# 第5章 函　数

## 5.1 引例与概述

### 5.1.1 引例


例 5.1.1

**例 5.1.1　处理并输出(HLOJ 1921)**

**Problem Description**

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。

| Sample Input | Sample Output |
| --- | --- |
| 5<br>3 2 1 5 4 | 3 2 1 5 4<br>4 5 1 2 3<br>1 2 3 4 5 |

显然,本题宜用数组处理。而本题的每个任务都要求输出,可以把输出的代码重复使用三次,但这样的代码比较冗长。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。本题具体代码如下。

```
#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;
    cin>>n;
    for(j=0; j<n; j++) cin>>a[j];
```

```
    prtArray(a, n);                          //调用自定义函数
    for( k = 0; k < n/2; k++) {
        swap(a[k],a[n - 1 - k]);
    }
    prtArray(a, n);                          //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);                          //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(函数必须被调用了才有效果)三次完成三个任务中的输出。这个代码显然更加简洁。实际上,一些常用功能即使在一个程序中不被调用多次,也经常写成一个个自定义函数,例如,判断一个数是否素数,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 C++所有的头文件,简化了需要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,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(例如 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 fabs、pow、ceil、floor 等,调用示例如下。

```
fabs(-3.5)得到 3.5,求实数的绝对值。
pow(2.0, 3.0)得到 8.0,求幂次方。
ceil(3.78) 得到 4,求不小于参数的最小整数。
floor(3.78) 得到 3,求不大于参数的最大整数。
```

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

```
srand(time(NULL));                           //随机种子初始化,time 需包含头文件 time.h
int a = rand();                              //产生 0～RAND_MAX 的一个随机整数
int b = 20 + rand() % (80 - 20 + 1);         //产生 20～80 的一个随机整数
```

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、min、round 等系统函数。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及若干个其他函数构成。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。

下面给出一个函数的调用示意图。

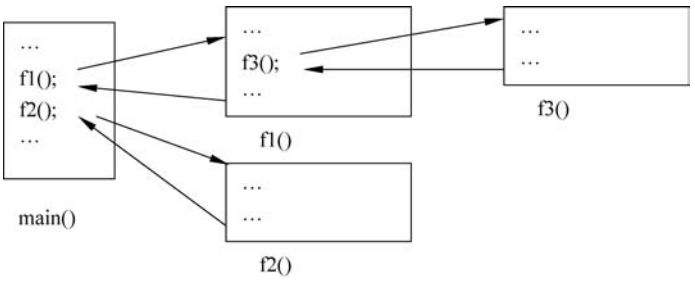


图5-1 函数的调用示意图

由图5-1可见,main函数调用f1、f2函数,f1函数调用f3函数,f3函数调用结束返回到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部分。

(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边位置
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

函数的调用形式一般如下。

**［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)
//带默认值的函数调用可以指定或不指定默认值
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
```

函数调用也可用变量作为实参,实参和形参可以同名,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

**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,调用三次即可完成两个输入的整数及一个结果整数的逆置。

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:$r=((4\times10+3)\times10+2)\times10+1=4321$,即让 $r$ 一开始为 0,再不断把 $x$ 的个位取出来加上 $r\times10$ 重新赋值给 $r$,直到 $x$ 为 0(通过 $x=x/10$ 不断去掉个数)。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**


例 5.3.2

输入一个正整数 $n$，判断 $n$ 是否素数，是则输出“yes”，否则输出“no”（引号不必输出）。要求写一个判断一个正整数是否素数的函数。

关于 $n$ 是否素数，已知可以从 2 至$\sqrt{n}$判断是否有 $n$ 的因子，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 是否为素数，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(1.0 * n);         //参数最好为 double 类型
    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

**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回文数的函数。

判断是否回文数可以调用例 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。

```
#include <iostream>
```

```
using namespace std;
int main() {
    bool isSymmetric(int);                    //定义在调用之后,则须在调用前先声明
    int T;
    cin >> T;
    for(int i = 0; i < T; i++) {
        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 是否为回文数,是返回 true,否返回 false
    if(n == reverseNum(n))                    //回文数的判断
        return true;
    else
        return false;
}
```

实际上,函数 isSymmetric 可以简写如下。

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

例 5.3.4

**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。

两个大正整数的加法的基本思路: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到实参,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,拟用第一个串存放最终结果,因此需要保证其长度不小于第二个串,方法是判断两个串的长度,若前一个串短,则调用系统函数 swap 交换两个串。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;                     //最后的进位处理
    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 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 数组作函数参数

### 5.4.1 数组元素作实参

数组元素也称下标变量,因此数组元素作函数实参时,与普通变量作函数实参是一致的: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。


例 5.4.1

```
#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;         //数组元素作实参
    }
    return 0;
}
```

## 5.4.2　数组名作函数参数

数组名作函数的参数,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,所以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

**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，因此，在函数调用期间 $a[i]$和 $b[i]$($0 \leqslant i < n$)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。

**类型 & 别名变量＝变量；**

其中，“&”与数据类型一起时是引用定义符，而不是地址符或按位与运算符。

例如：

```
int a = 123;
int& ra = a;                                //ra 为 a 的别名，对 ra 的操作就是对 a 的操作
const int N = 100;
const int &rN = N;                          //rN 为符号常量 N 的别名
```

引用是对象的别名，而一旦一个别名给了一个对象，此别名就不能用作其他对象的别名，所以引用定义时必须进行初始化。

对于有了别名的对象，不管使用原名还是别名进行操作，都是对该对象的操作。

因为引用形式的形参变量(简称引用形参)是实参变量(简称实参)的别名,对引用形参的改变就是对实参的改变,所以可以通过引用形参来返回值,而且可以通过使用多个引用形参达到返回多个值的目的。

**例 5.5.1　交换函数**

设计一个函数,实现交换两个整型变量的值。


例 5.5.1

```
#include<iostream>
using namespace std;
void mySwap(int &a, int &b) {
    int t=a;
    a=b,b=t;
}
int main() {
    int m, n;
    cin>>m>>n;
    mySwap(m, n);
    cout<<m<<" "<<n<<endl;
    return 0;
}
```

运行结果:

```
1 3↵
3 1
```

从运行结果可见,调用交换两个引用形参的 mySwap 函数后,两个实参的值也发生了交换。因为在参数传递时,相当于执行语句"int &a=m; int &b=n;",即引用形参 $a$、$b$ 分别是实参 $m$、$n$ 的别名,则对 $a$、$b$ 的改变就是对 $m$、$n$ 的改变。可见,通过引用形参可以达到返回多个值的目的。注意,引用形参对应的实参必须是相同类型的变量。

**例 5.5.2　分析程序运行结果**


例 5.5.2

```
#include<iostream>
#include<string>
using namespace std;
void f(string s, int i, int j) {
    while(i<j) swap(s[i++], s[j--]);
    cout<<"In f(): "<<s<<endl;
}
int main() {
    string ts="1234567";
    f(ts,0,ts.size()-1);
    cout<<"In main(): "<<ts<<endl;
    return 0;
}
```

运行结果:

```
In f(): 7654321
In main(): 1234567
```

分析可知，f 函数的功能是逆置形参 *s*，因此在 f 函数中输出的 *s* 是逆置后的结果；但因为 f 函数的三个参数都是值参数，即只是把实参的值传给形参而不能再从形参传回给实参，因此形参 *s* 在 f 函数中的改变不会影响实参 *ts*，则 main 函数中输出的 *ts* 保留原值。

若希望 f 函数中形参 *s* 的改变使得 main 函数中的实参 *ts* 相应改变，则应如何改写 f 函数？由于引用形参可以达到形参改变则实参改变的效果，因此只要在形参 *s* 的前面增加引用定义符，即把 f 函数的函数头改为“void f(string &s, int i, int j)”。可见，引用形参可以把形参的改变“传回”给实参，实际上是因为引用形参是实参的别名，在函数调用期间引用形参和对应的实参是同一个对象。

**例 5.5.3　以引用参数返回多个值**


例 5.5.3

编写函数，以引用参数方式返回 *n* 个整数的最大值、最小值、大于平均值的数据的个数。

输入数据个数 *n*（*n*≤100），然后输入 *n* 个整数，再调用函数得到最大值、最小值、大于平均值的数据的个数并输出。例如，输入 5 4 3 5 1 2，则输出 5 1 2。

通过引用形参可以返回多个值，因此本题可以使用三个引用形参分别返回最大值、最小值、大于平均值的数据的个数。具体代码如下。

```
#include<iostream>
using namespace std;
void solve(int a[], int n, int &max, int &min, int &cnt) {
    int i,sum=0;
    for(i=0; i<n; i++) sum+=a[i];
    double avg=sum*1.0/n;
    cnt=0;
    for(i=0; i<n; i++) {
        if(a[i]>avg) cnt++;
    }
    max=min=a[0];
    for(i=1; i<n; i++) {
        if(max<a[i]) max=a[i];
    }
    for(i=1; i<n; i++) {
        if(min>a[i]) min=a[i];
    }
}
int main() {
    int a[100],max,min,cnt,n;
    cin>>n;
    for(int i=0; i<n; i++)cin>>a[i];
    solve(a,n,max,min,cnt);
    cout<<max<<" "<<min<<" "<<cnt<<endl;
    return 0;
}
```

