记录其行下标和列下标。首先假定第 0 行（最前面一行）的第 0 列元素是最大的，即执行 max= a[0][0]; row=0,colum=0;。然后用 max 逐行逐列与 a 的元素比较，把比 max 更大元素的值赋给 max，并用变量 row、colum 分别记录其行下标和列下标；这种有规律的比较过程正好可用双重循环控制，控制行标 i 和列标 j 变化即可。比较完后，输出 max、row、colum 就得到结果。

程序如下：

```
//*****ex4_3.cpp*****
#define M 3
#define N 4
#include <iostream>
using namespace std;
int main()
{   int i,j,row,colum,max;
    int a[M][N];
    for(i=0;i<M;i++)          //输入 a 数组
        for(j=0;j<N;j++)
            cin>>a[i][j];
    max=a[0][0];              //假定第一行第一列元素是最大的
    row=0,colum=0;            //对记录最大元素行、列下标的变量 row、colum 赋初值
    for (i=0;i<M;i++)
        for(j=0;j<N;j++)
            if (max<a[i][j])
            {   max=a[i][j];
                row=i;
                colum=j;} //把当前最大值送 max，其行列下标送变量 row、colum
    cout<<"\nmax＝"<<max<<"，row="<< row<<"，colum="<<colum<<'\n';
    return 0;
}
```

【例 4.4】范得蒙（Vandermonde）矩阵是一种特殊方矩阵，其最后一列全为 1，倒数第二列为一个指定的列向量，其他各列每行上元素是其后列同行元素与倒数第二列同行元素的乘积。如下 A 为范得蒙矩阵：

$$A=\begin{bmatrix} a^3 & a^2 & a & 1 \\ b^3 & b^2 & b & 1 \\ c^3 & c^2 & c & 1 \\ d^3 & d^2 & d & 1 \end{bmatrix}$$

现要求编程生成 $A_{N\times N}$ 范得蒙矩阵，其倒数第二列从键盘输入。

分析：显然 $A_{N\times N}$ 范得蒙矩阵可用一个二维数组 x[N][N]存储。先输入 N 个数据存于倒数第二列各行元素，然后从后向前求各列每行上元素的值。

程序如下：

```
//*****ex4_4.cpp*****
#define N 4
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{   int x[N][N],i,j;                //定义数组
    for(i=0;i<N;i++)                //最后一列各行元素都为 1
        x[i][N-1]=1;
    for(i=0;i<N;i++)                //输入倒数第二列各行元素
        cin>>x[i][N-2];
    for(j=N-3;j>=0;j--)             //从后向前求各列每行上元素的值
        for(i=0;i<N;i++)
            x[i][j]=x[i][j+1]*x[i][N-2];
    for(i=0;i<N;i++)                //输出 x 数组中的矩阵
    { for(j=0;j<N;j++)
          cout<<setw(6)<<x[i][j];
          cout<<'\n';
    }
    return 0;
}
```

4.1.4 数组作为函数的参数

常量、变量和表达式可以作为函数的实参，数组元素相当于单个变量，因此数组元素和含有数组元素的表达式同样可以作为函数的实参。数组元素作函数的实参，与普通变量作实参一样，函数调用时，系统将数组元素实参的值传给形参，形参的变化不会影响实参数组元素，即参数值是单向传递的。

前面章节介绍的函数只对少数几个参数进行处理，当函数要对一批数据进行处理时，定义函数时可以采用数组作为函数的形参。如前所述，数组名代表数组存储区域的首地址，因此，调用函数时实参应该用数组名或地址值。数组名作实参，函数调用时计算机系统会把实参数组的起始地址传给形参数组名，从而使形参数组与实参数组对应同一存储区域，函数中语句对形参数组（元素）的修改就是对实参数组的修改。地址是一种特别数据值，后面指针部分会详细介绍。

实参数组与形参数组的数据类型要相同，维数要相同。一维数组作函数形参时，其元素个数不必指定为某常数，一般用另一整型形参表示数组中元素的个数。二维数组作形参时，其行数不必指定，一般用另一整型形参表示行数，但必须指定列数为某常数。

【例 4.5】编写函数 void sort(int x[],int k)对形参数组 x 的前 k 个元素中的数据按由小到大次序排序，要求用冒泡排序法。

冒泡法的基本思想是：依次比较相邻两数，若前面数大，则交换两数位置，直至最后 2 个数被处理，此时，最大的数就“沉”到最下面，即在最后一个数组元素位置上，此时第一轮

冒泡处理完毕。如有 k 个数，第二轮冒泡只对前面 k-1 个数（当前未排好序的数）处理。如图 4-4 表示对 x 数组中 5 个数据的第一、二轮冒泡过程。

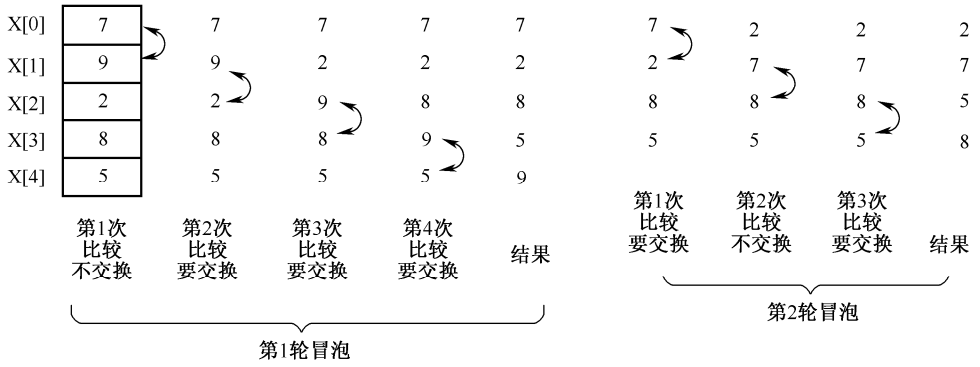

图 4-4　冒泡过程示意图

这样，如有 k 个数，共进行 k-1 轮处理，每轮将当前未排好序的数中最大者“沉”到下面，从而完成排序。k-1 轮是最多的排序轮数，只要在某一轮排序中没有进行数据交换，说明已排好序，可以提前退出外循环，结束排序。程序中通过设置标志变量 flag 来实现，每轮冒泡前置 flag 为 0，若冒泡过程中要交换某两元素的值，则执行 flag=1；冒泡后，若 flag 仍为 0，说明没有进行一次元素值交换操作，也表示元素值已排好序，故用 break 提前结束排序过程。

程序如下：

```
//*****ex4_5.cpp*****
#define N 10
#include <iomanip>
#include <iostream>
using namespace std;
void sort(int x[],int k)                    //sort 函数对形参数组 x 中的前 k 个元素排序
{   int i,j,t,flag;
    for (j=0;j<k-1;j++)                     //多轮冒泡
    {   flag=0;                             //标志置初值
        for (i=0;i<k-j-1;i++)               //控制每一轮处理
          if (x[i]>x[i+1])                  //相邻元素值交换位置
                  {   t=x[i];
                      x[i]=x[i+1];
                      x[i+1]=t;
                      flag=1;               //有元素交换位置，改标志为 1
                  }
          if (flag==0) break;               //没有交换元素，结束循环
    }
}
void print(int b[],int k)                   //print 函数输出数组 b 中的前 k 个元素
{   int i;
     for (i=0;i<k;i++)
      { if (i%5==0) cout<<'\n';
        cout<<setw(6)<< b[i];   }
}
```

```
int main()
{   int a[N];
    int i;
    cout<<"\ninput:";
    for (i=0;i<N;i++)
        cin>>a[i];        //输入 N 个数据到数组 a 中
    sort(a,N);
    print(a,N);
    return 0;
}
```

在调用 sort 函数的过程中，形参 x 与实参 a 共用相同的存储区域，即 x[i]就是 a[i]，sort 函数将 x 数组排好序，也就是将 a 排好了序。

【例 4.6】编写函数求二维数组（10 列）形参的各行元素之和，并将结果存于一个一维数组中。

程序如下：

```
//*****ex4_6.cpp*****
#define N 4
#define M 3
#include <iomanip>
#include <iostream>
using namespace std;
void sum_row(int b[][N],int c[],int k)        //求二维数组 b 的 k 行元素之和
                                              //结果返回在一维数组 c 中
{   int i,j;
    for (i=0;i<k;i++)
        for (c[i]=0,j=0;j<N;j++) c[i]+=b[i][j];
}
void print(int b[],int k)                     //print 函数输出数组 b 中的前 k 个元素
{   …                                         //代码同上例 4.5
}
int main()
{   int x[M][N],y[M];
    int i,j;
    cout<<"\ninput:";
    for(i=0;i<M;i++)                          //输入 x 数组
        for(j=0;j<N;j++)
            cin>>x[i][j];
    sum_row(x,y,N);
    print(y,N);
    return 0;
}
```

从上例可知，二维形参数组的第一维大小可省略，但第二维大小必须指定。类似地，用多维数组作函数参数时，形参的第一维可以不指定大小，但其他维必须指定。另外可发现函数调用 sum_row(x,y,N)可以用实参数组 y 一次带回 N 个数据。一般而言，用数组作为参数可以将函数处理中得到的多个结果值带回主调函数，这也是从被调函数获得多个结果的一种方法。

4.2　指针及其应用

指针是 C++的一种重要数据类型，其用途广泛。借助指针，用户可用灵活多变的方式来访问内存中的变量、数组、字符串、类对象和调用函数；利用指针做函数参数，使得被调函数可以间接访问主调函数中的变量；用指针可建立变量之间的逻辑指向关系，从而构造复杂的数据结构；可以用指针来引用程序在运行过程中动态地申请的内存空间或创建的变量；在 C++面向对象编程中，借助指针，利用虚函数能实现程序运行时的多态性。因此用指针的机会较多。初学者要通过多编程，多上机调试程序来理解指针的概念，掌握其使用规律并将它应用于今后的实际编程中。

4.2.1　指针的概念

程序运行时，程序中的变量（数据）和指令都要占用内存空间。计算机（中央处理器）如何找到指令，执行的指令又如何找到它要处理的变量呢？这得先介绍内存地址。内存是以字节为单位的一片连续存储空间，为了便于访问，计算机系统给每个字节单元一个唯一的编号，编号从 0 开始，第一字节单元编号为 0，以后各单元按顺序连续编号，这些编号称为内存单元的地址，利用地址来标识内存单元，就像用房间编号来标识一栋大楼的各个房间一样。地址的具体编号方式与计算机体系结构有关，如同大楼房间编号方式与大楼结构和管理方式有关一样。对于 C++程序中定义的变量，编译系统会根据变量数据类型为其分配相应数目的字节存储单元。如有下列定义：

```
int a;
char b,c;
float x;
```

