第5章 函　　数

5.1 引例与概述

5.1.1 引例

例 5.1.1　处理并输出

先输入整数 n（n<100），然后再输入 n 个整数。请完成以下任务。

（1）输出这些整数。

（2）把这些整数逆置后输出。

（3）把这些整数升序排列并输出。

输出时，每两个数据之间留一个空格。

输入格式：

测试数据有多组，处理到文件尾。每组测试输入两行，第一行输入 n（1<n<100），第二行输入 n 个整数。

输出格式：

对于每组测试，输出三行，第一行直接输出所输入的 n 个整数，第二行输出逆置后的 n 个整数，第三行输出升序排列后的 n 个整数。每行的每两个数据之间留一个空格。

输入样例：

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

输出样例：

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

本题宜用数组处理。而题中 3 个任务每个都要求输出，可把输出的代码重复使用 3 次，但这样的代码较冗长，编码效率较低。我们应勤学善思，提高效率意识和规范意识。当一段代码需要重复多次使用时，通常会考虑能否把这段代码独立出来作为一个整体，这就需要用到自定义函数。具体代码如下：

```
#include<bits/stdc++.h>          //万能头文件
using namespace std;
const int N=100;
```

```
void prtArray(int a[], int n) {//函数定义，输出数组元素的函数
    for(int i=0; i<n; i++) {
        if(i!=0) cout<<" ";
        cout<<a[i];
    }
    cout<<endl;
}
int main() {
    int a[N],n,j,k;
    while(cin>>n) {
        for(j=0; j<n; j++) cin>>a[j];
        prtArray(a, n);          //第 1 次调用自定义函数
        for(k=0; k<n/2; k++) {
            swap(a[k],a[n-1-k]);
        }
        prtArray(a, n);          //第 2 次调用自定义函数
        sort(a, a+n);            //调用系统函数，排序区间[&a[0], &a[n-1]+1)
        prtArray(a, n);          //第 3 次调用自定义函数
    }
    return 0;
}
```

这里定义了一个自定义函数 prtArray，用于输出包含 *n* 个元素的一维数组，数据之间留一个空格，然后调用该函数（函数必须被调用后才有效果）3 次完成 3 个任务中的输出。这种代码实现显然更加简洁。实际上，一些常用功能即使在一个程序中不被调用多次，也经常写成一个一个自定义函数。例如，判断一个数是否是素数，求两个整数的最大公约数或最小公倍数、二分查找及排序等。

另外，这个程序用了 C++的万能头文件“bits/stdc++.h”，包含这个头文件相当于包含了 C++所有的头文件，省去需包含各种头文件的麻烦。例如，本题中使用了系统函数 sort，本来应该包含头文件 algorithm，但有了万能头文件就不用再写该头文件了。使用万能头文件的 C++程序只需要使用以下两句，而不用再包含其他头文件：

```
#include<bits/stdc++.h>          //万能头文件
using namespace std;             //引入 std 命名空间
```

需要注意的是，虽然 Dev-C++编译环境和目前很多高校的 OJ 都支持万能头文件，但也有些 OJ 和编译环境（如 VC6、VC2010 等）不支持万能头文件，建议使用新接触的 IDE 时或在线做题及程序设计竞赛之前先做测试。通用起见，本书代码一般不使用万能头文件，读者写代码时可自行决定是否使用万能头文件。

5.1.2　概述

简言之，函数是一组相关语句组织在一起所构成的整体，并以函数名标注。

从用户的角度而言，函数分为库函数（系统函数）和用户自定义函数。库函数有很多，例如，在 math.h 中的 sqrt、fabs、pow、ceil、floor、round 等，调用示例如下：

```
sqrt(9.0)                        //得到 3.0，求平方根
fabs(-3.5)                       //得到 3.5，求实数的绝对值
```

```
pow(2,3)                          //得到8.0，求幂
ceil(3.1)                         //得到4.0，即不小于参数的最小整数
floor(3.9)                        //得到3.0，即不大于参数的最大整数
round(3.56)                       //得到4.0，即对参数四舍五入取整
```