# 5.6　递归函数

## 5.6.1　递归基础

递归函数是直接或间接地调用自身的函数，可分为直接递归函数和间接递归函数。本

书仅讨论直接递归函数。递归函数的两个要素是边界条件(递归出口)与递归方程(递归式),只有具备了这两个要素,才能在有限次计算后得出结果。

对于简单的递归问题,关键是分析得出递归式,并在递归函数中用 if 语句表达。



**例 5.6.1　递归函数求 $n!$**

例 5.6.1

递归式 $n!=\begin{cases}1, & n=0,1\\ n(n-1)!, & n>1\end{cases}$

根据 $n!$ 的递归式,直接用 if 语句表达。

```
int f(int n) {
    if (n == 0 || n == 1) return 1;
    else return f(n - 1) * n;
}
```

递归函数的执行分为扩展和回代两个阶段。例如,$f(5)$的调用先不断扩展到递归出口求出结果为 1,然后逐步回代结果到各个调用点,最终的调用结果为 120,如图 5-2 所示。

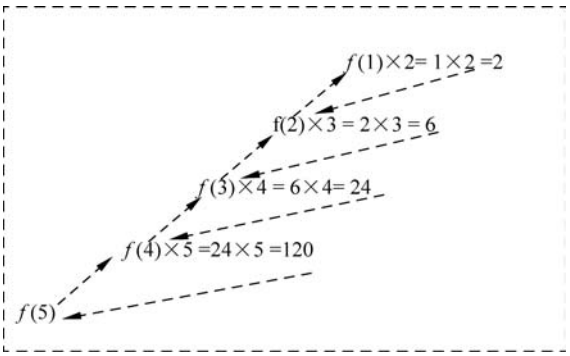


图 5-2　递归调用过程示意图

另外,$n!$ 增长速度很快(13! 已超出 int 表示范围),需注意溢出问题。那么,稍大一点儿的数的阶乘怎么办?更大的数,例如求 100! 又怎么办呢?读者可以自行思考或参考第 9 章求解。

递归是实现分治法和回溯法的有效手段。分治法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似问题,各个击破,分而治之。回溯法是一种按照选优条件往前搜索,在不能再往前时回退到上一步的方法。

**例 5.6.2　最大公约数函数**

例 5.6.2

输入两个正整数 $a$、$b$,求这两个整数的最大公约数。要求定义一个函数求最大公约数。

已知两个正整数的最大公约数是能够同时整除它们的最大正整数。求最大公约数可以用穷举法,也可以用辗转相除法(欧几里得算法)。

利用辗转相除法确定两个正整数 $a$ 和 $b$ 的最大公约数的算法思想如下。

若 $a\%b=0$,则 $b$ 即为最大公约数,否则 $\gcd(a,b)=\gcd(b,a\%b)$。

即递归式如下。

$$\gcd(a,b)=\begin{cases}b, & a\%b=0\\ \gcd(b,a\%b), & a\%b!=0\end{cases}$$

根据辗转相除法的思想，求最大公约数的递归版和迭代版函数如下。

```
int gcd(int a, int b) {                    //递归版
    if(a % b == 0)
        return b;
    else
        return gcd(b, a % b);
}
int gcdIt(int a, int b) {                  //迭代版
    while(a % b!= 0) {
        int t = a % b;
        a = b;
        b = t;
    }
    return b;
}
```

迭代法是一种不断地用变量的原值(旧值)递推出其新值的方法。例如，上面的迭代版代码中，不断地用 $a$、$b$ 的旧值递推出新值。

通过调用以上定义的最大公约数函数，可以方便地求得两个整数的最小公倍数，也可以方便地求得多个整数的最大公约数或最小公倍数，具体代码实现留给读者自行完成。

## 5.6.2 典型递归问题

**例 5.6.3 斐波那契数列**


例 5.6.3

意大利数学家斐波那契(Leonardo Fibonacci)是 12、13 世纪欧洲数学界的代表人物。他提出的“兔子问题”引起了后人的极大兴趣。

“兔子问题”假定一对大兔子每一个月可以生一对小兔子，而小兔子出生后两个月就有繁殖能力，问从一对小兔子开始，$n$ 个月后能繁殖成多少对兔子？

这是一个递推问题，可以构造一个递推的表格，如表 5-1 所示。

**表 5-1 兔子问题递推表**

| 时间/月 | 小兔/对 | 大兔/对 | 总数/对 |
|---|---|---|---|
| 1 | 1 | 0 | 1 |
| 2 | 0 | 1 | 1 |
| 3 | 1 | 1 | 2 |
| 4 | 1 | 2 | 3 |
| 5 | 2 | 3 | 5 |
| 6 | 3 | 5 | 8 |
| 7 | 5 | 8 | 13 |
| 8 | 8 | 13 | 21 |
| 9 | 13 | 21 | 34 |
| 10 | 21 | 34 | 55 |

从表 5-1 可得，每月的兔子总数构成如下数列：

1,1,2,3,5,8,13,21,34,55,…

可以发现此数列的规律：前两项是 1，从第三项起，每一项都是其前两项的和。

因此,可得递归式如下。

$$f(n)=\begin{cases}1, & n=1,2\\ f(n-1)+f(n-2), & n>2\end{cases}$$

根据递归式,容易写出求斐波那契数列第 $n$ 项的递归函数。具体代码如下。

```
#include<iostream>
using namespace std;
//求解斐波那契数列(递归法)
int fib(int n) {
    if(n<=2)
        return 1;
    else
        return fib(n-1)+fib(n-2);
}
int main() {
    int n;
    cin>>n;
    cout<<fib(n)<<endl;
    return 0;
}
```

若在本地运行时输入 $n$ 为 40,程序需要较长的时间才能得到结果,若在线提交一般将得到超时反馈。一般而言,递归的深度不宜过大,否则递归程序的执行效率过低,在线做题时将导致超时。此时可用迭代法改写 fib 函数。具体代码如下。

```
//求解斐波那契数列(迭代法)
int fib(int n) {
    if(n<=2) return 1;
    int f1=1,f2=1,ans;
    for(int i=3; i<=n; i++) {
        ans=f1+f2;
        f1=f2;
        f2=ans;
    }
    return ans;
}
```

另外,注意到斐波那契数列的增长速度也很快,当输入 $n$ 为 47 时,结果已经超出 int 的表示范围,若要求斐波那契数列第 46 项之后的若干项,可以使用 long long int 类型。为了避免在线做题超时,可以把斐波那契数列的所有项一次性算出来存放在外部数组(定义在函数之外的数组)中,输入数据后直接从数组中把结果取出来,即空间换时间。具体代码如下。

```
#include<iostream>
using namespace std;
const int N=93;                              //增长速度快,用 long long int
long long int a[N]={1,1};                    //外部数组
void fib() {              //使用此函数一次性算得结果放在 a 中,main 中一开始调用一次
    for(int i=2; i<N; i++) {
        a[i]=a[i-1]+a[i-2];
    }
```

```
}
int main() {
    fib();                                  //一次性把结果算出来,放在a数组中,调用一次即可
    int n;
    while(cin>>n) {                         //增加循环输入,以便于多次输入
        cout<<a[n-1]<<endl;                 //下标从0开始
    }
    return 0;
}
```

关于斐波那契数列,有许多有趣的知识,有兴趣的读者可以自行了解。例如,斐波那契数列螺旋线(当海螺被切成两半的时候,它内腔壁的形状是“斐波那契螺旋线”形状),或当斐波那契数列趋向于无穷大时,相邻两项的比值趋向于黄金分割比例0.618。

**例5.6.4　快速幂**


例5.6.4

输入两个整数$a$、$b$,如何高效地计算$a^b$?结果保证在long long int范围内。

若$b=32$,则用循环“for(s=1,i=0;i<b;i++) s *=a;”将需要运算32次。

如果用二分法,则可以按$a^{32}\to a^{16}\to a^8\to a^4\to a^2\to a^1\to a^0$的顺序来分析,在计算出$a^0$后可以倒过去计算出$a^1\sim a^{32}$。这与递归函数的执行过程是一致的,因此可以用递归方法求解。

二分法计算$a^b$的要点举例说明如下。

(1) $a^{10}=a^5\times a^5$

(2) $a^9=a^4\times a^4\times a$

即根据$b$的奇偶性来做不同的计算,$b=0$是递归出口。因此可得递归式如下。

$$a^b=\begin{cases}1, & b=0\\ a^{\frac{b}{2}}\cdot a^{\frac{b}{2}}, & b\%2=0\\ a^{\frac{b}{2}}\cdot a^{\frac{b}{2}}\cdot a, & b\%2=1\end{cases}$$