则给整型变量 a 分配 4 字节的存储空间，给字符变量 b 和 c 各分配 1 字节，给变量 x 分配 4 字节的存储空间。内存空间分配如图 4-5 所示。

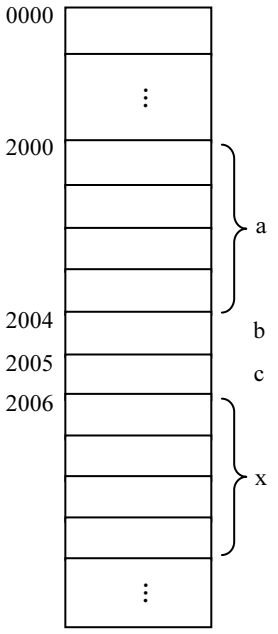

图 4-5　存储空间分配示意

系统给某变量所分配存储空间的首字节地址称为该变量的地址。如 a 的地址为 2000，b 的地址为 2004，x 的地址为 2006。可见，通过地址 2000 就可找到变量 a，地址就像存储单元的指示标，在高级语言中形象地称地址为指针，也可说指针就是地址。

在 C++源程序中，变量名标识变量占据的存储空间和其中存储的数据（变量值），C++编译系统将变量名转换为变量地址，变量占用存储空间的大小由其类型决定。机器指令对变量或内存单元的访问则使用其地址。因此，我们编程时使用变量名 a 来存取 2000 处的存储空间，计算机最终则是使用地址 2000 来访问变量 a 的存储空间。这种按变量名或地址存取变量值的方式称为变量的“直接存取”方式。

与“直接存取”方式相对应的是“间接存取”方式，这种方式要利用一种特殊的变量来存放内存单元或变量的地址。如图 4-6（a）所示，系统为特殊变量 p 分配的存储空间地址是 4000，p 中保存的是变量 a 的地址，即 2000，当要读取变量 a 的值时，不是直接通过变量名 a

或地址 2000 去取 a 的值 123，而是先通过变量名 p 取得 p 的值 2000，即 a 的地址，再根据地址 2000 读取它所指向单元的值 123。这种通过变量 p 获得变量 a 的地址，再由该地址存取变量 a 的值的方式称为变量 a 的“间接存取”方式。通常称变量 p 指向变量 a，变量 a 是变量 p 所指向的对象，它们之间的关系可以用图 4-6（b）表示。

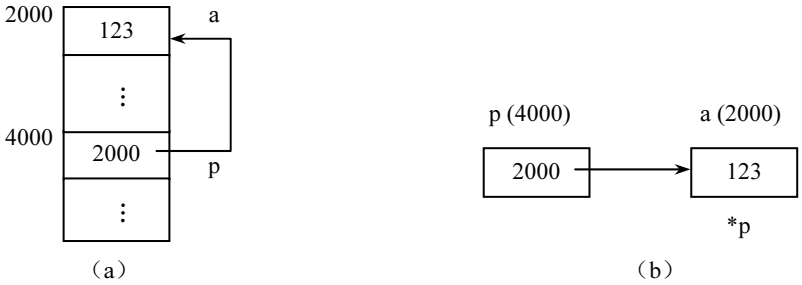

图 4-6　间接存取示意图

用来存放地址（指针）的变量，称为指针变量，如上面的变量 p。通过指针变量可访问它所“指向”的变量，通过这种“指向”关系可建立变量之间的逻辑联系。变量的地址（或指针）是常量，如变量 a 的指针 2000。

如上所述，指针和指针变量是有区别的，但有时统称为指针，读者可根据上下文理解其具体含义。

4.2.2　指针变量的定义及初始化

指针变量定义的一般格式为：

```
数据类型　*变量名;
```

其中，符号“*”表示这里定义指针变量；类型表示该指针变量能指向的对象的类型，也称为指针变量的基类型。

```
int *p;          //指针变量 p 能指向整型变量，p 的基类型为整型，也称 p 为整型指针
char *q;
float *r;
```

定义指针变量时也可指定其初值，如：

```
int a,*p=&a;          //&为取地址运算符，在前面章节已介绍过
```

这里在定义指针变量 p 时，将其初值设为变量 a 的地址，说明指针 p 指向 a，此后，可用 *p 来间接访问变量 a，在程序语句或表达式中可用*p 来表示 a，此处“*”是指针运算符，或叫间接访问运算符、间址运算符，用来访问其后指针所指对象，与指针变量定义处“*”的含义、功能不同。如下二条语句：

```
a=123;
*p=123
```

都能对变量 a 进行赋值。*p 与 a 是等价的，表示同一存储空间，见图 4-6（b），*p 可以像变量 a 一样使用。

通过指针变量间接访问某变量（存储空间）前，要确保指针变量指向该变量，若指针变量存放的是别的变量（存储空间）的地址，即指针变量指错了位置，则程序运行时很可能出错，因为可能读取了别的变量的值或将数据写入了别的存储空间而破坏了计算机内存中的指令或数据。

为说明某指针变量不指向任何变量，可将指针变量初始化为 NULL 或 0。也称指针值为

NULL 的指针为空指针。对于静态的指针变量，如在定义时未给它指定初值，系统自动给它指定初值为 0。

4.2.3　指针的运算

1. 间接访问运算

前面已经使用了间接访问运算*，指针指向的对象可以表示成如下的形式：

*指针

间址运算符*的运算对象必须出现在它的右侧，且运算对象只能是指针变量、地址或值为地址的表达式。

运算符*和&都是单目运算符且具有相同的优先级，结合方向从右到左。因此，表达式&*p与&(*p)等价，是对变量*p 取地址。若 p 指向 a，则 p 的值等于&a，*p 间接表示 a，&*p 的值等于&a；a 与*(&a)等价。

注意：&a 算出 a 的地址，是指针常量。

请思考：*p 等于&a 吗？指针变量定义中的“*p=&a;”与赋值语句“*p=a;”的区别。

2. 指针变量的赋值

程序运行时，可用赋值运算修改指针变量的值。可以将变量的地址赋给指针变量，也可以把一个指针变量的值赋给另一指针变量，还可以给指针变量赋 NULL 值。例如：

```
int i,*p1,*p2,*p3;
p1=&i;                    //将 i 的地址赋给 p1，使 p1 指向 i
p2=p1;                    //将 p1 的值赋给 p2，这样 p1 和 p2 均指向 i
p3=NULL;                  //给 p3 赋空值
```

用一种基类型的指针或指针变量给另一种基类型的指针变量赋值时，要进行强制类型转换，例如，若续上还有：

```
char *pc;
```

若要让 pc 指向 i 则可用 pc=(char *)&i;或 pc=(char *)p1;。

3. 指针的加减运算

指针可以与整数进行加、减运算得出新的地址值。在对指针进行加、减整数 n 时，其结果不是直接加、减 n，一般是加、减“n×sizeof(指针基类型)”。在 Visual C++ 6.0 编译系统中，基类型为字符型的指针加 1，结果增加 1；基类型为整型的指针加 1，结果是加 4（整型数据占 4 个字节）；基类型为 float 类型的指针加 1，结果是加 4，其他类推。

例如有下列定义：

```
int *p,a=1,b=3,c=5;
```

假设 a，b，c 三个变量被分配在一片连续的内存区，a 的起始地址为 2000，如图 4-7（a）所示。

语句 p=&a;使 p 指向变量 a，即 p 的内容是 2000，如图 4-7（b）所示。

语句 p=p+2;指针向下移两个整型变量的位置，p 的值为 4000+2×sizeof(int)=2000+2×4=2008，而不是 2002，因为整型变量占 4 个字节，如图 4-7（c）所示。

指针变量自增自减运算，指针变量值一般是加、减 sizeof(指针基类型)，即其基类型数据所占的字节数。

相同基类型的两个指针可以相减，结果是两指针（地址）值相隔的存储空间个数，一个存储空间一般包含 sizeof(指针基类型)个字节。对于图 4-7，表达式&c-&a 值为 2。若是小值指

针减大值指针，结果为负数。两个指针不能相加，不管其基类型是否相同。

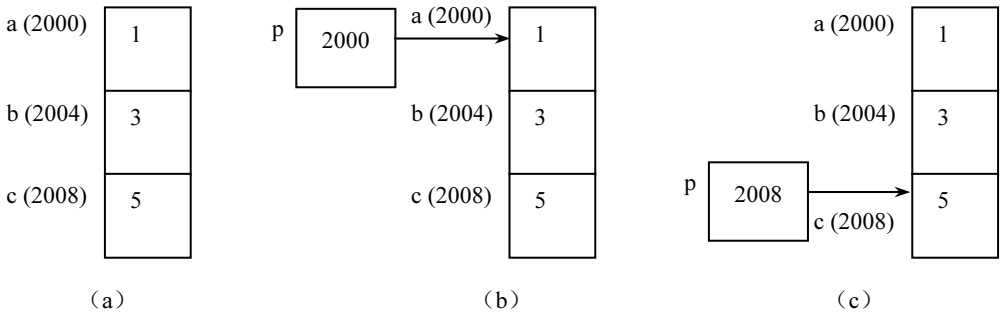

图 4-7　指针移动示意图

4. 指针的关系运算

相同基类型的两个指针可以进行各种关系运算，指向低（小）地址端存储单元的指针小于指向高（大）地址端存储单元的指针。对于图 4-7，表达式&a<&b 为真。

4.2.4　指针作函数参数

指针变量可以作函数的形参，对应的实参用地址或指针变量。函数调用时实参向形参传递的数据是地址（值），使得形参和实参指向主调函数中的同一个变量（内存空间），函数改变形参指向的变量，也就是改变实参指向的变量。尽管被调函数不能改变实参指针的值，但可以改变实参指针所指变量的值。因此，指针参数为被调函数修改主调函数中的变量（数据）提供了手段。

【例 4.7】编写能交换两整型变量值的函数 swap()。

分析：这里要改变（主调函数中）两变量的值，可用两个指针变量作函数形参，让函数利用形参指针变量间接访问主调函数中的两变量。调用 swap()函数时，用两待交换值的整型变量的地址作实参。

