第 3 章

函　数

C++ 继承了 C 语言的全部语法，也包括函数的定义与使用方法。在面向过程的结构化程序设计中，函数是模块划分的基本单位，是对处理问题过程的一种抽象。在面向对象的程序设计中，函数同样有着重要的作用，它是面向对象程序设计中对功能的抽象。

一个较为复杂的系统往往需要划分为若干子系统，然后对这些子系统分别进行开发和调试。高级语言中的子程序就是用来实现这种模块划分的。C 和 C++ 语言中的子程序体现为函数。通常将相对独立的、经常使用的功能抽象为函数。函数编写好以后，可以被重复使用，使用时可以只关心函数的功能和使用方法而不必关心函数功能的具体实现。这样有利于代码重用，可以提高开发效率、增强程序的可靠性，也便于分工合作和修改维护。

3.1　函数的定义与使用

此前例题中出现的 main 就是一个函数，它是 C++ 程序的主函数。一个 C++ 程序可以由一个主函数和若干子函数构成。主函数是程序执行的开始点。由主函数调用子函数，子函数还可以再调用其他子函数。

调用其他函数的函数称为**主调函数**，被其他函数调用的函数称为**被调函数**。一个函数很可能既调用别的函数又被另外的函数调用，这样它可能在某一个调用与被调用关系中充当主调函数，而在另一个调用与被调用关系中充当被调函数。

3.1.1　函数的定义

1. 函数定义的语法形式

```
类型说明符　函数名(含类型说明的形式参数表)
{
    语句序列
}
```

2. 形式参数

形式参数(简称形参)表的内容如下：

```
type1 name1, type2 name2, …, typen namen
```

type1，type2，…，type*n* 是类型标识符，表示形参的类型。name1，name2，…，name*n* 是形

参名。形参的作用是实现主调函数与被调函数之间的联系。通常将函数所处理的数据、影响函数功能的因素或者函数处理的结果作为形参。

如果一个函数的形参表为空，则表示它没有任何形参，例如此前例题中的 main 函数都没有形参。main 函数也可以有形参，其形参也称命令行参数，由操作系统在启动程序时初始化。不过命令行参数的数量和类型有特殊要求，请读者参考学生用书中本章的实验指导，尝试编写带命令行参数的程序。

函数在没有被调用的时候是静止的，此时的形参只是一个符号，它标志着在形参出现的位置应该有一个什么类型的数据。函数在被调用时才执行，也是在被调用时才由主调函数将实际参数(简称实参)赋予形参。这与数学中的函数概念相似，例如在数学中我们都熟悉这样的函数形式：

$$f(x) = x^2 + x + 1$$

这样的函数只有当自变量被赋值以后，才能计算出函数的值。

3. 函数的返回值和返回值类型

函数可以有一个返回值，函数的返回值是需要返回给主调函数的处理结果。类型说明符规定了函数返回值的类型。函数的返回值由 return 语句给出，格式如下：

```
return 表达式;
```

除了指定函数的返回值外，return 语句还有一个作用，就是结束当前函数的执行。

例如，主函数 main 的返回值类型是 int，主函数中的 return 0 语句用来将 0 作为返回值，并且结束 main 函数的执行。main 函数的返回值最终传递给操作系统。

一个函数也可以不将任何值返回给主调函数，这时它的类型标识符为 void，可以不写 return 语句，但也可以写一个不带表达式的 return 语句，用于结束当前函数的调用，格式如下：

```
return;
```

3.1.2 函数的调用

1. 函数的调用形式

在 2.2.3 小节曾经提到过，变量在使用之前需要首先声明，类似地，函数在调用之前也需要声明。函数的定义就属于函数的声明，因此，在定义了一个函数之后，可以直接调用这个函数。但如果希望在定义一个函数前调用它，则需要在调用函数之前添加该函数的**函数原型**声明。函数原型声明的形式如下：

```
类型说明符  函数名(含类型说明的形参表);
```

与变量的声明和定义类似，声明一个函数只是将函数的有关信息(函数名、参数表、返回值类型等)告诉编译器，此时并不产生任何代码；定义一个函数时除了同样要给出函数的有关信息外，主要是要写出函数的代码。