根据递归式可以方便地实现递归函数。为减少重复计算从而提高程序执行效率,可以先计算$a^{\frac{b}{2}}$并放到临时变量中。具体代码如下。

```
#include<iostream>
using namespace std;
long long int cal(int a, int b) {
    if (b==0) return 1;
    long long int t = cal(a, b/2);
    if (b%2==0) return t*t;
    else return t*t*a;
}
int main() {
    int a,b;
    while(cin>>a>>b) {
        cout<<cal(a,b)<<endl;
    }
    return 0;
}
```

**例 5.6.5　Hanoi 塔问题**

设 A、B、C 是三个塔座。开始时,在塔座 A 上有 $n(1\leqslant n\leqslant 64)$个圆盘,这些圆盘自下而上,由大到小地叠在一起。例如,三个圆盘的 Hanoi 塔问题初始状态如图 5-3 所示。现在要求将塔座 A 上的这些圆盘移到塔座 B 上,并仍按同样顺序叠放。在移到圆盘时应遵守以下移动规则。

例 5.6.5

规则(1):每次只能移动一个圆盘。

规则(2):任何时刻都不允许将较大的圆盘压在较小的圆盘之上。

规则(3):在满足移动规则(1)和(2)的前提下,可将圆盘移至 A,B,C 中任何一个塔座上。

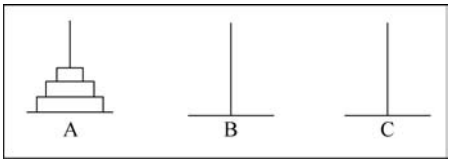


图 5-3　Hanoi 塔问题示意图(3 个圆盘)

设 $a_n$ 表示 $n$ 个圆盘从一个塔座全部转移到另一个塔座的移动次数,显然有 $a_1=1$。当 $n\geqslant 2$ 时,要将塔座 A 上的 $n$ 个圆盘全部转移到塔座 B 上,可以采用以下步骤。

(1) 先把塔座 A 上的 $n-1$ 个圆盘转移到塔座 C 上,移动次数为 $a_{n-1}$。

(2) 然后把塔座 A 上的最后一个大圆盘转移到塔座 B 上,移动次数等于 1。

(3) 最后再把塔座 C 上的 $n-1$ 个圆盘转移到塔座 B 上,移动次数为 $a_{n-1}$。

经过这些步骤后,塔座 A 上的 $n$ 个圆盘就全部转移到塔座 B 上。

由组合数学的加法规则,移动次数为 $2a_{n-1}+1$。计算总的移动次数的递归关系式如下。

$$a_n=\begin{cases}1, & n=1\\ 2a_{n-1}+1, & n>1\end{cases}$$

求解该递归关系式,可得 $a_n=2^n-1$。例如:

当 $n=3$,移动 7 次;

当 $n=4$,移动 15 次;

⋮

当 $n=64$,移动 $2^{64}-1=18\ 446\ 744\ 073\ 709\ 551\ 615$ 次,设每秒移动一次,完成所有 64 个圆盘的移动需要 $18\ 446\ 744\ 073\ 709\ 551\ 615/(365\times 24\times 60\times 60)/100\ 000\ 000\approx 5849.42$ 亿年。

如果想知道具体是如何移动的,可以根据前面的步骤,把每次只有 1 个圆盘时的移动情况输出(调用下面的 move 函数)。模拟 Hanoi 塔问题中圆盘移动过程的具体程序如下。

```
void move(char get, char put) {                    //输出移动情况,从 get 到 put
    cout << get <<" -- >"<< put << endl;
}
// 递归函数,n 个圆盘,从 from 移动到 to,借助 by
void hanoi(int n, char from, char to, char by) {
    if(n == 1) move(from,to);                      //只有 1 个直接移动
    else {
        hanoi(n - 1,from,by,to);                   //把 n - 1 个盘从 from 移动到 by,借助 to
        move(from,to);                             //只有 1 个则直接移动
        hanoi(n - 1,by,to,from);                   //把 n - 1 个盘从 by 移动到 to,借助 from
    }
}
```

```
int main() {
    int n;
    cin >> n;
    hanoi(n,'A', 'B', 'C');                    //调用,实现把 n 个盘从 A 移动到 B,借助 C
    return 0;
}
```

# 5.7 变量的作用域与生命期

## 5.7.1 变量的作用域

变量的作用域分为块作用域和文件作用域。

变量在花括号内(块)声明时,通常称之为局部变量;块作用域指的是局部变量从定义处开始可用,到块结束处为止。局部变量若不指定初值,则为随机数。

变量在函数外部声明时,通常称之为全局变量或外部变量;文件作用域指的是外部变量从声明处开始可用,到文件结束处为止。外部变量若不指定初值,则默认为相应类型的 0。

全局变量和局部变量是从空间角度来分的。不同作用域的同名变量在使用时,遵循“最近”原则。建议编程时取名有所区分。另外,for 语句的()中定义的变量也是局部变量,仅在该 for 语句中可用。

**例 5.7.1 分析程序运行结果**

```
#include<iostream>
using namespace std;
int n;                                         //全局变量,默认初值为 0
int m=3;                                       //全局变量,初始化值为 3
void f() {
    int m=6;                                   //局部变量
    cout<<n<<" "<<m<<endl;
}
int main() {
    int n=4;                                   //局部变量
    f();
    if(n>m) {
        int m=5;                               //局部变量
        cout<<n<<" "<<m<<endl;
    }
    return 0;
}
```

运行结果:

```
0 6
4 5
```

可以使用 extern 声明扩展外部变量的作用域。外部变量的作用域从定义点开始,如果

在定义点之前的函数想引用该外部变量,则应该在引用之前用关键字 extern 对该变量做“外部变量声明”,表示该变量是一个已经定义的外部变量。有了此声明,就可以从“声明”处起,合法地使用该外部变量。

**例 5.7.2　用 extern 扩展作用域**

```
#include<iostream>
using namespace std;
int max(int x, int y) {
    return x>y?x:y;
}
int main() {
    extern int A,B;
    printf("%d\n",max(A,B));
    return 0;
}
int A=1, B=3;
```

在本程序的最后一行定义了外部变量 $A$、$B$,但由于外部变量 $A$、$B$ 定义的位置在函数 main 之后,若不加以声明则在 main 函数中不能使用。通过在 main 函数中用 extern 对 $A$ 和 $B$ 进行“外部变量声明”,就可以从“声明”处起合法地使用这两个外部变量。

## 5.7.2　变量的生命期

用户存储空间可以分为三个部分:程序区,静态存储区,动态存储区。变量的存储方式可以分为静态存储(在程序运行期间分配固定的存储空间)和动态存储(在程序运行期间根据需要进行动态存储空间的分配)。

全局变量和静态局部变量(static 局部变量)存放在静态存储区,在程序执行过程中始终占据固定的存储单元。静态局部变量在编译时赋初值,且只赋初值一次;若不赋初值自动赋初值为 0(或 0.0、NULL、'\0'等)。

函数形式参数、局部变量(不包括 static 局部变量)及函数调用时的现场保护和返回地址存放在动态存储区,在函数调用开始时分配动态存储空间,函数调用结束时释放空间。


例 5.7.3

**例 5.7.3　分析程序运行结果**

```
#include<iostream>
using namespace std;
int f(int a) {
    int b=1;
    static int c=3;
    b++;
    c++;
    return a+b+c;
}
int main() {
    int a=3;
    for(int i=0;i<3;i++) printf("%d\n",f(a));
    return 0;
}
```

运行结果:

```
9
10
11
```

第一次调用时，执行 $a+b+c$ 时，$a=3$、$b=2$、$c=4$，返回 9；第二次调用时，$a$、$b$ 的值与第一次调用一样，而 $c$ 在保留原值 4 的基础上加了 1 而等于 5，故返回 10，第三次调用 $c$ 在 5 的基础上加 1 而等于 6，故返回 11。

## 5.8 编译预处理

编译预处理主要包括宏定义、文件包含和条件编译，以＃开始，不需要用分号结束，一般单独写在一行上。符号“＃”用来指明其后为编译预处理命令。

### 5.8.1 宏定义

宏定义只是简单字符串替换，经常用来实现带名常量的定义，例如：

```
#define PI 3.1415926                    /* PI 是宏名，3.1415926 是宏替换体 */
#define ZERO 1e-9                       /* ZERO 是宏名，1e-9 是宏替换体 */
#define N 100                           /* N 是宏名，100 是宏替换体 */
```

宏定义也可以带参数，例如，下面是求两个数最大值的带参宏定义。

```
#define Max(a,b) ((a)>(b)?(a):(b))
```

其中，Max($a$，$b$)是宏名及其参数，(($a$)＞($b$)？($a$)：($b$))是宏替换体。

建议带参宏定义的宏替换体中的参数都用()括起来，以避免类似以下的错误。

**例 5.8.1 分析程序运行结果**

```
#include <stdio.h>
int main() {
    #define f(a,b) a*b
    printf("%d\n", f(1+2,3+4));
    return 0;
}
```