程序如下：

```
//*****ex4_7.cpp*****
#include <iostream>
using namespace std;
void swap(int *p1,int *p2)   //交换 p1、p2 所指变量，即*p1、*p2 的值。
{   int t;
    t=*p1;
    *p1=*p2;
    *p2=t;
}
int main()
{   int a,b;
    cin>>a>>b;
    swap(&a,&b);   //要交换 a 和 b 的值，实参为 a、b 的地址，也可是指针变量
    cout<<"a="<<a<<",b="<<b;
    return   0;
}
```

swap()函数调用过程如图 4-8 所示。两个实参分别为变量 a、b 的地址&a、&b，按值传递

规则，形参 p1 和 p2 分别得到&a、&b，从而分别指向 a、b，*p1、*p2 是 a、b 的间接访问表示形式，函数交换*p1 和*p2 的值，也就是使 a 和 b 的值互换。函数调用结束后，虽然 p1 和 p2 已释放不存在了，但 a 与 b 的值已经互换。

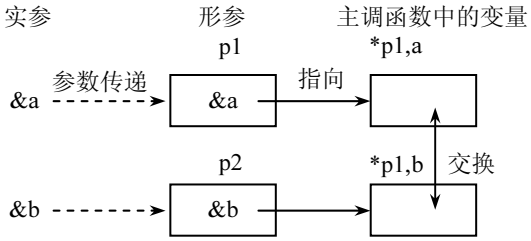

图 4-8　swap()函数调用过程示意

下面分析一下两种错误形式的 swap()函数。

如果 swap()的形参不用指针变量，写成如下 swap1()形式：

```
void swap1(int x,int y) //交换 x、y 的值。
    { int z;
      z=x; x=y; y=z;
    }
```

如有函数调用 swap1(a,b)，实参 a、b 的值分别传递给形参 x、y；函数 swap1()完成形参 x、y 的值的交换，但实参 a、b 的值不变，即实参向形参单向值传递，形参值的改变不影响对应实参的值。所以不能达到 a、b 值互换的目的。

如果 swap()的形参用了指针变量，但写成如下 swap2()形式：

```
void swap2(int *p1,int *p2)   //交换 p1、p2 的值。
{   int *p;
    p=p1; p1=p2; p2=p;
}
```

如有函数调用 swap2(pa,pb)，其中实参 pa、pb 为分别指向 a、b 的指针变量。swap2()函数调用过程如图 4-9 所示。实参 pa、pb 传递给形参 p1、p2，函数交换了 p1、p2 的值，但 pa 和 pb 的值不变，*pa、*pb 即 a、b 的值也不会交换。

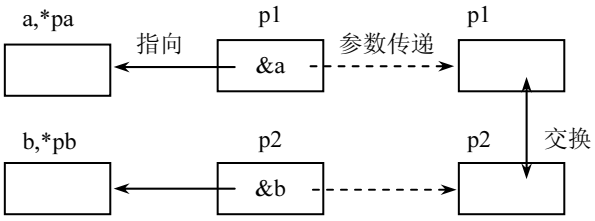

图 4-9　指针参数的单向传递

将上面的 swap2()改成前面的 swap()就能得到正确结果，实参可以用指针变量 pa、pb，也可以直接用变量地址&a、&b。

【例 4.8】请编写函数 f()根据三角形的边长同时求出其周长和面积。

分析：函数 f()要返回两个值，而 return 语句只能返回一个值，故用 return 语句返回结果不能奏效。可以采用两个指针变量做形参，让周长和面积值分别存放到两个形参指针变量所指

的两个变量中，函数调用后，主调函数读取两个变量的值即可。

程序如下：

```
//*****ex4_8.cpp*****
#include <iostream>
#include <cmath>
using namespace std;
int main()
{   float a,b,c,perimeter,area;
    void f(float, float, float, float *, float *);   //函数说明
    cin>>a>>b>>c;                    // 输入时确保能以 a、b、c 为边长构成三角形
    f(a, b, c, &perimeter,&area);
    cout<<"周长："<< perimeter<<"面积："<< area<<endl;
    return 0;
}
void f(float a, float b, float c, float *p1, float *p2)
{   float s;
    s=a+b+c;
    *p1=s;                    //让周长存放到形参指针变量 p1 所指的变量中
    s=s/2;
    *p2=sqrt(s*(s-a)*(s-b)*(s-c)); //面积存放到形参指针变量 p2 所指的变量中
}
```

上述例子表明，指针变量作函数形参时，对应的实参是一个变量的地址，将该变量的地址传给形参，则形参指向该变量。这样在被调用的函数中，可以用形参指针变量间接访问该变量，函数执行完毕返回到主调函数后，该变量就得到了新的值。可以通过多个指针变量形参带回多个变化了的值，这也是指针变量作函数形参的一个功能。后面章节还会提到，用指向结构、类对象等复杂结构数据的指针变量作函数形参，在函数调用时可以避免传递（复制）大量数据，能提高程序运行速度。

4.2.5 返回指针值的函数

函数的返回值可以是指针类型的数据，即地址。返回指针值的函数也称为指针函数。

定义指针函数的一般形式为:

```
数据类型  *函数名(形式参数表)
```

函数名之前的符号“*”表示函数返回指针值，例如：

```
int *f(int x,int y)
{   局部变量定义
    语句序列
}
```

函数 f()即是一个指针函数，返回值为一个指向 int 类型变量的指针，即基类型是 int 的指针。显然，在指针函数的函数体中应有返回指针或地址的语句，形式如下：

```
return  指针;
```

指针函数返回值不能是本函数中的局部变量的地址，因为本函数执行完毕返回主调函数后，函数中的局部变量全部释放，主调函数再去访问它，不能保证结果正确。返回值可以是主调函数中的局部变量的地址、数组元素的地址或全局变量的地址等。

4.2.6　指向函数的指针

程序运行时，一个函数包括的指令序列要占据一段内存空间，这段内存空间的首字节编号称为函数的入口地址，编译系统用函数名代表这一地址。如函数 fun()，其存储示意如图 4-10 所示，函数名 fun 就表示第一条指令的地址（6000）。运行中的程序调用函数时就是通过该地址找到这个函数对应的指令序列，故称函数的入口地址为函数的指针，简称函数指针。显然，函数名是函数指针常量。可以定义一个指针变量，用来存放函数的入口地址，如图中下部的变量 fq，这种存放函数入口地址的指针变量称为指向函数的指针变量，简称函数指针变量。显然，通过指向函数的指针变量（如 fq）可以找到函数的第一条指令，因此，利用这个指针变量（如 fq）也能调用函数（如 fun）。

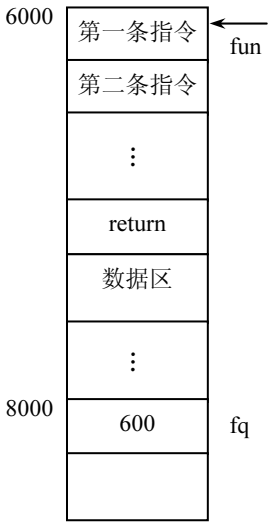

图 4-10　函数存储示意图

用函数名调用函数称为函数的直接调用，用函数指针变量调用函数称为函数的间接调用。

1. 函数指针变量的定义

定义函数指针变量的一般形式为：

数据类型（*变量名）（形参类型表）；

例如：

```
int (*fp)(int,float);
void (*fq)();
```

则函数指针变量 fp 能指向一个返回整型值的函数，该函数第一个形参为整型，第二个形参为浮点型；fq 能指向一个无返回值的函数，且该函数无形参（可省掉 void）。

注意 int (*fp)(int,float);与 int *fp(int,float);的差别，前者定义 fp 为一个指向函数的指针变量，后者说明 fp 是一个函数，该函数返回值为指针，格式不能写错。若要让函数指针变量 fpp 指向一个返回指针值的函数，且指针值基类型为整型，则如下定义：

```
int      *(*fpp)(int,float);
```

一个函数指针变量只能指向某类函数中的一个，该类函数的形参个数、类型、顺序相同且返回值类型也相同，只是名称可不同。这里权且称该类函数为函数指针的相容类函数。

2. 用函数指针变量调用函数

定义了函数指针变量，就可给它赋相容类函数的入口地址。函数名代表函数入口地址，故采用如下形式赋值：

函数指针变量=函数名;

当一个函数指针变量指向某个函数时，就可用它调用所指的函数。一般调用形式为：

函数指针变量 (实参表)

或　(*函数指针变量)(实参表)

【例 4.9】编程求出从键盘输入的 2 个整数中较大者。要求编写函数 max()求 2 个整数中的较大者，在主调函数中用函数指针变量调用该函数。

本问题很简单，程序如下：

```
//*****ex4_9.cpp*****
#include <iostream>
```

```
using namespace std;
int max(int a,int b)
{ return a>b?a:b;}
void main ()
{   int x,y,z;
    int (*p)(int,int);          //定义函数指针变量 p
    cout<<"Enter x,y\n";
    cin>>x>>y;
    p=max;                      //函数入口地址（函数名）赋给指针变量 p
    z=p(x,y); //或者写成 z=(*p)(x,y);用指针方式调用函数，等效于 z=max(x,y)
    cout<<"max("<<x<<","<<y<<")="<<z<<"\n";
}
```

注意：不要把 p=max 写成 p=max(x,y)。

编写程序时，若知道被调函数的名字，一般用函数名调用该函数，没必要用函数指针变量。但有时要在程序运行过程中根据情况临时确定调用哪一个函数，此时用函数指针变量调用函数就显得方便，让同一函数指针变量指向不同的函数就能调用不同的函数，从而增加程序的适应性或通用性。此时可用函数指针变量作函数形参。

3. 函数指针变量作函数的形参

函数指针变量作函数的形参，对应实参是函数指针或函数指针变量。不同的函数指针实参传递过来后，就可调用不同的函数来完成不同的功能。这也是函数指针变量作函数的形参的意义所在，在一些复杂结构程序中有时采用这种函数调用方式。

为说明具体运用过程，举简例如下。

【例 4.10】输入圆的半径，求它的面积和周长。要求编写 float fun(float a,float (*p)(float))，它根据参数 p 来调用不同的函数求面积或周长。

定义基本函数 area()和 perimeter()分别用于求圆的面积和周长。为了说明函数指针变量参数的用法，程序另定义一个函数 fun()。主函数不直接调用两个基本函数，而是调用函数 fun()，并提供圆的半径和基本函数名作为实参。由函数 fun()根据主调函数传递来的参数值再去调用相应的函数。