因以上函数的返回类型都是 double，故结果都表示为包含小数点的实数。规范起见，建议这些函数的参数也用 double 类型。

又如，在 stdlib.h 中的 abs、srand、rand、malloc、free 等，调用示例如下：

```
abs(-1)                           //得到1，整数的绝对值
srand(time(NULL));                //设置随机数生成器的种子,time需包含头文件time.h
int a=rand();                     //产生0~RAND_MAX（32767）之间的一个随机整数
int b=20+rand()%(80-20+1);        //产生20~80之间的一个随机整数
int *a=(int *) malloc(5*sizeof(int));       //申请5个int类型元素的动态数组
free(a);                                    //释放用malloc申请的动态数组
```

注意，库函数使用时须包含相应头文件。另外，在 Dev-C++下也能调用 max（求两者中的大者）、min（求两者中的小者）、stoi（将数字字符串转换为整数）、to_string（将数值转换为字符串）等系统函数。其中，后两者是 C++ 11 标准下的函数，使用前需添加编译命令“−std=c++11”（设置路径详见例 4.6.4）。读者可以自行查阅并测试感兴趣的系统函数。

本章主要介绍用户自定义函数。

一个大的程序一般分为若干个程序模块，每个模块用来实现一个特定的功能，每个模块一般写一个函数定义来实现（需调用）。

函数是 C/C++程序的基本构成单位，一个 C/C++程序由一个主函数 main 及若干其他函数构成。C/C++程序从 main 函数中开始执行，在 main 函数中结束。

函数的作用是通过函数调用实现的。操作系统调用主函数，主函数调用其他函数，其他函数可以调用主函数之外的其他函数。同一函数可以被一个或几个函数调用任意次。

如图 5-1 所示，是一个函数的调用示意图。

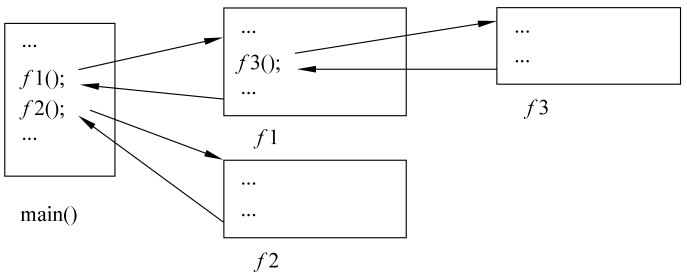

图 5-1　函数的调用示意图

由图 5-1 可见，main 函数调用 $f1$、$f2$ 函数，$f1$ 函数调用 $f3$ 函数，$f3$ 函数调用结束返回 $f1$ 函数中的调用点，$f1$ 函数调用结束返回到 main 函数中的调用点，$f2$ 函数调用结束返回到 main 函数中的调用点。

5.2　函数基本用法

5.2.1　函数的定义

函数定义由函数头和函数体两部分组成。一般形式如下：

```
类型 函数名([形参列表]) {
    函数体
}
```

说明：

（1）“函数类型函数名([形参列表])”是函数头，{}中的是函数体。

（2）函数类型（也称返回类型）可以是各种基本数据类型、指针类型、结构体类型、void（空类型，明确指定函数不返回值）等。C 语言的函数默认返回类型为 int，但 Dev-C++、VC2010 等编译环境都不支持默认返回类型。建议明确指定函数的返回类型。

（3）函数名必须是合法的标识符。

（4）函数定义中的参数为形式参数，简称形参。根据是否有形参，函数可分为带参函数和无参函数。形参列表的每个参数包括参数类型和参数名，形参列表若有多个参数，则以逗号分开。C++支持参数带默认值，默认值参数须放在最右侧。

（5）根据是否有返回值，函数可分为有返回值函数和无返回值函数。通过函数中的 return 语句返回函数的返回值。return 语句的一般格式如下：

```
return [返回值表达式];
```

其中，返回值表达式的类型一般应与返回类型一致，否则以返回类型为准。语句“return;”控制程序流程返回到调用点；若 return 语句后带返回值表达式，则在控制程序流程返回调用点的同时带回一个值。

下面给出几个函数定义的例子：

```
void print() {                          //无参函数，也没有返回值，返回类型用 void
   cout<<"hello\n"<<endl;
}
int max(int a, int b) {                 //求两个整数的最大值
   return a>=b?a:b;
}
//重载函数：C++中参数不同的若干同名函数
string max(string a, string b) {        //求两个字符串的最大值
   return a>=b?a:b;
}
//C++默认值参数的函数，带默认值的参数处于最右侧
int max3nums(int a, int b=2, int c=3){ //求三个整数的最大值
   int t=max(a, b);                     //嵌套调用 max(int, int)
   return max(t, c);
}
```