运行结果：

```
11
```

期望的结果是(1＋2)＊(3＋4)＝21，但得到的结果是 11。原因在于宏定义是直接进行简单的字符串替换，例如，f(1＋2,3＋4)被替换为 1＋2＊3＋4，因此得到 11。

### 5.8.2 文件包含

文件包含是将一些头文件或其他源文件包含到本文件中。一个文件被包含后，该文件的所有内容都被包含进来。C/C++语言提供了＃include 命令用来实现“文件包含”的操作。在＃include 命令中，文件名可以用双引号""或尖括号<>括起来，即：

**#include <文件名>**

**#include "文件名"**

用尖括号时,称为标准方式,系统直接到存放 C/C++库函数头文件所在的目录中查找要包含的文件;用双引号时,系统先在用户当前目录中查找要包含的文件,若找不到,再按标准方式查找。

一般来说,如果为调用库函数而用#include 命令来包含相关的头文件,则用尖括号,以节省查找时间。如果要包含的是用户自己编写的文件(这种文件一般都在当前目录中),一般用双引号。

在 Dev-C++及很多 OJ 中,选择 C++或 G++编译器时,可以使用本章引例中所用的万能头文件"bits\stdC++.h",从而避免遗漏头文件而导致的编译错误。当然,在程序设计竞赛前测试比赛环境时应先测试万能头文件是否可用。

## 5.8.3 条件编译

条件编译是带#的 if 语句(有#if、#ifdef、#ifndef、#elif 等多种形式),以#endif 结束。

显然,在 OJ 中提交代码之前需要先在本地运行正确。如果样例测试数据较多,直接在控制台输入输出时容易看错。此时,可以先把测试输入数据放到一个文件中,再通过 C 语言的 freopen 函数从该文件输入数据,并把测试结果输出到一个新文件中,即把以下代码放到 main 函数的开始处。

```
freopen("input.txt","r",stdin);          //input.txt 在当前路径,也可以指定路径
freopen("output.txt","w",stdout);        //output.txt 自动生成在当前路径
```

为避免包含调用 freopen 函数的代码直接提交到 OJ 导致错误,可以把 freopen 放到条件编译中,如下所示。

```
#ifndef ONLINE_JUDGE
    freopen("input.txt","r",stdin);      //第二、三个参数固定即可
    freopen("output.txt","w",stdout);    //第二、三个参数固定即可
#endif
```

ONLINE_JUDGE 是一个符号常量,通常 OJ 上都有定义,因此提交后并不会去打开文件;而本地没有定义这个符号常量,就会使用 freopen 函数分别以读方式("r")打开输入文件 input.txt(需先建好并放好数据)、以写方式("w")打开输出文件 output.txt(自动生成)。freopen 的第一个参数是具体文件名,可以指定文件所在的路径,例如,e:\\ output.txt。建议程序设计竞赛正式比赛前先测试评测 OJ 是否定义了符号常量 ONLINE_JUDGE。

**例 5.8.2 重复读**

例 5.8.2

输入一个整数 $n$,再输入 $n$ 个字符串(可能包含空格),请原样输出 $n$ 个字符串。要求使用 freopen 文件测试的方法,先把输入数据放在 D 盘的 1.txt 文件中,再把结果输出到 D 盘的 2.txt 文件中。

本题用 getline 函数输入字符串,用 cin.get()吸收整数后的换行符,用 freopen 函数打开文件读写。具体代码如下。

```
#include<iostream>
#include<string>
using namespace std;
int main() {
#ifndef ONLINE_JUDGE
    freopen("d:\\1.txt","r",stdin);
    freopen("d:\\2.txt","w",stdout);
#endif
    int n;
    cin>>n;
    cin.get();
    for(int i=0;i<n;i++) {
        string s;
        getline(cin,s);
        cout<<s<<endl;
    }
    return 0;
}
```

注意,D 盘下的 1.txt 文件必须存在,设其中内容如下。

```
3
I like programming.
Enjoying it.
Just do it.
```

则将自动在 D 盘下生成文件 2.txt,包含以下输出结果。

```
I like programming.
Enjoying it.
Just do it.
```

## 5.9 程序调试

### 5.9.1 调试简介

程序调试在程序设计学习过程中是必不可少的。所谓程序调试是指在发现程序运行结果有误时,测试、分析并修正程序的错误之处的过程。为便于调试,并增强程序的可读性,一般需规范代码缩进格式。在 VC6、VC2010 和 Dev-C++等编译环境下可以分别使用快捷键 Shift+F8、Ctrl+K+F 和 Ctrl+Shift+A 缩排代码。

最基本的调试方法是在必要的地方添加输出语句,通过查看输出结果分析错误所在并修正程序。

常用的调试方法一般包括设置断点、单步执行、监视变量等基本步骤。其中,监视变量是把需要监视的变量等添加到监视(查看)窗口中并观察其值。

在 VC6 下,使用 F9 键在光标处设置断点,使用 F5 键开始调试,使用 F10 键单步执行。其他调试快捷键可以查看调试菜单,如图 5-4 所示:


图 5-4　VC6 调试菜单

VC6 调试

Dev-C++调试

## 5.9.2　Dev-C++调试过程

以下程序的目的是逐个判断输入的整数 $m$ 是否素数,是则输出“yes”,否则输出“no”。

```
#include<iostream>
#include<cmath>
using namespace std;
bool isPrime(int n) {
    for (int i=2; i<sqrt(n); i++) {
        if (n%i==0) return false;
    }
    return true;
}
int main() {
    int m;
    while(cin>>m) {
        if(isPrime(m)==true)
            cout<<"yes"<<endl;
        else
            cout<<"no"<<endl;
    }
    return 0;
}
```

运行结果:

```
1↵
yes
9↵
yes
```

从运行结果看,该程序有误。但若一时发现不了错误所在,则可以开始调试。

在 Dev-C++下,在调试前需要先把“产生调试信息”设置为 Yes,如图 1-8 所示。

在 Dev-C++下，简单的调试过程如下所示。

**1. 设置断点，开始调试**

通过单击行号设置断点；使用 F5 键(或单击快捷菜单中的√)开始调试，如图 5-5 所示(此处断点设在第 5 行)。

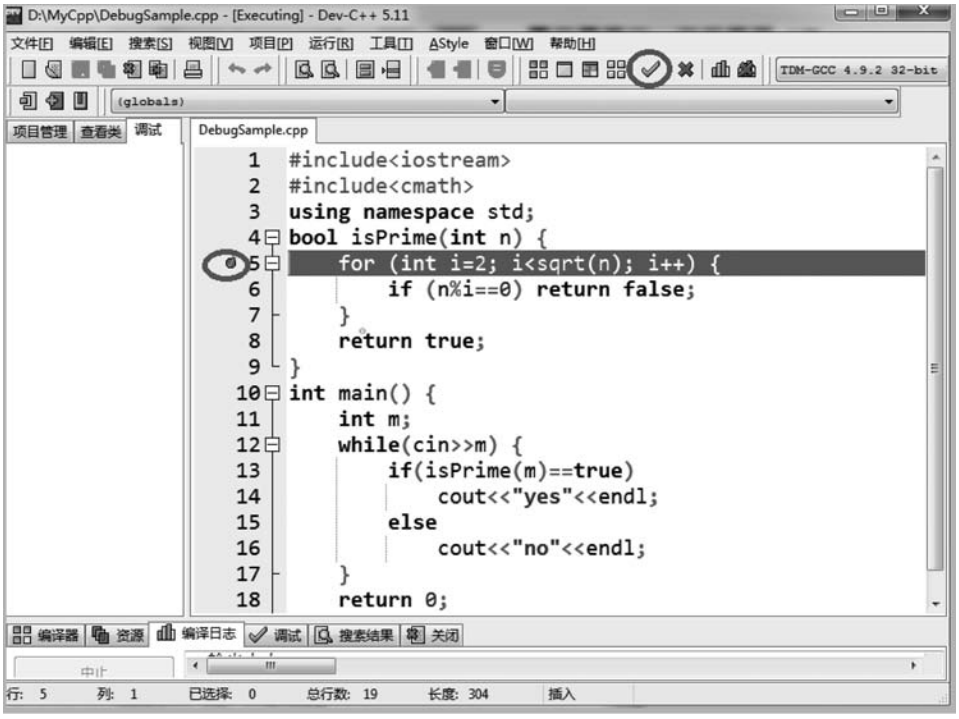


图 5-5　Dev-C++调试之设置断点

**2. 单步执行**

开始调试后，程序运行到断点处停住，使用 F7 键或单击左下角“调试”区域的“下一步”按钮开始单步执行，如图 5-6 所示。

**3. 添加监视**

单击“调试”区域的“添加查看”按钮添加监视量，如图 5-7 所示(此处已添加的监视变量 $n$，正在添加的变量是 $i$，单击 OK 按钮后将添加该变量)。

**4. 继续调试**

单击“下一步”按钮，程序流程跳到第 8 行，即将返回 true，观察变量 $i$ 的值，发现是 1(随机数)，循环条件 $i<\text{sqrt}(n)$不成立，考虑到 $i$ 从 2 开始循环，$\text{sqrt}(1)=1$，不可能执行循环体，因此要对 1 进行特判。单击“调试”区域的“停止执行”按钮结束调试，修改代码并重新编译，输入 1 结果正确，但输入 9 结果依然错误。重新从设置断点开始调试。添加监视 $\text{sqrt}(n)$，发现其值为 3，而 $i$ 的值单步执行下来也为 3，如图 5-8 所示。