程序如下：

```
//*****ex4_10.cpp*****
#define PI 3.1415926
#include <iostream>
using namespace std;
float area(float), perimeter(float);      //函数声明
float fun(float,float (*)(float));        //含有函数指针形参的函数的声明
void main()
{   float r,s,l;
    float (*q)( float);                   //定义函数指针变量 q
    cout<<"请输入半径:\n";
    cin>>r;
    s=fun(r,area);                        //直接用函数名作实参，通过函数 fun 求面积
    q=perimeter;                          //求周长函数函数名（入口地址）赋给 q
    l=fun(r,q);                           //用函数指针变量作实参，通过函数 fun 求周长
    cout<<"面积:"<<s<<endl;
    cout<<"周长:"<<l<<endl;
```

```
}
float fun(float r,float (*p)(float))
{   float y;
    y=p(r);   //用形参 p 所得实参传来的函数入口地址调用相应函数
    return(y);
}
float area(float r)                  // 函数定义
{ return (PI*r*r);}
float perimeter(float r)
{ return(2*PI*r);}
```

当主调函数（如上例 fun）的形参是函数指针变量时，可以用不同的函数名实参与形参对应，从而实现在不对主调函数进行任何修改的前提下调用不同的函数（如上例 area、perimeter），完成不同的功能；主调函数可以是别人编写的或系统软件提供的，只要别人给了函数目标代码或函数库即可，而不必要源代码（因别人保密源代码）。当然，实参也可用函数指针变量，当给该指针变量赋不同的函数入口地址（指向不同的函数）时，即可实现在主调函数中调用不同的函数。

函数指针与变量指针性质相同，都是内存地址，区别是变量指针指向内存中的数据区，而函数指针指向内存中的函数指令区。

4.3　指针与数组

数组元素所占存储空间的地址就是元素的指针。可以利用指针间接访问或引用数组元素。根据数组各元素在内存中连续存放这一特点，并且利用前面介绍的指针运算，借助指针可以实现对数组或数组元素灵活多变的引用方式，这使得程序编写方便。另外，数组名和指针作函数参数时，在函数定义、说明、调用时，两者可通用，可相互替换，因此，这也使得程序编写方便。

4.3.1　指针与一维数组

1. 指向数组元素的指针变量

当指针变量的基类型与数组元素数据类型相同时，将数组元素的地址或一维数组名赋给它，此时指针变量就指向数组元素。

例如：

```
int a[5]={1,2,3,4,5};
int *p;
```

为了让 p 指向数组元素 a[0]，可以给它赋值：p=&a[0];，数组章节说过，一维数组名表示其首元素（a[0]）的地址，因此也可写成赋值：p=a;，如图 4-11 所示。

如要使 p 指向数组元素 a[3]，则执行：p=&a[3];。

可用数组名或数组元素地址初始化指针变量，让指针变量指向数组或数组元素。例如：

```
int a[5]={1,2,3,4,5};
int *p=a;                          //等效于 int *p=&a[0];，使 p 指向 a[0]
```

注意：数组名 a 是常量指针或称指针常量，而 p 是指针变量。a 代表的值在程序运行过程中不能被改变，不能进行 a++，a=a+1 等类似的操作；而 p 是指针变量，其值是可以改变的，当赋给 p 不同元素的地址值时，它可指向不同元素，如 p++;操作使 p 指向下一个元素。

2. 一维数组元素的指针引用法

如上定义了 a 和 p，且 p 指向数组的首元素后，则 a、&a[0]、p 的值相等，故*a、*p 可表示 a[0]；由指针的运算规则和数组的存储特点可知，a+1、p+1 都等于 a[1]的地址& a[1]，故*(a+1)、*(p+1)都表示 a[1]；类似地，a+i、p+i 都等于 a[i]的地址&a[i]，故*(a+i)、*(p+i)都表示 a[i]，也称它们为 a[i]的指针表示法，如图 4-12 所示。

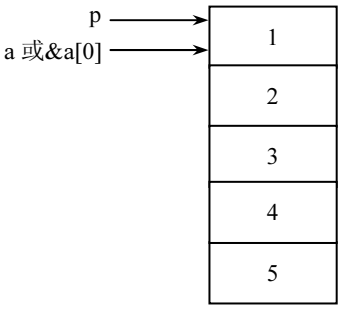

图 4-11　数组指针示意图

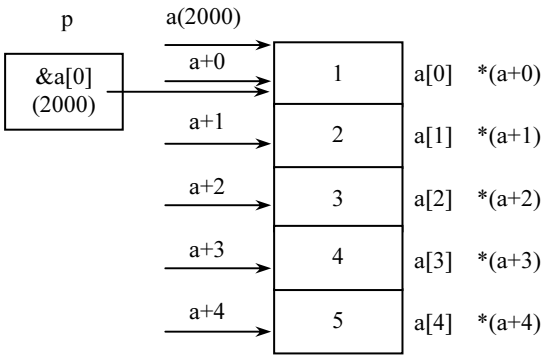

图 4-12　数组元素的多种表示法

指针变量 p 也可以带下标，亦即*(p+i)可写成 p[i]，p[i]也表示 a[i]，称 p[i]为 a[i]的指针下标表示法；&p[i]也等于&a[i]。此时数组元素及其地址的表示法汇总于表 4-1。

表 4-1　数组元素及其地址的表示法

元素地址	地址描述	数组元素	元素描述
a[0]的地址	a、p、&a[0]、&p[0]	a[0]的值	*a、*p、a[0]、p[0]
a[1]的地址	a+1、p+1、&a[1]、&p[1]	a[1]的值	*(a+1)、*(p+1)、a[1]、p[1]
a[i]的地址	a+i、p+i、&a[i]、&p[i]	a[i]的值	*(a+i)、*(p+i)、a[i]、p[i]

由表可见，指向数组元素的指针变量 p 和数组名 a 在表示数组元素及其地址时可相互替换。

思考：如果 p 已指向 a[3]，则 p+i 代表哪个数组元素地址，*(p+i)、p[i]代表哪个数组元素？p-2 代表哪个元素的地址？

【例 4.11】使用不同的数组元素表示法输出整型数组 a 各元素。

```
//*****ex4_11.cpp*****
#include <iostream>
#include <iomanip>
using namespace std;
void   main()
{   int a[5]={1,2,3,4,5};
    int i,*p=a;
    for (i=0;i<5;i++)
        cout<<setw(6)<< a[i];              //下标法
    cout<<'\n';
    for(i=0;i<5;i++)
        cout<<setw(6)<<*(a+i);             //指针法；或用 *(p+i)
    cout<<'\n';
```

```
    for(p=a;p<a+5;p++)
        cout<<setw(6)<<*p;              //指针法，但此处 p 向下移动
    cout<<'\n';
    for(p=a,i=0;i<5;i++)
        cout<<setw(6)<<p[i];            //指针下标法
    cout<<'\n';
}
```

思考：如果要按从后向前的次序输出各元素有哪些控制方法？

当指针变量与自增运算符、自减运算符、间接访问运算符构成如下表达式时，要注意其运算次序。

（1）p++使 p 指向数组的后一个元素；p--使 p 指向数组的前一个元素。这前面已讲过。

（2）*p++等价于*(p++)，其作用是先得到 p 指向变量的值（即*p），然后再使 p 指向后一个元素。因此，上例的输出还可以写成如下循环语句：

```
for(p=a,i=0;i<5;i++)   cout<<setw(6)<<*p++;
```

（3）*(++p)的作用是先使指针变量 p 自增，再取*p。

（4）(*p)++表示 p 所指向的变量值加 1。

4.3.2　一维数组名和指针作函数参数的进一步讨论

上文指出，指针变量和数组名在表示数组元素及其地址时可相互替换，唯有不同的是数组名是常量，只能指向数组首元素，值不能改变，而指针变量可以指向任意元素。但是，作函数形参的数组名与指针变量用法几乎一样。如下函数求 x 数组的 n 个数据之和：

```
int sum(int x[],int n)
{ int i,s=0;
  for(i=0;i<n;i++) s+=x[i];          //可写成 s+=*x++;
  return s;}
```

其中 s+=x[i];可写成 s+=*x++;，将形参数组名看做指针变量，更容易理解其接受实参值的过程。

数组元素或连续内存单元可用指针来表示，因此 sum()函数又可定义如下：

```
int   sum(int *x,int n)              //改形参数组为指针变量
{ int i,s=0;
  for(i=0;i<n;i++)   s+=*(x+i)       //也可写成 s+=x[i];
  return   s;}
```

在函数说明时，数组形参与指针形参也等价。sum()函数可说明为

```
int sum(int [ ],int);
```

或

```
int sum(int *, int );
```

可见，一维数组名和指针变量作函数形参时可以通用。

数组名和指针变量作形参时，实参可用数组名或指针。在类型对应关系上共有 4 种情况，即：形参用数组名，对应实参可用数组名或指针；形参用指针变量，对应实参也可用数组名或指针。

二维数组名和行指针变量作函数参数时也可以通用，见下节。

4.3.3　指针与二维数组

二维数组的地址涉及每个元素地址、每行地址，下面针对一个具体数组来介绍它们的具

体含义和表达形式。

1. 二维数组的地址

如前所述，二维数组可看成一个一维数组，其元素本身又是一个一维数组。例如，有下面的二维数组定义：

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

a 数组是一个由 a[0]、a[1]和 a[2]组成的一维数组，其元素本身又是一维数组，如其中 a[0]又是包含 a[0][0]、a[0][1]、a[0][2]和 a[0][3]的一维数组，a[1]和 a[2]类似地也各包含四个元素。a 的 12 个元素逻辑上构成如下 3 行 4 列二维阵列，如图 4-13 所示。