5.2.2　函数的声明

若函数定义在函数调用之前，则定义时的函数头可以充当函数声明（或称函数原型）；

否则函数调用之前必须先进行函数声明，形式如下：

函数类型 函数名([形参列表]);

即以函数定义时的函数头加分号表示函数声明，其中，形参列表中的形参名可以省略。

例如 5.2.1 节定义的 max 函数声明如下：

```
int max(int a, int b);
string max(string a, string b);
```

或

```
int max(int, int);                          //省略形参名
string max(string, string);                 //省略形参名
```

5.2.3 函数的调用

函数调用的形式一般如下：

[变量=]函数名([实际参数表])[;]

void 返回类型的函数只能以语句形式调用，其他返回类型的函数一般以表达式形式调用，否则其返回值没有意义。

调用时的参数称为实际参数，简称实参，一般不需要指定数据类型，除非是进行强制类型转换。参数的类型、顺序、个数必须与函数定义中的一致，但带默认值参数的函数调用时实参个数可以与形参个数不一致。

函数调用时，把实参依序传递给形参，然后执行函数定义体中的语句，执行到 return 语句或函数结束时，程序流程返回到调用点。

例如，调用 5.2.1 节定义的函数的方法如下：

```
print();                                    //void 返回类型的函数以语句形式调用
int t=max(123,99);                          //有返回值的函数一般以表达式形式调用
cout<<max(1, 2)<<endl;                      //调用 max(int, int)
cout<<max("abc", "cdfg")<<endl;             //调用 max(string, string)
cout<<max3nums(1)                           //实参为 1,2,3，后两个参数使用默认值
    <<" "<<max3nums(4,5)                    //实参为 4,5,3，后一个参数使用默认值
    <<" "<<max3nums(5,3,4)<<endl;           //实参为 5,3,4，不使用默认值
```

max3nums 函数共有 3 个参数，其中 2 个带默认值，调用时可提供 1、2、3 个实参。

函数调用也可用变量作为实参，主调函数中的实参和被调函数中形参可以同名，但它们实际上是局限于各自所在函数的不同变量。

在 OJ 做题或程序设计竞赛时，题目通常有多组测试数据，可以把一组测试的代码单独作为一个函数处理，然后在循环中调用该函数完成多组测试。

5.3 函数举例

例 5.3.1 逆序数的逆序和

输入两个正整数，先将它们分别倒过来，然后再相加，最后再将结果倒过来输出。注意：前置的零将被忽略。例如，输入 305 和 794，倒过来相加得到 1000，输出时只要输出 1

就可以了。

因为求逆序数的方法是一样的，可以编写一个求逆序数的函数，调用 3 次即可完成 2 个输入的整数及 1 个结果整数的逆置。

思考：当 x=1234，如何得到 x 的逆序数？

设 r 为 x 的逆序数，可以这样考虑：r=((4×10+3)×10+2)×10+1=4321，即让 r 一开始为 0，再不断把 x 的个位取出来加上 r×10 重新赋值给 r，直到 x 为 0（通过 x=x/10 不断去掉个位数）。具体代码如下：

```
#include<iostream>
using namespace std;
int reverseNum(int x) {          //构成逆序数的函数，x 是正整数
    int r=0;
    while(x>0) {
        r=r*10+x%10;             //移位后右边加上 x 的个位数
        x=x/10;
    }
    return r;
}
int main() {
    int a,b;
    cin>>a>>b;
    int c=reverseNum(a)+reverseNum(b);
    cout<<reverseNum(c)<<endl;
    return 0;
}
```