**5. 纠错**

思考并发现错误在于没有判断到 3 是否是 $n$ 的因子，因此循环条件应改为 $i<=\text{sqrt}(n*1.0)$。使用 F6 键(或单击“停止执行”按钮或单击快捷菜单中的“×”)结束调试，重新运行程序无误。

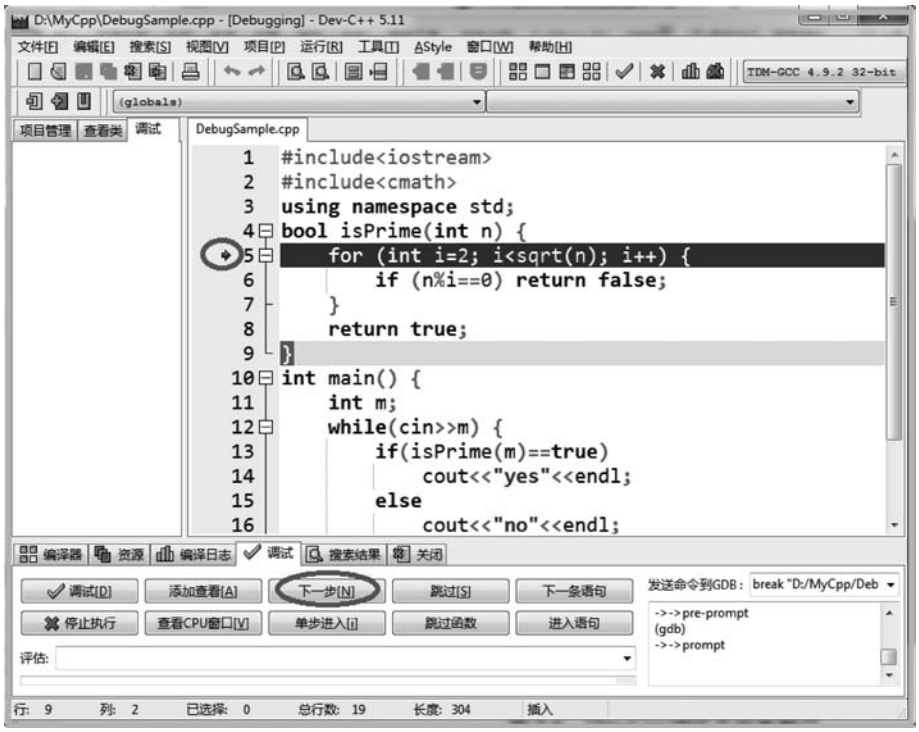


图 5-6　Dev-C++调试之单步执行

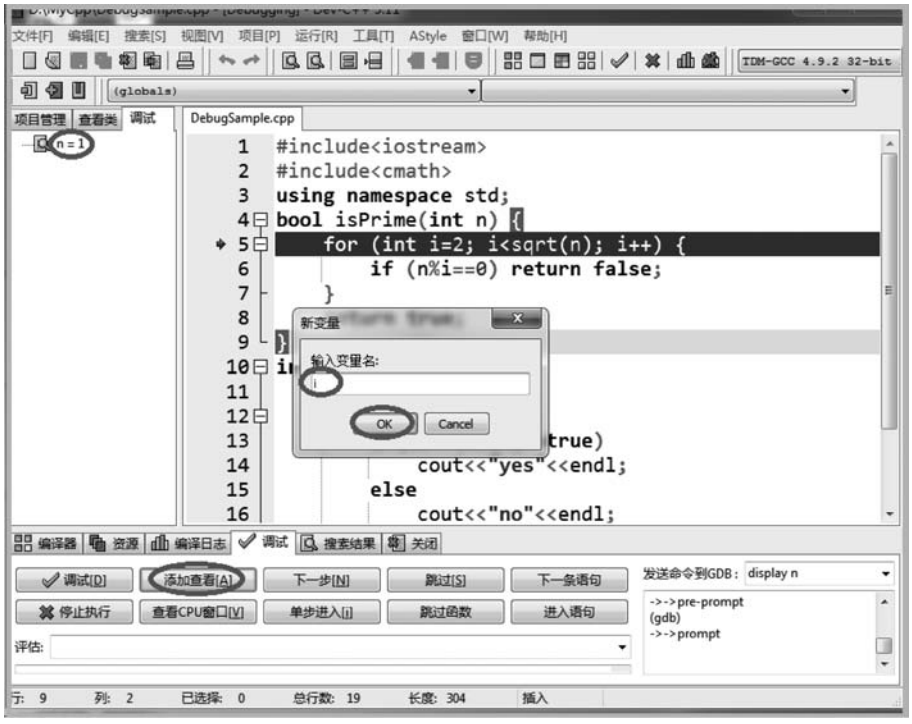


图 5-7　Dev-C++调试之添加监视

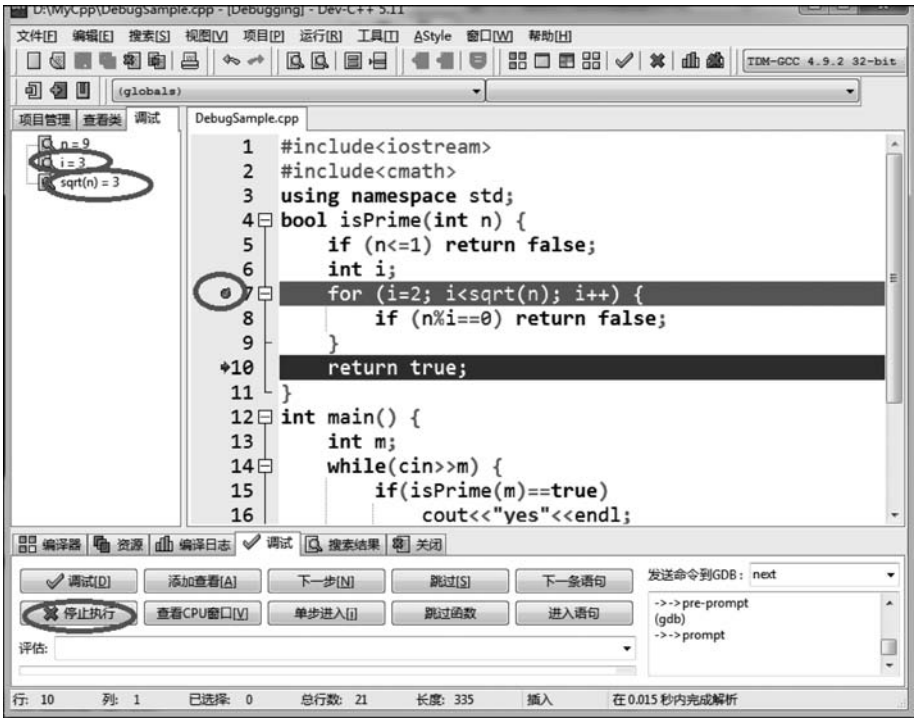


图 5-8　Dev-C++调试之继续调试

在掌握上述基本的调试步骤之后，读者可以学习使用“调试”区域的其他调试按钮进一步掌握 Dev-C++的调试方法。

## 5.10　OJ 题目求解

**例 5.10.1　验证哥德巴赫猜想(HLOJ 1922)**


例 5.10.1

**Problem Description**

哥德巴赫猜想之一是指一个偶数(2 除外)可以拆分为两个素数之和。请验证这个猜想。

因为同一个偶数有可能拆分为不同的素数对之和，这里要求结果素数对彼此最接近。

**Input**

首先输入一个正整数 $T$，表示测试数据的组数，然后是 $T$ 组测试数据。每组测试输入一个偶数 $n(6\leqslant n\leqslant 10\ 000)$。

**Output**

对于每组测试，输出两个彼此最接近的素数 $a$、$b(a\leqslant b)$，两个素数之间留一个空格。

| Sample Input |
| --- |
| 2 |
| 30 |
| 40 |

| Sample Output |
| --- |
| 13 17 |
| 17 23 |

本题可以先写一个判断某个正整数是否素数的函数,然后循环变量 $i$ 从 $n/2$ 处开始到 2 进行循环(因为两个素数的差值 $d=(n-i)-i=n-2i$,当 $i$ 越大时 $d$ 越小),若发现 $i(\leqslant n/2)$ 和 $n-i(\geqslant n/2)$同时是素数则得到结果并结束循环。具体代码如下。

```
#include <iostream>
#include <cmath>
using namespace std;
bool isPrime(int n) {
    if(n == 1) return false;
    bool flag = true;
    int m = sqrt(n * 1.0);
    for(int i = 2; i <= m; i++) {
        if (n % i == 0) {
            flag = false;
            break;
        }
    }
    return flag;
}
int main() {
    int T;
    cin >> T;
    while(T--) {
        int n;
        cin >> n;
        for(int i = n/2; i >= 2; i--) {
            if (isPrime(i) == true && isPrime(n - i) == true) {
                cout << i <<" "<< n - i << endl;
                break;
            }
        }
    }
    return 0;
}
```