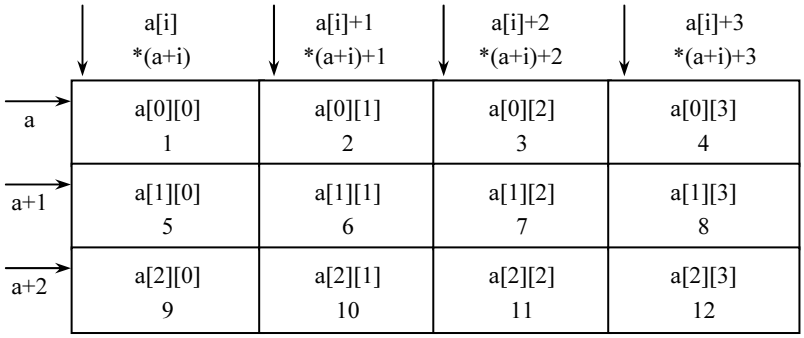

图 4-13　二维数组的指针表示

视 a 为一维数组，则数组名 a 代表其第一个元素 a[0]的地址&a[0]，即阵列第一行的地址，a+1 代表&a[1]，代表第二行地址，a+i 代表&a[i]，代表阵列第 i+1 行地址。因此也称 a、a+i 为行指针。

视 a[0]为一维数组，则数组名 a[0]代表其第一个元素 a[0][0]的地址&a[0][0]，a[0]+1 代表&a[0][1]，a[0]+j 代表&a[0][j]；类似地，a[1]代表&a[1][0]，a[1]+j 代表&a[1][j]；a[i]代表&a[i][0]，a[i]+1 代表&a[i][1]，a[i]+j 代表&a[i][j]。

视 a[0]、a[1]、a[i]为一维数组 a 的元素时，沿用前面一维数组元素的指针表示法形式，则 a[0]形式上可表示成*(a+0)，简写为*a，a[1]可表示为*(a+1)，a[i]可表示为*(a+i)，但此处它们不代表数组元素，而是代表各行第一个元素（第一列各行元素）的地址&a[0][0]、&a[1][0]和&a[i][0]，相应地，*(a+i)+j 就代表&a[i][j]。

由上可知，a[i][j]的指针引用形式有：*(a[i]+j) 、*(*(a+i)+j)，沿用前面一维数组元素的指针下标法形式，后者还可写为(*(a+i))[j]。对于特例 a[0][0]，其等价表示形式有：*a[0]、**a 和(*a)[0]。

将上面二维数组的各种地址和元素指针表示法汇总于表 4-2，为描述方便，此处将最前面一行称为第 0 行，最前面一列称为第 0 列。

表 4-2　二维数组的各种地址和数组元素指针表示法汇总表

含义	表示形式
第 0 行地址	a、&a[0]
第 0 行第 0 列元素地址	a[0]、*(a+0)、*a、&a[0][0]
第 0 行第 j 列元素地址	a[0]+j、*(a+0)+j、*a+j、&a[0][j]

续表

含义	表示形式
第 1 行地址	a+1、&a[1]
第 1 行第 0 列元素地址	a[1]、*(a+1)、&a[1][0]
第 i 行地址	a+i、&a[i]
第 i 行第 0 列元素地址	a[i]、*(a+i)、&a[i][0]
第 i 行第 j 列元素地址	a[i]+j、*(a+i)+j、&a[i][j]
第 i 行第 j 列元素	*(a[i]+j)、*(*(a+i)+j)、(*(a+i))[j]、a[i][j]

2. 行指针变量

行指针是指向一维数组整体的指针，存储行指针的变量叫行指针变量。如

```
int (*p)[4];
```

定义了行指针变量 p，它能指向一个包含 4 个 int 型元素的一维数组。在定义中不要丢掉圆括号，若写成

```
int *p[4];
```

则定义了一个指针数组 p（后面介绍）。

行指针变量 p 是一个指向由 4 个整型元素组成的一维数组，对 p 作增减 1 运算就表示前进或后退 4 个整型元素。

设有变量定义

```
int a[3][4],(*p)[4];
```

则赋值语句

```
p=a;            //a 表示第 0 行的地址，行指针
```

使 p 指向二维数组 a 的第 0 行。从表 4-2 知，p 所指的一维数组中的第 j 个元素地址可表示为*p+j，元素则可表示为*(*p+j)或(*p)[i]。

另外，表达式 p+1 的值指向二维数组 a 的第 1 行，p+i 指向二维数组 a 的第 i 行，这与 a+i 一样。同样 p[i]+j 或*(p+i)+j 指向 a[i][j]，所以，数组元素 a[i][j]的引用形式可写成：

```
*(p[i]+j)、*(*(p+i)+j)、(*(p+i))[j]、p[i][j]
```

与表 4-2 对照可知，用 p 表示二维数组元素和地址与用数组名表示形式一致。而且，行指针变量和二维数组名作函数参数时两者也可通用。如函数定义首部

```
void input(int x[][N],int m)
```

可写成

```
void input(int (*x)[N],int m)
```

4.4　字符串

字符串是一种非数值类型数据，它也是计算机加工处理的对象。C 语言用字符数组和指针表示字符串，C++仍然支持这两种表示方法，另外，C++还可用字符串类型 string（后面章节介绍）对象表示字符串，string 类型最终也是用字符数组和指针表示字符串，因此学好前两种表示方法对学习 string 类型是有帮助的，本节介绍这两种方法。

4.4.1 字符串的概念

若干个字符连续排列起来构成的序列就是字符串。在源程序中要用一对双引号将字符串括起来，如："China"，这对双引号只充当界限符，不是字符串的成员，仅便于编译系统识别源程序中的字符串。字符串中字符的个数叫字符串的长度。

程序运行时，字符串在内存中占据一片连续的字节，串中的每个字符占一个字节，运行中的程序根据串首字符所占字节的地址就可以找到整个字符串；为了让程序检测到串尾字符或测定字符串的长度，C++语言规定在字符串尾添加一个字符'\0'作为字符串结束标志，这样程序从串首字符向后找，遇到了字符串结束标志字符'\0'就表示找到了整个字符串，也就识别了字符串。"China"的存储示意见图 4-14，图中 1000 表示首字符所占字节的地址。

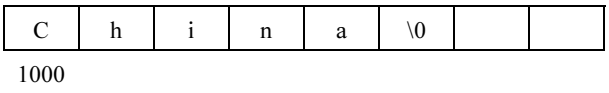

图 4-14　字符串存储示意图

字符串可以包括转义字符及 ASCII 码表中的控制字符（以转义字符出现）。字符串可以包含汉字，如："中国"、"中国=China"。一个汉字占两个字节，长度计为 2。

输入输出字符串时，仅作为界限标志的双引号和'\0'均不要输入，也不会输出。

4.4.2 字符串的存储表示法

1. 字符数组表示法

字符数组可用来存储字符串，其元素依次存储串中的各字符。内存中字符串尾总附加'\0'字符，它要占一个字节，所以长度为 L 的字符数组能容纳长度不超过 L-1 的任何字符串，若试图容纳长度超过 L-1 的字符串，将在编译时出现语法错误或运行时数组越界，必须设法避免。字符数组名代表首字节地址，通过字符数组名就可以找到存于其中的字符串。用数组元素就可以找到存于其中的字符。字符数组为字符串提供存储空间，因此字符数组相当于字符串变量。如有定义：

```
char s1[]＝{'s','t','u','d','e','n','t','\0'};
```

其存储状态如图 4-15 所示：

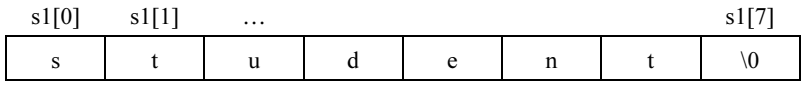

图 4-15　字符数组存储示意图

显然，一个元素存放一个字符。

若通过赋值语句将各字符分别赋给字符数组的各元素，注意串尾要添上' \0'。为简便起见，在编写程序时，可以直接用字符串初始化字符数组如下：

```
char s1[8]={"student"};
```

花括号也可省略，简单地写为

```
char s1[8]="student";
```

为避免计算字符串的长度，可以写成：

```
char s1[]="student";
```

系统将双引号括起来的字符依次存于字符数组的各个元素，并自动在末尾补上字符串结束标志字符'\0'，并一起存到字符数组中，上面数组 s1 的长度为 8，有效字符只有 7 个。若有定义：

```
char s2[]＝{'s','t','u','d','e','n','t'};
```

s2 的长度为 7，s2 中未存放完整字符串，如执行语句：

```
cout<<s2;
```

很可能得不到正确输出结果，该语句将在输出 student 之后继续输出，直至遇到 8 位全 0 代码（即'\0'）为止。但写成：

```
char s2[8]＝{'s','t','u','d','e','n','t'};
```

可以得到正确输出，字符数组的初始化也与其他数组的初始化一样，对部分未明确指定初值的元素，系统自动设定 0 值字符，也正是字符串结束标志符'\0'。

若要表示多个字符串，可以采用二维字符数组，例如：

```
char str2[][30]={"I am happy!","I am a student."};
```

用数组存储的字符串，串中字符通过数组元素来引用，例如：

```
char s[]="I am a student.";
```

用 s[i]或*(s+i)就能访问数组中字符串的第 i 个字符。

上面说明了用字符数组存储字符串，另一方面也表示可用字符串初始化字符数组。

2. 指针表示法

若将字符串首字符所占字节的地址存于字符基类型的指针变量中，则程序可通过该指针变量访问字符串或其中的某个字符，因此，指针是字符串的一种表示法，例如：

```
char *cp ="I am a student.";
```

定义了指向字符的指针变量 cp，并初始化 cp 使它指向字符串的第一个字符（'I'）。以后就可通过 cp 间接访问字符串或其中的某个字符，如*cp 或 cp[0]的值就是'I'，用*(cp+i)或 cp[i]就能访问字符串中的第 i 个字符。有时也称 cp 是指向字符串的指针变量。也可先定义 cp，事后再赋值，如下：

```
char *cp;
cp="I am a student.";
```

这是将字符串首字符所占据字节的地址赋给 cp，而不是将字符串赋给 cp，cp 并不为字符串提供存储空间。可以把该字符串"I am a student."看作字符串常量，由编译系统为它分配存储空间。

用字符数组和字符指针变量都能表示字符串并参加各种运算，但两者之间是有区别的。字符数组元素在内存中占据一片固定的连续内存空间，每个元素可存放一个字符，因此，字符数组为字符串提供存储空间。在程序运行过程中，字符数组所占的固定内存空间在不同时刻可存放不同的字符串，所以，字符数组相当于字符串变量。数组名代表这片内存空间的首地址，是常量指针，因此它只能指向位于这个固定地址内存中的字符串。而字符指针变量并不为字符串提供存储空间，但同一指针变量在不同时刻可指向内存中不同地址处的字符串，当然也可指向存于数组中的字符串。既然数组名是指针常量，就不能用一个字符串给一个字符数组赋值，例如：

```
char str[50];
str="I am a student.";
```