本例所写的 reverseNum 能够忽略前导 0，原因请读者自行分析。

例 5.3.2　素数判定函数

输入一个正整数 n，判断 n 是否是素数，是则输出 yes，否则输出 no。要求写一个判断一个正整数是否是素数的函数。

关于 n 是否是素数，已知可以从 2 至 $\sqrt{n}$ 判断是否有 n 的因子，若有则不是素数。这里只要把相关代码作为一个整体写成一个函数。因为结果只有两种可能（是或否），所以返回类型设为 bool（只有 true、false 两个值）；而 n 是要被判断的数，因此需要一个整型参数。具体代码如下：

```
#include<iostream>
#include<cmath>                  //系统函数 sqrt 求开方数须包含此头文件
using namespace std;
bool isPrime(int n) {            //判断 n 是否是素数,若是则返回 true,否则返回 false
    bool flag=true;              //一开始假设是素数，标记变量初值设为 true
    double limit=sqrt(n);
    for(int i=2; i<=limit; i++) {
        if(n%i==0) {             //若有因子，则可以判断 n 不是素数
            flag=false;
            break;
        }
    }
```

```
    if(n==1) flag=false;          //对1特判
    return flag;
}
int main() {
    int n;
    cin>>n;
    if(isPrime(n)==true)
        cout<<"yes"<<endl;
    else
        cout<<"no"<<endl;
    return 0;
}
```

例 5.3.3　最小回文数

输入整数 n，输出比该数大的最小回文数。其中，回文数指的是正读、反读一样的数，如 131、1221 等。要求写一个判断一个整数是否是回文数的函数。

判断是否是回文数可以调用例 5.3.1 中的求逆序数的函数 reverseNum，判断该数与逆序数是否相等。因为要找比 n 大的最小回文数，可以从 n+1 开始逐个尝试是否满足逆序数等于本身的条件，第一个满足条件的数即为结果。具体代码如下：

```
#include<iostream>
using namespace std;
int main() {
    bool isSymmetric(int);        //定义在调用之后，须在调用前先声明
    int n;
    cin>>n;
    while(true) {
        n++;
        if(isSymmetric(n)==true) break;
    }
    cout<<n<<endl;
    return 0;
}
int reverseNum(int x) {           //构成逆序数的函数，x 是正整数
    int r=0;
    while(x>0) {
        r=r*10+x%10;              //移位后右边加上 x 的个位数
        x=x/10;
    }
    return r;
}
bool isSymmetric(int n) {         //判断 n 是否是回文数,若是返回 true,否则返回 false
    if(n==reverseNum(n))          //回文数的判断
        return true;
    else
        return false;
}
```

实际上，函数 isSymmetric 可以简写为如下：

```
bool isSymmetric(int n) {
    return n==reverseNum(n);
}
```

例 5.3.4　大整数加法

输入两个大正整数（长度可能达到 1000 位），求两者之和。

两个大正整数加法的基本思路：两个大正整数作为字符串（用 string 类型变量）处理，加法根据“右对齐、逐位相加”的方法，关键在于右对齐相加及进位处理。其中，右对齐相加可以在把两个字符串逆置后从第一个字符开始相加。字符串的逆置可以写一个以字符串变量为形参的函数，需要注意的是，string 类型形参的变化不会影响实参，因此通过返回值返回变化后的结果（实际上把 string 类型变量作为引用参数来返回结果更简单，读者可以在掌握引用之后自行修改）。在做加法时，拟用第一个字符串存放最终结果，因此需要保证其长度不小于第二个字符串，方法是判断两个字符串的长度，若前一个字符串短，则调用系统函数 swap 交换两个字符串。进位处理方面，可以用一个整型变量表示，其初值一开始设为 0，在加法计算过程中把其加到和中，并不断更新为最新的进位。具体代码如下：

```
#include<iostream>
#include<string>
using namespace std;
string reverse(string s) {                  //逆置字符串
    int n=s.size(), mid=n/2;
    for(int i=0; i<mid; i++) {              //以中间为界，两端字符交换
        swap(s[i],s[n-1-i]);
    }
    return s;
}
string bigAdd(string s, string t) {
    if(s.size()<t.size()) swap(s,t);        //若 s 短于 t，则交换
    s=reverse(s);                           //逆置 s
    t=reverse(t);                           //逆置 t
    int carry=0;                            //进位
    for(int i=0; i<s.size(); i++) {
        carry+=s[i]-'0';                    //把 s[i]转换为整数加到 carry 中
        if(i<t.size())                      //若第二个字符串还没有结束
            carry+=t[i]-'0';                //则把 t[i]转换为整数加到 carry 中
        s[i]=carry%10+'0';                  //余数转换为数字字符存放在 s[i]中
        carry/=10;                          //保存新的进位
    }
    s=reverse(s);                           //结果逆置
    if(carry>0) s="1"+s;                    //最后的进位
    return s;
}
int main() {
    string a,b;
    cin>>a>>b;
    cout<<bigAdd(a,b)<<endl;
    return 0;
}
```