例 5.10.2

**例 5.10.2　素数的排位(ZJUTOJ 1341)**

**Problem Description**

已知素数序列为 2、3、5、7、11、13、17、19、23、29、…,即素数的第一个是 2,第二个是 3,第三个是 5,…

那么,对于输入的一个任意整数 $N$,若是素数,能确定是第几个素数吗?若不是素数,则输出 0。

**Input**

测试数据有多组,处理到文件尾。每组测试输入一个正整数 $N(1\leqslant N\leqslant 1\ 000\ 000)$。

**Output**

对于每组测试,输出占一行,如果输入的正整数 $N$ 是素数,则输出其排位,否则输出 0。

| Sample Input | Sample Output |
| --- | --- |
| 13 | 6 |

本题可以利用上题的 isPrime 函数及空间换时间的思想一次性把排位计算出来放在数组中,输入数据时再直接从数组中取结果输出。但这种方法对每个数都要调用 isPrime 函数,效率依然较低。若用筛选法,则能较好地提高效率。具体代码如下。

```
#include <iostream>
#include <cmath>
using namespace std;
const int N = 1000001;
int a[N];                                   //排名数组,数组太大,开在函数之外
bool f[N];                                  //标记数组
int m;
void init() {                               //筛选法
    int i;
    for(i = 1; i < N; i++) f[i] = true;
    f[1] = false;
    int k = sqrt(N);
    for(i = 2; i <= k; i++) {
        if (f[i] == false) continue;
        for(int j = i * i; j < N; j += i) {
            f[j] = false;
        }
    }
}
void rank() {                               //排位
    int cnt = 0;
    for(int i = 1; i < N; i++) {
        if (f[i] == true) {
            cnt++;
            a[i] = cnt;
        }
        else {
            a[i] = 0;
        }
    }
}
int main() {
    init();
    rank();
    int n;
    while(cin >> n) {
        cout << a[n]<< endl;                //输入数据时直接从数组中取数据
    }
    return 0;
}
```

如果读者已经熟练掌握了筛选法,实际上可以改写 init 函数,同时进行排位和筛选。具体代码如下。

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

```
const int N = 1000001;
int a[N];                                    //数组太大,作为外部数组
void init() {
    int i, cnt = 0;
    for(i = 1; i < N; i++) a[i] = 1;
    a[1] = 0;
    for(i = 2; i < N; i++) {
        if (a[i] == 0) continue;             //非素数则跳过
        a[i] = ++cnt;                        //cnt 初值为 0,第一素数从 1 开始排位
        if (i > N/i) continue;               //i > sqrt(N)的另一种表达
        for(int j = i * i; j < N; j += i) {  //筛掉非素数
            a[j] = 0;
        }
    }
}
int main() {
    init();
    int n;
    while(cin >> n) {
        cout << a[n]<< endl;
    }
    return 0;
}
```

例 5.10.3

### 例 5.10.3　母牛问题(ZJUTOJ 1182)

**Problem Description**

若一头小母牛从第 4 个年头开始每年生育一头母牛,按照此规律,第 $n$ 年时有多少头母牛?要求编写两个函数,分别以迭代、递归的方法进行求解。

**Input**

测试数据有多组,处理到文件尾。每组测试输入一个正整数 $n(1\leqslant n\leqslant 40)$。

**Output**

对于每组测试,输出第 $n$ 年时的母牛总数。

| Sample Input | Sample Output |
| --- | --- |
| 15 | 129 |

本题也是一道递推题,递推表如表 5-2 所示。

**表 5-2　母牛问题递推表**

| 时间/年 | 小　牛 | 中　牛 | 大　牛 | 总　数 |
| --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 1 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 1 | 0 | 1 | 2 |
| 5 | 1 | 1 | 1 | 3 |
| 6 | 1 | 1 | 2 | 4 |

续表

| 时间/年 | 小　牛 | 中　牛 | 大　牛 | 总　数 |
|---|---|---|---|---|
| 7 | 2 | 1 | 3 | 6 |
| 8 | 3 | 2 | 4 | 9 |
| 9 | 4 | 3 | 6 | 13 |
| 10 | 6 | 4 | 9 | 19 |

根据表 5-2,可以得到如下数列。

1、1、1、2、3、4、6、9、13、19、…

观察数列,发现规律如以下递归式所示。

$$f(n)=\begin{cases}1, & n=1,2,3\\ f(n-1)+f(n-3), & n\geqslant 4\end{cases}$$

根据递归式,容易写出使用递归法或迭代法的函数。另外,本题也可以采用空间换时间的思想:一次性先把结果计算出来并放在数组中,然后在输入数据时从数组中取出数据。

具体代码如下。

```
#include <iostream>
using namespace std;
int f(int n) {                          //递归法
    if (n<4)
        return 1;
    return f(n-1)+f(n-3);
}
int cow(int n) {                        //迭代法
    if (n<4) return 1;
    int n1=1, n2=1, n3=1, n4;           //分别表示(当前)第 1～4 年的牛数
    for(int i=4; i<=n; i++) {
        n4=n3+n1;
        n1=n2;
        n2=n3;
        n3=n4;
    }
    return n4;
}
const int N=41;
int a[N]= {0,1,1,1};                    //外部数组,保存结果,空间换时间
void init() {
    for(int i=4; i<N; i++) {
        a[i]=a[i-1]+a[i-3];
    }
}
int main() {
    init();
    int n;
    while(cin>>n) {
        cout<<f(n)<<endl;               //递归法结果
        //cout<<cow(n)<<endl;           //迭代法结果
```

```
        //cout << a[n]<< endl;                    //空间换时间结果
    }
    return 0;
}
```



读者不妨再观察数列,尝试找到其他的递归式。


例 5.10.4

**例 5.10.4　特殊排序(HLOJ 1923)**

**Problem Description**

输入一个整数 $n$ 和 $n$ 个各不相等的整数,将这些整数从小到大进行排序,要求奇数在前,偶数在后。

**Input**

首先输入一个正整数 $T$,表示测试数据的组数,然后是 $T$ 组测试数据。每组测试先输入一个整数 $n(1<n<100)$,再输入 $n$ 个整数。

**Output**

对于每组测试,在一行上输出根据要求排序后的结果,数据之间留一个空格。

**Sample Input**

```
3
5 1 2 3 4 5
3 12  4 5
6 2 4 6 8 0 1
```

**Sample Output**

```
1 3 5 2 4
5 4 12
1 0 2 4 6 8
```

本题采用函数的方法来实现,主要是写两个函数,一个是表明排序规则的比较函数 cmp,另一个是自定义的排序函数 mySort(使用冒泡排序的思想)。比较函数中体现题意要求的奇数在前,偶数在后,并都按从小到大排序。具体代码如下。

```
#include<iostream>
using namespace std;
bool cmp(int a, int b) {                        //比较函数
    if (a%2 != b%2)                             //奇偶性不同
        return a%2 != 0;                        //前奇后偶,a%2 != 0 表明 a 是奇数
    else
        return a<b;                             //前小后大
}
void mySort (int a[], int n) {
    for ( int i=0; i<n-1; i++) {
        for ( int j=0; j<n-1-i; j++) {
            if (cmp(a[j], a[j+1]) == false) {
                int t = a[j];
                a[j] = a[j+1];
                a[j+1] = t;
            }
        }
    }
}
int main() {
    int T;
```

```
    cin >> T;
    while(T -- ) {
        int n;
        cin >> n;
        int a[100];
        for (int i = 0; i < n; i++) cin >> a[i];
        mySort (a, n);
        for (int j = 0; j < n; j++) {
            if (j > 0) cout <<" ";
            cout << a[j];
        }
        cout << endl;
    }
    return 0;
}
```