编译时将产生错误。但能用一个字符串给一个字符数组初始化。当然，字符数组各个元素可单独赋值。

4.4.3 字符串的输入与输出

字符串的输入输出总的来说有两种方式：

（1）逐个字符输入输出，即每次输入输出一个字符，用循环结构控制整个串的输入输出。

（2）将整个字符串作为一个整体一次性完成输入输出。

每种方式具体完成输入输出操作又有两种方法，一种是调用标准 C 输入输出（Standard C I/O）函数库中的字符/字符串输入输出函数（scanf、printf、getchar、putchar、gets、puts 等），C++同样支持这些函数。另一种是用 C++的标准输入输出流对象 cin 和 cout，这种方法使用简单，举例如下：

设有：

```
char s1[20]="I am a student.";
```

逐个输出字符语句：

```
for(i=0;s[i];++i)          //循环条件等价于 s[i]!=0
        cout<<s1[i];
```

将字符串一次性输出：

```
cout<<s1;
```

两者输出结果都为：

```
I am a student.
```

若要从第 i 个字符起输出字符串后面一截，则写成：

```
cout<<&s1[i];            //s1+i
```

若指针变量 cp 指向某个字符串，同样可以用上面两种方式输出。

若程序运行时要输入字符串，那么程序必须为它提供存储空间，一般采用一个够长的字符数组，若有数组定义：

```
char s2[100];
```

则逐个字符输入语句：

```
for(i=0;i<15;++i)
     cin>>s2[i];
s2[i]= '\0';      //额外补加串结束标志
```

将字符串作为一个整体一次性输入语句：

```
cin>>s2;
```

这种方式输入字符串是以空格符、Tab 控制符、回车符作为终止标志，因此字符串中不能含这些字符。若要输入以回车键为终止标志的一行字符则用 cin.getline()。

两种输入方式都必须保证输入的字符串长度比数组长度少 1。

上面是用 cout 对象的重载运算符>>和 cin 对象的<<进行输入输出，还可以用成员函数 cout.put()、cin.get()、cin.getline()进行灵活的输入输出控制，具体见后面输入输出流章节。

4.4.4 字符串处理处理函数

为了便于使用字符串，C 语言的字符串函数库提供了丰富的字符串处理函数，C++仍然兼

容它们。这里介绍其中几个比较常用的字符串处理函数。注意，在程序首部加上#include <string>，其中包含了函数说明。

下面函数调用形式中的参数 str、str1 和 str2，除特别声明外，均是字符数组名、字符（串）指针变量或字符串常量。

1. 字符串连接函数 strcat()

函数原型：char *strcat(char *, const char *);

调用格式：strcpy(str1,str2);

函数功能：将字符串 str2 连接到 strl 的后面，函数原型中的修饰符“const”说明 str2 串的内容不会被改变，而不是要求 str2 为字符串常量。

函数调用返回一个指针值，仍为 str1。与 str1 对应的实参一般为数组，正确使用该函数，要求与 str1 对应的实参尾部有足够剩余空间，以便能容纳 str2 的内容。连接前，str1 和 str2 尾部都有'\0'。连接后，str1 中的'\0'在连接时被覆盖掉，而在新的字符串有效字符之后保留一个'\0'。例如：

```
char s1[30]="this";
char s2[30]=" is";          //或 char *s2=" is";
strcat(s1,s2);              //结果存在 s1 中，第一个实参为数组
cout<<s1;                   //两条语句等价于 cout<<strcat(str1,str2);
```

将输出：

```
this is
```

图 4-16 表示连接前后 str1 与 str2 的内容。

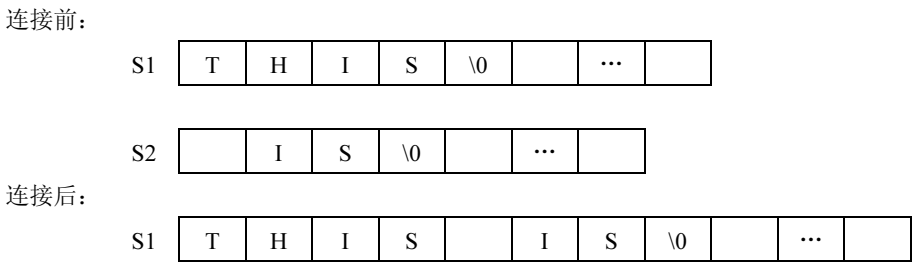

图 4-16　连接前后 s1 与 s2 的内容变化图

2. 字符串拷贝函数 strcpy()和 strncpy()

函数原型：char *strcpy(char *,const char *);

调用格式：strcpy(str1,str2);

函数功能：将字符串 str2 整个复制到字符数组 str1 中，str2 的值不变。

调用该函数时，一般 str1 是字符数组，且 str1 定义得足够大，以便能容纳被拷贝的 str2 的内容。例如：

```
strcpy(str1,"Changsha");
```

在某些应用中，需要将一个字符串的前面一部分拷贝，其余部分不拷贝，调用函数 strncpy()可实现这个要求。函数调用格式：

```
strncpy(str1,str2,n);
```

作用是将 str2 中的前 n 个字符拷贝到 str1（附加'\0'）。其中 n 是整型表达式，指明欲拷贝的字符个数。如果 str2 中的字符个数不多于 n，则该函数调用等价于 strcpy(str1,str2)。

3. 字符串比较函数 strcmp()

函数原型：int strcmp(char *,char *);

调用格式：strcmp(str1,str2);

函数调用 strcmp(str1,str2)比较两个字符串大小。对两个字符串自首至尾逐个字符相比较（按字符的 ASCII 代码值的大小），直至出现不同的字符或遇到'\0'为止。如全部字符都相同，则认为相等，函数返回 0 值；若出现不相同的字符，则以第一个不相同的字符比较结果为准。若 str1 的不相同字符小于 str2 的相应字符，函数返回一个负整数（-1）；反之，返回一个正整数（1）。

注意：对字符串不能进行关系运算，必须调用 strcmp(str1,str2)对字符串作比较。

4. 求字符串长度函数 strlen()

函数原型：unsigned int strlen(char *);

调用格式：strlen(str);

函数功能：求字符串 str 的长度（不包括'\0'）。例如：strlen("right")返回值为 5。

5. 子串查找函数 strstr()

函数调用 strstr(str1,str2)，在 str1 中查找是否包含子串 str2，若找到，则返回第一次出现 str2 的位置（地址），否则，返回空指针 NULL。

6. 字符串大写字母转换成小写字母函数 strlwr()

函数调用 strlwr(str)将 str 中的大写字母转换成小写字母，str 一般用字符数组名。

7. 字符串小写字母转换成大写字母函数 strupr()

函数调用 strupr(str)将 str 中的小写字母转换成大写字母，str 一般用字符数组名。

与字符串有关的库函数还有很多，如：

int atoi(char *str); 将 str 字符串转换成整数。

long atol(char *str); 将 str 字符串转换成长整数。

double atof(char *str); 将 str 字符串转换成 double 型数值。

以上三个函数的说明在 stdlib.h 中。

例如，有以下定义：

```
char s[]="12345";
long n;
```

则语句 n=atol(s);执行后 n 的值为 12345。

如有必要，读者可查阅有关 C++参考手册。

4.4.5 字符串的简单应用举例

【例 4.12】编写函数 void del_ch(char *p,char ch)，删除 p 所指字符串中的 ch 字符。例如：若 p 所指串为"changsha"，ch 值为'a'，调用该函数后，p 所指串变为"chngsh"，即为结果字符串。

分析：为便于理解，首先假设用一片连续字节空间存放结果字符串，且用 q 指向这片空间的首字节，如图 4-17 中的初始状态。

p 指向待处理字符串的首字节。现用循环控制：逐个判断 p 指向的字符是否为要删除的字符，若不是，将 p 指向的字符复制到 q 指向的字节，并使 p、q 都增加 1 下移一个字节；若 p 指向的是要删除的字符（'a'）时不复制，此时指针 p 仍然增加 1 下移一个字节，而 q 不下移，如图 4-17 中的第 3 次循环。图 4-17 只表示了前 4 次循环，后面继续循环直到处理完整个字符

串。其实可将结果字符串仍存放在原待处理的字符串所占的空间中，因为复制后面的字符操作不影响前面已复制过来的字符，复制前面的字符操作也不影响后面还没被复制的字符。

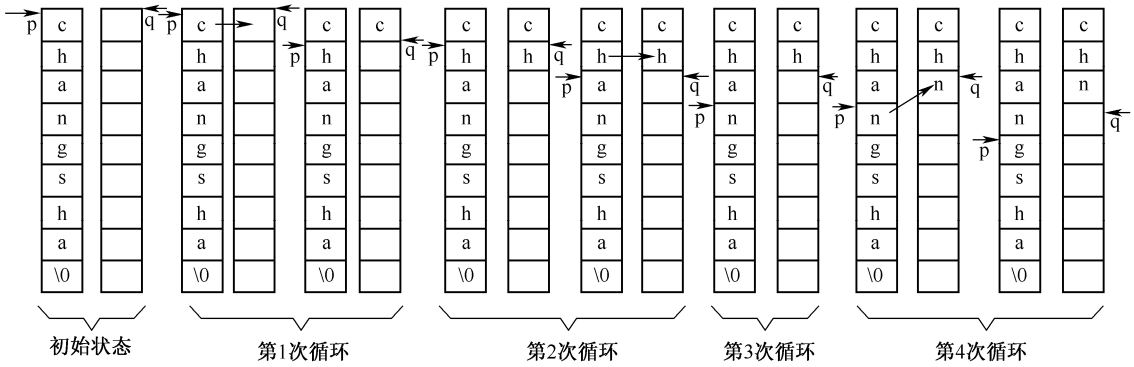

图 4-17　例 4.12 初始状态与前 4 次循环示意

程序如下：

```
//*****ex4_12.cpp*****
#include <iostream>
using namespace std;
void del_ch(char *p,char ch)        //定义函数
{   char *q=p;
    for(;*p!='\0';p++)
    if (*p!=ch)                     //若 p 指向的字符不是要删除的字符则复制
        {*q=*p; q++; }              //本复合语句等效于*q++=*p;
    *q='\0';                        //添加串结尾符
}
void main()
{   void del_ch(char *,char);
    char str[80],ch;
    cout<<"Input a string:\n";
    cin.getline(str, 80,'\n');      //当输入了 79 个字符或回车符，就表示输入完毕
    cout<<"Input the char deleted:\n";
    cin>>ch;
    del_ch(str,ch);
    cout<<"Then new string is:\n";
    cout<<str<<endl;
}
```