实际上，逆置字符串也可以调用 algorithm 头文件中的 reverse 函数实现。例如，逆置字符串 *s* 的代码如下：

```
reverse(s.begin(),s.end());
```

其中，两个参数对应的逆置区间为[*s*.begin(), *s*.end())，即此调用语句将逆置整个字符串 *s*。

5.4 数组作为函数参数

5.4.1 数组元素作为实参

数组元素也称下标变量，因此数组元素作为函数实参时，与普通变量作为函数实参是一致的：单向值传递，即只能把实参的值传递给形参，而不能再把形参的值传回给实参。

例 5.4.1 数组元素作为实参

在一维数组 *a* 中存放 10 个整数，请输出它们的立方数。要求定义一个求立方数的函数。具体代码如下：

```
#include<iostream>
using namespace std;
int cubic(int  n) {                          //自定义求立方数的函数
   return n*n*n;
}
int main() {
   int a[10], i, j;
   for(i=0; i<10; i++) cin>>a[i];
   for(i=0; i<10; i++) {
      cout<<cubic(a[i])<<endl;               //数组元素作为实参
   }
   return 0;
}
```

5.4.2 数组名作为函数参数

本小节讨论数组名作为函数的参数，即形参和实参都使用数组名。此时传递的是数组的首地址（数组名代表数组的首地址），即传地址。实际上，数组名作函数参数时参数传递依然是单向的，即由实参传递给形参，但由于在函数调用期间，形参数组与实参数组同占一段连续的存储单元，因此对形参数组的改变就是对实参数组的改变，即从效果上看，达到了双向传递的效果。

例 5.4.2 *m* 趟选择排序

输入 *n* 个整数构成的数列，要求利用选择排序进行排序，并输出第 *m* 趟排序后的数列状况。请把选择排序定义为一个函数。

选择排序的思想和方法在前面的章节中已经讨论过，这里以函数的形式表达。具体代码如下：

```
#include<iostream>
using namespace std;
```

```
const int N=100;
//n 个数，进行 m 趟排序
void selectSort(int a[], int n, int m) {
    for(int i=0; i<m; i++) {            //控制 0~m-1 共 m 趟排序
        int k=i;
        for(int j=i+1; j<n; j++) {
            if(a[k]>a[j]) k=j;
        }
        if(k!=i) swap(a[k],a[i]);       //直接调用系统函数 swap 交换
    }
}
void prt(int a[], int n) {              //输出 n 个数据，每两个数据之间一个空格
    for(int i=0; i<n; i++) {
        if(i>0) cout<<" ";
        cout<<a[i];
    }
    cout<<endl;
}
int main() {
    int b[N], n, k;
    cin>>n>>k;
    for(int i=0; i<n; i++) cin>>b[i];
    selectSort(b, n, k);                //第一个参数是数组名作为函数的实参
    prt(b, n);                          //第一个参数是数组名作为函数的实参
    return 0;
}
```

运行结果：

```
6 3↵
3 5 1 2 8 6↵
1 2 3 5 8 6
```

从运行结果可见，实参数组 *b* 在调用 selectSort 函数之后发生了改变，即形参数组 *a* 的改变影响到了实参数组 *b*。因为数组名作为函数参数时，是将实参数组的首地址传给形参数组，所以，在函数调用期间 *a*[*i*]和 *b*[*i*]（0≤*i*<*n*）同占一个存储单元，则对 *a*[*i*]的改变就是对 *b*[*i*]的改变。

注意：

（1）用数组名作为函数参数，应该在主调用函数和被调用函数中分别定义数组，本例中 *a* 是形参数组，*b* 是实参数组，分别在其所在函数中定义，不能只在一方定义。

（2）实参数组与形参数组类型应一致（本例都为 int 类型），如不一致，将出错。

（3）string 类型形参或者 vector 形参的改变不会影响实参，除非使用引用参数。

5.5　引用

通俗地说，引用是对象（可以是变量、符号常量等）的“别名”。定义引用变量的格式如下：