请读者仔细体会 cmp 函数的写法。实际上,对于 OJ 做题,可以从小到大直接排好序,输出时先输出奇数,再输出偶数,具体代码留给读者自行完成。

**例 5.10.5　平方和排序(ZJUTOJ 1038)**


例 5.10.5

**Problem Description**

输入 int 类型范围内的 $N$ 个非负整数,要求按各个整数的各数位上数字的平方和从小到大排序,若平方和相等则按数值从小到大排序。

例如,三个整数 9、31、13,各数位上数字的平方和分别为 81、10、10,则排序结果为 13、31、9。

**Input**

测试数据有多组。每组数据先输入一个整数 $N(0<N<100)$,然后输入 $N$ 个非负整数。若 $N=0$,则输入结束。

**Output**

对于每组测试,在一行上输出按要求排序后的结果,数据之间留一个空格。

**Sample Input**

```
9
12 567 91 33 657 812 2221 3 77
0
```

**Sample Output**

```
12 3 2221 33 812 91 77 567 657
```

本题用函数的方法来实现,主要是分别定义 cmp 比较函数、mySort 排序函数。本题的 mySort 使用选择排序的思想。方便起见,写一个函数求一个整数各数位上数字的平方和。具体代码如下。

```
#include < iostream >
#include < string >
using namespace std;
int sumSquare(int n) {                    //整数 n 的各数位上数字的平方和
    int sum = 0;
    while(n > 0) {
        sum = sum + (n % 10) * (n % 10);
```

```
        n = n/10;
    }
    return sum;
}
bool cmp(int a, int b) {                    //比较函数
    int sa = sumSquare(a), sb = sumSquare(b);
    if (sa == sb)                           //若平方和相等
        return a < b;                       //则按数据本身前小后大
    else
        return sa < sb;                     //若平方和不等,则按平方和前小后大
}
void mySort(int a[], int n) {               //排序函数
    for(int i = 0; i < n - 1; i++) {
        int k = i;
        for(int j = i + 1; j < n; j++) {
            if (cmp(a[k],a[j]) == false) k = j;
        }
        if (k!= i) swap(a[k], a[i]);
    }
}
int main() {
    while(true) {
        int n;
        cin >> n;
        if (n == 0) break;
        int a[100];
        for (int i = 0; i < n; i++) cin >> a[i];
        mySort (a, n);
        for (int j = 0; j < n; j++) {
            if (j > 0) cout << " ";
            cout << a[j];
        }
        cout << endl;
    }
    return 0;
}
```

例 5.10.6

**例 5.10.6　按长度排序(ZJUTOJ 1030)**

**Problem Description**

先输入一个正整数 $N$,再输入 $N$ 个整数,要求对 $N$ 个整数进行排序:先按长度从小到大排,若长度一样则按数值从小到大排。

**Input**

测试数据有多组。每组测试数据的第一行输入一个整数 $N(0<N<100)$,第二行输入 $N$ 个整数(每个整数最多可达 80 位)。若 $N=0$,则输入结束。

**Output**

对于每组测试,输出排序后的结果,每个数据占一行。每两组测试结果之间留一个空行。

**Sample Input**

```
3
123
12
3333
0
```

**Sample Output**

```
12
123
3333
```

本题用函数的方法来实现，主要是分别定义 cmp 比较函数、mySort 排序函数。由于输入的整数位数可能达到 80 位，采用 string 类型处理。实际上，mySort 与前一个例子的不同之处仅在于数组的类型不一样。所以关键是 cmp 函数，根据题意，先比长度，若长度不等则返回前后字符串长度的比较结果，否则返回前后字符串大小的比较结果。两组测试结果之间留一个空行可以使用计数器的方法控制。具体代码如下。

```
#include <iostream>
#include <string>
using namespace std;
const int N = 100;
void prt(string a[], int n) {                    //输出函数
    for(int i = 0; i < n; i++) cout << a[i]<< endl;
}
bool cmp(string a, string b) {                   //比较函数
    if (a.size()!= b.size())                     //若长度不等,则直接比长度
        return a.size()< b.size();
    return a < b;                                //若长度相等,则比大小
}
void mySort(string a[], int n) {                 //排序函数
    for(int i = 0; i < n - 1; i++) {
        int k = i;
        for(int j = i + 1; j < n; j++) {
            if (cmp(a[k],a[j]) == false) k = j;
        }
        if (k!= i) swap(a[k], a[i]);
    }
}
int main() {
    string b[N];
    int i, n,cnt = 0;
    while(true) {
        cin >> n;
        if (n == 0) break;
        for(i = 0; i < n; i++) cin >> b[i];
        mySort(b, n);
        cnt++;
        if (cnt > 1) cout << endl;               //用计数器方法控制,两组测试结果之间留一个空行
        prt(b,n);
    }
    return 0;
}
```

**例 5.10.7　按日期排序(ZJUTOJ 1045)**

**Problem Description**

输入若干日期,按日期从小到大排序。

**Input**


例 5.10.7

本题只有一组测试数据,且日期总数不超过 100 个。按"MM/DD/YYYY"(月/日/年,其中,月份、日份各 2 位,年份 4 位)的格式逐行输入若干日期。

**Output**

按"MM/DD/YYYY"的格式输出已从小到大排序的各个日期,每个日期占一行。

| Sample Input | Sample Output |
|---|---|
| 12/31/2005 | 11/12/1999 |
| 10/21/2003 | 10/21/2003 |
| 02/12/2004 | 10/22/2003 |
| 11/12/1999 | 02/12/2004 |
| 10/22/2003 | 11/30/2005 |
| 11/30/2005 | 12/31/2005 |

本题输入的日期格式是"月/日/年"的格式,而日期排序实际上是按"年/月/日"格式进行的;一种思路是定义三个整型数组分别存放年、月、日,再依次按年、月、日的顺序分别排序。因为日期格式固定为"MM/DD/YYYY",若把日期格式改为"YYYY/MM/DD",则可以直接按字符串大小进行比较(定义一个返回类型为 bool 的比较函数 cmp)。在程序设计竞赛时,排序相关的问题一般通过调用系统函数 sort 来提高编程效率,该函数表明比较规则的第三个参数通常是根据题意自定义的比较函数 cmp。本题具体代码如下。

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
const int N = 100;
bool cmp(string a, string b) {
    a = a.substr(6) + "/" + a.substr(0,5);     //改为"YYYY/MM/DD"格式
    b = b.substr(6) + "/" + b.substr(0,5);     //改为"YYYY/MM/DD"格式
    return a < b;                              //日期的比较就是日期字符串的比较,前小后大
}
int main() {
    string a[N],t;
    int n = 0;
    while(cin >> a[n++]);
    sort(a,a + n,cmp);                         //sort 的第 3 个参数是 bool 类型的比较函数
    for(int i = 0; i < n; i++) cout << a[i]<< endl;
    return 0;
}
```

本题还可以用结构体数组排序等其他方法求解,读者可在掌握相关知识后自行编程实现。

# 习　　题

## 一、选择题

1. 当一个函数无返回值时，函数的返回类型应为(　　)。

A. 任意　　B. void　　C. int　　D. char

2. C/C++语言中不可以嵌套的是(　　)。

A. 函数调用　　B. 函数定义　　C. 循环语句　　D. 选择语句

3. 在C/C++语言中函数返回值的类型是由(　　)决定的。

A. 主函数　　B. return语句中的表达式类型

C. 函数定义中指定的返回类型　　D. 调用该函数时的主调函数类型

4. 以下(　　)函数，没有返回值。

A. int a(){ int a=2; return a * a; }

B. void b(){ printf("c\n"); }

C. int c(){ int a=2; return a * a * a;}

D. 以上都是

5. 被调函数返回给主调函数的值称为(　　)。

A. 形参　　B. 实参　　C. 返回值　　D. 参数

6. 被调函数通过(　　)语句，将值返回给主调函数。

A. if　　B. for　　C. while　　D. return

7. 有 void f(int a){ a=3;}，则

```
int n = 1;
f(n);
cout << n << endl;
```

的结果是(　　)。

A. 3　　B. 1　　C. 0　　D. 不确定

8. 函数定义如下：

```
void f(int b) { b = 9;}
```

实参数组及函数调用如下：

```
int a[5] = {1};
f(a[1]);
```

则以下输出语句的结果为(　　)。

```
cout << a[1]<< endl;
```

A. 0　　B. 1　　C. 9　　D. 以上都不对

9. 函数f定义如下，执行语句 m=f(5); 后，m的值应为(　　)。

```
int f(int k) {
    if(k == 0 || k == 1) return 1;
```

```
    else return f(k - 1) + f(k - 2);
}
```

A. 3　　B. 8　　C. 5　　D. 13

10. 递归函数的两个要素是(　　)。

A. 函数头、函数体　　B. 递归出口、边界条件

C. 边界条件、递归方程　　D. 递归表达式、递归方程

11. 若使用一维数组名作函数实参,则以下正确的说法是(　　)。

A. 必须在主调函数中说明此数组的大小

B. 实参数组类型与形参数组类型可以不匹配

C. 在被调用函数中,不需要考虑形参数组的大小

D. 实参数组名与形参数组名必须一致

12. 函数定义如下:

```
void f(int b[ ]) { b[1] = 9;}
```

实参数组及函数调用如下:

```
int a[5] = {1};
f(a);
```

则以下输出语句的结果为(　　)。

```
cout << a[1]<< endl;
```

A. 0　　B. 1　　C. 9　　D. 以上都不对

13. 关于数组名作为函数的说法错误的是(　　)。

A. 参数传递时,实参数组的首地址传递给形参数组

B. 在函数调用期间,形参数组的改变就是实参数组的改变

C. 通过数组名作为函数参数,可以达到返回多个值的目的

D. 在函数调用期间,形参数组和实参数组对应的是不同的数组

14. 数组名作为实参时,传递给形参的是(　　)。

A. 数组的首地址　　B. 第一个元素的值

C. 数组中全部元素的值　　D. 数组元素的个数

15. 关于全局变量和局部变量的说法,正确的是(　　)。

A. 全局变量必须在函数之外进行定义

B. 若全局变量与局部变量同名,则默认为全局变量

C. 全局变量的作用域为其所在的整个文件范围

D. 全局变量也称外部变量,仅在函数外部有效,而在函数内部无效

16. 以下程序的输出结果是(　　)。

```
int m;
void f(){
    int m = 4;
    cout << m <<" ";
}
```

```
int main(){
   f();
   cout << m << endl;
   return 0;
}
```

A. 0 4　　B. 4 0　　C. 4 4　　D. 4 随机数

17. “const int N=10;”定义了符号常量 N,以下功能相同的是(　　)。

A. ＃define N=10;　　B. ＃define N 10;

C. ＃define N 10　　D. const int N=10

18. C/C++语言的编译预处理以＃开始,其中不包括(　　)。

A. 宏定义　　B. 条件编译　　C. 文件包含　　D. 全局变量声明

19. 函数定义为 f(int &i),变量定义 int n=100,则下面调用正确的是(　　)。

A. f(10)　　B. f(10+n)　　C. f(n)　　D. f(&n)

20. 有 void f(int &a) { a=3;},则

```
int n = 1;
f(n);
cout << n << endl;
```

的结果是(　　)。

A. 1　　B. 3　　C. 0　　D. 不确定

## 二、OJ 编程题

### 1. 进制转换(HLOJ 2053)

**Problem Description**

将十进制整数 $n(-2^{31}\leqslant n\leqslant 2^{31}-1)$转换成 $k(2\leqslant k\leqslant 16)$进制数。注意,10～15 分别用字母 A、B、C、D、E、F 表示。

**Input**

首先输入一个正整数 $T$,表示测试数据的组数,然后是 $T$ 组测试数据。每组测试数据输入两个整数 $n$ 和 $k$。

**Output**

对于每组测试,先输出 $n$,然后输出一个空格,最后输出对应的 $k$ 进制数。

```
Sample Input
4
5 3
123 16
0 5
-12 2
```

```
Sample Output
5 12
123 7B
0 0
-12 -1100
```

### 2. 整数转换为字符串(HLOJ 2054)

**Problem Description**

将一个整数 $n$ 转换成字符串。例如,输入 483,应得到字符串"483"。其中,要求用一个递归函数实现把一个正整数转换为字符串。

**Input**

测试数据有多组,处理到文件尾。每组测试数据输入一个整数 $n(-2^{31}\leqslant n\leqslant 2^{31}-1)$。

**Output**

对于每组测试,输出转换后的字符串。

```
Sample Input
1234
```

```
Sample Output
1234
```

### 3. 多个数的最小公倍数(HLOJ 2093)

**Problem Description**

两个整数公有的倍数称为它们的公倍数,其中最小的一个正整数称为它们两个的最小公倍数。当然,$n$ 个数也可以有最小公倍数,例如,5,7,15 的最小公倍数是 105。

输入 $n$ 个数,请计算它们的最小公倍数。

**Input**

首先输入一个正整数 $T$,表示测试数据的组数,然后是 $T$ 组测试数据。

每组测试先输入一个整数 $n(2\leqslant n\leqslant 20)$,再输入 $n$ 个正整数($n$ 属于[1,100000])。这里保证最终的结果在 int 型范围内。

**Output**

对于每组测试,输出 $n$ 个整数的最小公倍数。

```
Sample Input
2
3 5 7 15
5 1 2 4 3 5
```

```
Sample Output
105
60
```

### 4. 互质数(HLOJ 2055)

**Problem Description**

Sg 认识到互质数很有用。若两个正整数的最大公约数为 1,则它们是互质数。要求编写函数判断两个整数是否互质数。

**Input**

首先输入一个正整数 $T$,表示测试数据的组数,然后是 $T$ 组测试数据。每组测试先输入 1 个整数 $n(1\leqslant n\leqslant 100)$,再输入 $n$ 行,每行有一对整数 $a$、$b(0<a,b<10^9)$。

**Output**

对于每组测试数据,输出有多少对互质数。

```
Sample Input
1
3
3 11
5 11
10 12
```

```
Sample Output
2
```

**5. 五位以内的对称素数(ZJUTOJ 1187)**

**Problem Description**

判断一个数是否为对称且不大于五位数的素数。要求判断对称和判断素数各写一个函数。

**Input**

测试数据有多组,处理到文件尾。每组测试输入一个正整数 $n$($0<n<2^{32}$)。

**Output**

对于每组测试,若 $n$ 是不大于五位数的对称素数,则输出"Yes",否则输出"No"。每个判断结果单独列一行。引号不必输出。

| Sample Input | Sample Output |
| --- | --- |
| 101 | Yes |

**6. 最长的单词(HLOJ 2056)**

**Problem Description**

输入一个字符串,将此字符串中最长的单词输出。要求至少使用一个自定义函数。

**Input**

测试数据有多组,处理到文件尾。每组测试数据输入一个字符串(长度不超过 80)。

**Output**

对于每组测试,输出字符串中的最长单词,若有多个长度相等的最长单词,输出最早出现的那个。这里规定,单词只能由大小写英文字母构成。

| Sample Input | Sample Output |
| --- | --- |
| Keywords insert, two way insertion sort,<br>Abstract This paper discusses three method for two way insertion | insertion<br>discusses |

**7. 按 1 的个数排序(HLOJ 2057)**

**Problem Description**

对于给定若干由 0、1 构成的字符串(长度不超过 80),要求将它们按 1 的个数从小到大排序。若 1 的个数相同,则按字符串本身从小到大排序。要求至少使用一个自定义函数。

**Input**

测试数据有多组,处理到文件尾。对于每组测试,首先输入一个整数 $n$($1\leqslant n\leqslant 100$),然后输入 $n$ 行,每行包含一个由 0、1 构成的字符串。

**Output**

对于每组测试,输出排序后的结果,每个字符串占一行。

| Sample Input | Sample Output |
| --- | --- |
| 3<br>10011111<br>00001101<br>1010101 | 00001101<br>1010101<br>10011111 |

**8. 旋转方阵(HLOJ 2058)**

**Problem Description**

对于一个奇数 $n$ 阶方阵,请给出经过顺时针方向 $m$ 次旋转后的结果。每次旋转 90 度。

**Input**

测试数据有多组,处理到文件尾。每组测试的第一行输入两个整数 $n$,$m$($1<n<20$,$1\leqslant m\leqslant 100$),接下来输入 $n$ 行数据,每行 $n$ 个整数。

**Output**

对于每组测试,输出奇数阶方阵经过 $m$ 次顺时针方向旋转后的结果。每行中各数据之间留一个空格。

```
Sample Input
3 2
4 9 2
3 5 7
8 1 6
```

```
Sample Output
6 1 8
7 5 3
2 9 4
```

**9. 求矩阵中的逆鞍点(HLOJ 2059)**

**Problem Description**

求出 $n\times m$ 二维整数数组中的所有逆鞍点。这里的逆鞍点是指在其所在的行上最大,在其所在的列上最小的元素。若存在逆鞍点,则输出所有逆鞍点的值及其对应的行、列下标。若不存在逆鞍点,则输出“Not”。要求至少使用一个自定义函数。

**Input**

测试数据有多组,处理到文件尾。每组测试的第一行输入 $n$ 和 $m$(都不大于 100),第二行开始的 $n$ 行每行输入 $m$ 个整数。

**Output**

对于每组测试,若存在逆鞍点,则按行号从小到大、同一行内按列号从小到大的顺序输出每个逆鞍点的值和对应的行、列下标,每两个数据之间一个空格;若不存在逆鞍点,则输出“Not”(引号不必输出)。

```
Sample Input
3  3
97 66 96
85 36 35
88 67 91
```

```
Sample Output
85 1 0
```

**10. 数字螺旋方阵(HLOJ 2060)**

**Problem Description**

已知 $n=5$ 时的螺旋方阵如 Sample Output 所示。输入一个正整数 $n$,要求输出 $n\times n$ 个数字构成的螺旋方阵。

**Input**

首先输入一个正整数 $T$,表示测试数据的组数,然后是 $T$ 组测试数据。每组测试输入

一个正整数 $n$($n \leqslant 20$)。

**Output**

对于每组测试,输出 $n \times n$ 的数字螺旋方阵。各行中的每个数据按 4 位宽度输出。

**Sample Input**

```
1
5
```

**Sample Output**

```
  25  24  23  22  21
  10   9   8   7  20
  11   2   1   6  19
  12   3   4   5  18
  13  14  15  16  17
```