思考：（1）如何编写 void del_num_ch(char *p)删除 p 所指字符串中的数字字符。

（2）如何编写 void insert_ch(char *p,int n, char *q)在 p 所指字符串的第 n 字节位置插入 q 所指字符串。

4.5 指针数组与多级指针

4.5.1 指针数组

如果一个数组的元素是指针类型数据，则该数组称为指针数组。一维指针数组的定义形式为：

类型符 *数组名[常量表达式]；

其中，数组名之前的*表明定义指针数组，类型符表明数组元素所指对象的数据类型，即元素指针的基类型，方括号中常量表达式的值表示数组元素的个数。例如

int *p[10];

定义了一维指针数组 p，它有 10 个元素，每个元素都是指向 int 型变量的指针变量。和普通一维数组名一样，数组名 p 也代表第一个元素即 p[0]的地址。

注意，在*与数组名之外不能加圆括号，否则变成指向一维数组的行指针变量（前面已详析）。如：

int (*p)[10];

则定义指向含 10 个整型元素的一维数组的行指针变量。

类似地可定义多维指针数组。

指针数组适合表示一批指针数据。例如，对多个字符串进行排序，这些字符串可用一个字符型的二维数组存储，每行存储一个字符串。排序时一般要多次交换两个字符串在数组中的位置，这样较费时。如用指针数组，让其元素指向各串，就不必交换两个字符串的位置，而只交换指针数组中两个元素的指向。

【例 4.13】编写 void sort(char *p[],int n)，用选择法（在例 4.2 中已介绍）将指针数组 p 元素值按元素所指字符串大小从小到大排序，即让 p[0]指向最小的串，p[n-1]指向最大的串，而不改变字符串的存储位置。编写 main()等相关函数构成完整程序。要求从键盘输入各字符串，且输出排序结果。

分析：从要求可知，sort()开始执行时，若数组 p 中 6 个元素及其指向的字符串如图 4-18（a）所示，sort()执行后，6 个元素的指向发生改变，结果如图 4-18（b）所示。此处对数组 p 选择排序，每次确定 p 两元素值大小关系时，不是比较两元素本身，而是比较两元素指向的串。因此将例 4.2 中的数组元素比较运算改用 strcmp(p[k],p[j])即可。

程序如下：

```
//*****ex4_13.cpp*****
#include <iostream>
#include <string>
using namespace std;
#define N 6
void main()
{   void sort(char *[],int),write(char *[],int);
    char *name[N],str[N][30];
    int i;
    for(i=0;i<N;i++)
    {   name[i]=str[i];      //str[i]指向二维数组 str 的第 i 行首字节
```

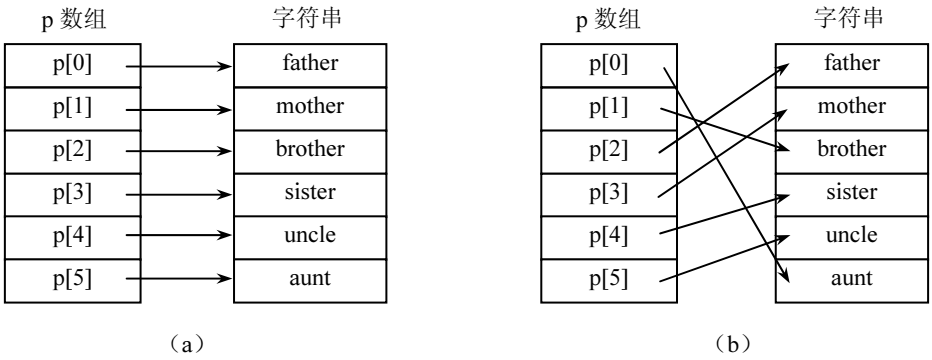

图 4-18　数组 p 及其指向的串

```
        cin.getline(str[i], 30,'\n'); }          //从键盘输入各字符串
        sort(name,N);
        write(name,N);
    }
void sort(char *p[],int n)                       //采用选择法排序
{   int i,j,k;
    char *t;
    for(i=0;i<n-1;i++)
    {   k=i;
        for(j=i+1;j<n;j++)
            if (strcmp(p[k],p[j])>0)             // p[j]指向的串小于 p[k]指向的串
                k=j;
            if (k!=i)
                { t=p[i]; p[i]=p[k];p[k]=t;}
    }
}
void write(char *p[],int n)                      //输出各串
{   int i;
    cout<<endl;
    for(i=0;i<n;i++) cout<<p[i]<<endl;
    cout<<endl;
}
```

对于上例而言，若待排序的各字符串在编写程序时就能确定，则也可用它们来初始化 name 数组如下：

```
char *name[]={"Russia","America","Japan","France","Britain","China"};
```

这样，name 的元素分别指向大括号中的各字符串。

4.5.2　多级指针

指针变量的地址仍是指针，叫多级指针。存放其他指针变量地址的变量叫多级指针变量，也简称多级指针。显然，多级指针指向某个指针变量。例如，指针变量 pp 指向一变量 ap，而 ap 又是指针变量，它指向另一个变量 a，则变量 pp 就是指向指针变量的指针，如图 4-19 所示。

定义二级指针变量的一般形式为

```
类型符　**变量名;
```

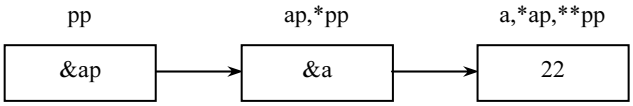

图 4-19　多级指针示意图

例如：

```
int **pp;
```

定义了指针变量 pp，它能指向另一个指针变量，该指针变量又能指向一个整型变量。pp 的前面有两个“*”号，由于指针运算符“*”是按自右向左顺序结合的，因此**p 相当于*(*p)，可以看出(*pp)是指针变量形式，它前面的“*”表示指针变量 pp 指向的又是一个指针变量，“int”表示后一个指针变量指向的是整型变量。

要定义图 4-19 所示的变量，可如下定义：

```
int a=22,*ap=&a,**pp=&ap;
```

ap 可用*pp 表示，a 可用*ap 和**pp 表示。

类似地可定义三级指针，例如：

```
char ***p;
```

编程时根据需要来确定指针变量的级数。

多级指针同样可以进行前面介绍的指针运算，只是指向关系多了一层而已，参加运算的两指针级别要一致。

指针数组元素的地址是多级指针，指针数组名代表首元素的地址，是多级指针常量。如同一维数组元素及其地址可以用指针变量表示（见表 4-1）一样，一维指针数组元素及其地址也可以用二级指针变量表示。前面 4.3.2 节说过，一维数组名和指针变量作函数形参时可以通用，类似地，一维指针数组名和二级指针变量作函数形参时也可通用，如例 4.13 中函数 sort()的形参 p 的说明 char *p[]也可写成 char **p。

4.5.3　带形参的 main 函数

前面介绍的程序中，main 函数都没有形参，其实，main 函数也可以带形参，用来接收来自命令行的实参。这个命令行是指运行 main 函数所在程序的命令。

带形参的 main()函数的一般形式是：

```
int main (int avgc, char *argv[] )   //   或  main (int avgc, char **argv)
{ … }
```

在操作系统命令状态下，启动、运行一个程序的命令行格式：

```
程序文件名   参数 1   参数 2    ……   参数 n
```

程序文件名和各参数之间用空格分隔。程序文件名、各参数都是字符串，统称为命令行参数。此处字符串可以不带双引号，若字符串本身含有空格，则要用双引号括起来。

程序中的 main 函数利用其形参接收命令行的实参信息。其中 argc 表示命令行中参数的个数，程序文件名字符串也算作其中一个，argv 是一个指向字符串的指针数组（或二级指针）。argv[0]指向命令行中第一个字符串，即程序文件名串，argv[1]指向命令行中第二个字符串，其余依次类推。

下面通过实例说明命令行参数是如何传递的。

【例 4.14】编写程序，要求输出运行该程序时所输入的命令行参数。源程序文件名为

ex4_14.cpp，经编译连接后生成的可执行程序为 ex4_14.exe。

```
//*****ex4_14.cpp*****
#include <iostream>
using namespace std;
int main(int argc,char *argv[])        //或  main(int argc,char **argv)
{   int k;
    cout<<"argc=" <<argc<<endl;
    for(k=0;k<argc;k++)
        cout<<"argv"<<k<<":"<<argv[k]<<'\n';
        cout<<"\n";
    return 0;
}
```

若运行该程序时输入的命令行是：

```
ex4_14   good   better   best↙
```

则输出结果：

```
argc=4
argv0：ex4_14
argv1：good
argv2：better
argv3：best
```

一旦命令行输入完毕，操作系统接收并分析命令行内容，辨别出其中字符串个数并将个数（4）传给形参 argc；将四个字符串：ex4_14、good、better、best 保存到特定位置，并将它们首地址分别传给字符指针数组元素 argv[0]、argv[1]、argv[2]、argv[3]，如图 4-20 所示。

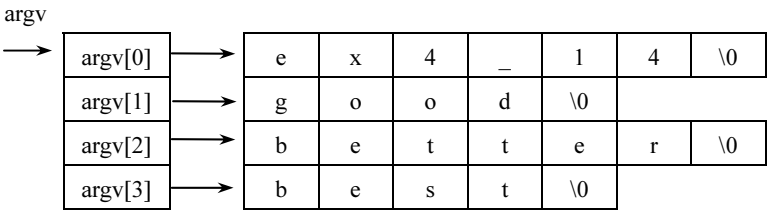

图 4-20　命令行参数指针数组示意图

4.6　引用

程序可以用变量名直接访问变量，也可以用指针变量间接访问变量，另外，C++中，程序还可以采用引用（reference）来访问变量。用引用作函数参数和返回值可以扩充函数传递数据的能力。

4.6.1　变量的引用

变量的引用就是变量的别名。一个引用总依附于某个实体，如变量、类对象（后面章节介绍），定义引用时必须进行初始化，说明是谁的引用，换句话说，总是为某个特定实体定义引用，例如：

```
int a;
int &ar=a;
```

为变量 a 定义了一个引用 ar，ar 也表示 a 的存储单元，此处&是引用声明符，不是取地址运算符，&前面的 int 说明 ar 是整型变量的引用，即被引用的变量应该是整型变量。

可以在定义变量的同时为它定义引用，上面两行也可合并为：

```
int a, &ar=a;          //还可同时初始化 ar 引用的变量：int a, &ar=a=100;
```

后续程序中变量 a 都可用它的引用 ar 替换，例如：

```
int *p=&ar;            //&ar 代表&a，定义指向 a 的指针变量
int &ar2=ar;           //为引用 ar 定义引用 ar2，因 ar 代表 a，相当于为 a 再定义另一引用 ar2
ar=100;                //等效于 a=100;
cin>>ar;               //等效于 cin>>a;
cout<<ar;              //等效于 cout<<a;
```

指针变量的引用定义格式如下：

```
int *q, *&qr=q;        //还可同时初始化 qr 引用的指针变量：int a, *q, *&qr=q=&a;
```

此后 qr 就可作 q 用。

前面已介绍，变量或存储单元的间接访问形式为“*指针”，也可以为间接访问形式的变量或存储单元定义引用，例如：

```
int   a, *p=&a, &ar=*p;    //定义 ar 是*p 的引用，而*p 是 a 的间接访问形式，故 ar 也就是 a 的引用
```

注意，不能定义引用数组，也不能定义指向引用的指针，下面的引用定义有语法错误。

```
int &refer[10], &*pr ;
```

变量引用的作用主要体现在作函数参数和返回值的情况。

4.6.2 引用作函数参数

前面多次提到，在函数调用时，实参的值传递给形参变量，函数在执行过程中改变了形参的值，但对应的实参不会被改变，这就是所谓的函数参数值单向传递现象。为了让被调用函数可以改变主调函数中变量的值，4.2.4 节介绍了用指针作函数参数的方法，此时实参向形参传递指针值，被调函数通过形参变量间接访问主调函数中的变量。若将函数形参说明为变量的引用（或引用型变量），也能达到修改实参变量的目的，而且编程更简洁。对于例 4.7 而言，将 swap 函数修改如下：

```
void swap(int &r1,int &r2)    //说明形参为变量的引用
{   int t;
    t=r1;
    r1=r2;
    r2=t;
}
```

相应的函数调用语句改为：

```
swap(a,b);
```

定义 swap()时说明形参为变量的引用，但没指定是哪一变量的别名。调用函数时，系统把实参 a、b 变量名（非值）传递给形参 r1、r2，r1、r2 分别变为 a、b 的引用（别名），即 r1、r2 也分别表示实参 a、b 对应的存储单元，如图 4-21 所示；此时，swap()修改形参 r1、r2 就是修改实参 a、b。此处本质上也是传递地址，与指针变量作形参不同的是，此时形参不需占用临时存储空间，而是直接引用 a、b，用引用作函数参数显得表达简单、自然。

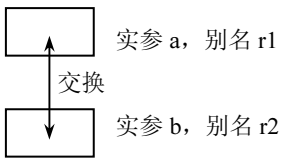

图 4-21 交换两引用型形参的值

与引用型形参对应的实参可以是变量，如上面调用 swap(a,b)中的实参 a、b 是已定义的变量；另外，实参也可以是指针所指向的变量或（动态）存储单元，例如，假设指针 x、y 已经指向内存单元，则调用 swap(*x,*y)将交换 x、y 所指内存单元的值。

4.6.3　引用作函数返回值

函数也可返回变量（或存储单元）的引用，其定义一般格式是：

```
类型名　& 函数名(形参表)
{…
 …
return 变量、变量的引用或用“*指针”形式表示的存储单元;}
```

return 后的变量不能是本函数中的局部变量，因为函数执行完毕返回主调函数后，函数中的局部变量全部释放，主调函数再用其引用去访问它，不能保证结果正确。例如，下面的函数在编译时会有警告提示：

```
int & minp(int &x,int &y)
{int q;
q=x<y?x:y;
return q;
}
```

因为该函数试图返回其局部变量 q 的引用。return 后的变量应该是主调函数可寻址或可见的变量或内存单元，如全局变量。该函数可改为

```
int & minp(int &x,int &y)
{ return x<y?x:y;}
```

该 minp()被调用时，x、y 变成实参变量的引用，显然，主调函数可找到实参变量。

一、选择题

1．设有 int x[][3]={ {0},{1,2},{3,4,6},{5}};　则 x[1][1]的值是（　　）。

A．3　　B．2　　C．6　　D．4

2．在下面的二维数组定义中，正确的是（　　）。

A．int a[3][];　　B．int a[][3];

C．int a(3)(3);　　D．int a[][3]={{1,2},{3}};

3．下面哪个定义或语句序列能使 p 指向 a？（　　）

A．int a,*p=a;　　B．int a,*p;　*p=a;

C．int a,*p;　*p=&a;　　D．int a,*p=&a;

4．若有：double a[10], *pa=a;，要将 10 赋值给 a 中的下标为 5 的元素，不正确的语句是（　　）。

A．pa[5]=10;　　B．*(pa+5)=10;

C．*(a[0]+5)=10;　　D．a[5]=10;

5．设有 int a[10], *p=&a[4] ; 则下面哪种表示与 a[9]不等价？（　　）

A．*(a+9)　　B．*(p+5)　　C．p[5]　　D．p+5

6．设有 int a[10], *p=a+5 ; 则下面哪种表示与 a[3]不等价？（　　）

A．*(a+9)　　B．*(p-2)　　C．p[-2]　　D．p+2

7．设有 int a[20], *p=a; 则下面哪个与 a[1]不等价？（　　）

A．p[1];　　B．*++p;　　C．*(a+1);　　D．*++a;

8．已知 char a[][20]={"hunan","jiangxi","shandong"};，语句 cout<<a[3];得到的输出是（　　）。

A．a　　B．shandong　　C．输出结果不确定　　D．数组定义有错

9．若函数形参是数组，则对应的实参（　　）。

A．只能是数组名　　B．只能是指针

C．任何类型的数据　　D．可以是数组名或指针

10．若函数形参是指针变量，则对应的实参（　　）。

A．只能是指针　　B．只能是数组名

C．任何类型的数据　　D.可以是指针或数组名

11．若实参是数组名，则实际传递给形参的是（　　）。

A．数组的第一个元素值;　　B．数组元素的个数

C．数组的全部元素值;　　D．数组首元素地址

二、填空题

1．设有 int a[20]; 则数组名 a 代表元素_____的地址。

2．C/C++编译系统用_____代表函数的入口地址。

3．若有 char *a="abcdef";，则 a[1]的值为_____。

4．int *f(int, float);说明 f 为_____，而 int (*f)(int,float);定义 f 为_____。

5．int (*p)[4];定义 p 为_____，而 int *p[4];定义 p 为_____。

三、程序阅读题，写出结果

1．
```
#include <iostream>
using namespace std;
int sum(int x[],int n)
{int i,s=0;
for(i=0;i<n;i++) s+=x[i];
return s;}
int main()
{int a[5],i;
 int s1,s2;
 for(i=0;i<5;i++) a[i]=i;
 s1=sum(a,5);
 s2=sum(a+2,3);
 cout<<s1<<"，"<<s2<<endl;
 return  0;
}
```

输出：_____，_____

2．
```
#include <iostream>
#include <iomanip>
using namespace std;
int table[5]={5,4,3,2,7 };
void f (int a[], int n, int *p)
{ int k, j;
   j=0;
```

```
    for(k=1;k<n;k++)
    if(a[k]<a[j])    j= k;
    *p =a[j];
}
int main()
{int a;
f(table ,5,&a);
cout<<a<<endl;
return 0;
}
```

输出结果：_____

3．

```
#include <iostream>
using namespace std;
char * f( char *p1,char *p2 )
{ char *t=p1;
  while(*p1!='\0') p1++;
  while(*p2!='\0')
  {*p1=*p2;
   p1++; p2++;}
   *p1='\0';
   return t;
}
int main()
{char s1[100]="ABC",*s2="A";
  cout<<'\n'<< f(s1,s2)<<", "<<s1<<'\n';
  return 0;
}
```

输出：_____，_____

四、完善程序填空题

1．函数 int lookup(int x[], int n,int y)在 x[0]…x[n-1]中查找是否有等于 y 的元素，若有，返回第一个相等元素的下标，否则，返回-1，请将函数补充完整。

```
int lookup(int x[], int n, int y)
{int flag=-1;
int i;
for(i=___①___ ;i<n ; i++)
if(x[i]==y)
    {flag=i;
       break;
    }
return ___②___ ;
}
```

2．下面的函数用来判断 s 指向的字符串是否为“回文串”，即从前向后或从后向前读是一样的（默认长度为 0、1 的字符串是“回文串”），若是返回 1，否则返回 0。

```
int f(char *s)
{   int result=1;
    char *p=s;
```

```
        while(*p)  ____①____;
        p--;
        while(____②____)
            {if( *s!=*p) { result=0;   break ; }
            ____③____ ; p--; }
         return result;
    }
```

3．已知费氏（Fibonacci）数列通项：

$F_0=F_1=1$　　（i=0，1）

$F_i=F_{i-1}+F_{i-2}$　（i>1）

用数组求 Fibonacci 数列前 20 项。

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{   int i, f[20];
    f[0]= f[1]=____①____ ;
    for(i=2;i<20;i++)
        f[i]=f[i-1]+____②____;
    cout<<" Fibonacci :"<<endl;
    for(i=0;i<20;i++)
      {if(i%5==0)   cout<<endl;
        cout<<setw(6)<<f[i];
      }
    return 0;
}
```

五、程序设计题

1．从键盘输入 10 个整数存于数组中，然后按从大到小的次序输出。请编程实现。

2．从键盘输入 100 个整数存入数组 np 中，其中凡相同的数在 np 中只存入第一次出现的数，其余的都被剔除。

3．找出一个二维数组中的鞍点，即该位置上的元素在该行上最大，在该列上最小。也可能没有鞍点。

4．编写函数 void f(int a[],int n)，将 a[]中的 n 个元素按逆序重新存放，例如，原来存放顺序为：8,6,5,4,1。要求改为：1,4,5,6,8。

5．编写函数 int f(char * s)，判断 s 所指的串是否为“回文串”，即前后对称的串，如：“a131a”、“a1bb1a”，若是返回 1，否则返回 0。

6．将字符数组 str1 中下标为双号的元素值赋给另一字符数组 str2，并输出 str1 和 str2 的内容。