细节 声明函数时，形参表只要包含完整的类型信息即可，形参名可以省略，也就是说，原型声明的形参表可以按照下面的格式书写：

```
type1, type2, …, typen
```

但这并不是值得推荐的写法，因为形参名可以向编程者提示每个参数的含义。

如果是在所有函数之前声明了函数原型，那么该函数原型在本程序文件中任何地方都有效，也就是说在本程序文件中任何地方都可以依照该原型调用相应的函数。如果是在某个主调函数内部声明了被调函数原型，那么该原型就只能在这个函数内部有效。

声明了函数原型之后，便可以按如下形式调用子函数：

```
函数名(实参列表)
```

实参列表中应给出与函数原型形参个数相同、类型相符的实参，每个实参都是一个表达式。函数调用可以作为一条语句，这时函数可以没有返回值。函数调用也可以出现在表达式中，这时就必须有一个明确的返回值。

调用一个函数时，首先计算函数的实参列表中各个表达式的值，然后主调函数暂停执行，开始执行被调函数，被调函数中形参的初值就是主调函数中实参表达式的求值结果。当被调函数执行到 return 语句，或执行到函数末尾时，被调函数执行完毕，继续执行主调函数。

例 3-1 编写一个求 x 的 n 次方的函数。

```
//3_1.cpp
#include <iostream>
using namespace std;

//计算 x 的 n 次方
double power(double x, int n) {
    double val=1.0;
    while (n--)
        val *=x;
    return val;
}

int main() {
    cout <<  "5 to the power 2 is "<<power(5, 2)<<endl;
    //函数调用作为一个表达式出现在输出语句中
    return 0;
}
```

运行结果：

```
5 to the power 2 is 25
```

本程序中，由于函数 power 的定义位于调用之前，所以无须再对函数原型加以声明。

例 3-2 输入一个 8 位二进制数，将其转换为十进制数输出。

分析：将二进制转换为十进制，只要将二进制数的每一位乘以该位的权然后相加。例如：$(00001101)_2=0\times(2^7)+0\times(2^6)+0\times(2^5)+0\times(2^4)+1\times(2^3)+1\times(2^2)+$

$0\times(2^1)+1\times(2^0)=(13)_{10}$，所以，如果输入 1101，则应输出 13。

这里我们调用例 3-1 中的函数 power 来求 2^n。

源程序：

```
//3_2.cpp
#include <iostream>
using namespace std;

//计算 x 的 n 次方
double power(double x, int n);

int main() {
    int value=0;

    cout<<"Enter an 8 bit binary number: ";
    for (int i=7; i>=0; i--) {
        char ch;
        cin>>ch;
        if (ch=='1')
            value+=static_cast<int>(power(2, i));
    }
    cout<<"Decimal value is "<<value<<endl;
    return 0;
}

double power (double x, int n) {
    double val=1.0;
    while (n--)
        val*=x;
    return val;
}
```

运行结果：

```
Enter an 8 bit binary number: 01101001
Decimal value is 105
```

本程序中，由于 power 函数的定义位于它的调用之后，因此要事先声明 power 函数的原型。

例 3-3　编写程序求 π 的值，公式如下。

$$\pi = 16\arctan\left(\frac{1}{5}\right)-4\arctan\left(\frac{1}{239}\right)$$

其中 arctan 用如下形式的级数计算：

$$\arctan x = x-\frac{x^3}{3}+\frac{x^5}{5}-\frac{x^7}{7}+\cdots$$

直到级数某项绝对值不大于 10^{-15} 为止；π 和 x 均为 double 型。

源程序：

```
//3_3.cpp
#include <iostream>
using namespace std;

double arctan(double x) {
    double sqr=x * x;
    double e=x;
    double r=0;
    int i=1;
    while (e/i>1e-15) {
        double f=e/i;
        r=(i%4==1)?r+f : r-f;
        e=e * sqr;
        i+=2;
    }
    return r;
}

int main() {
    double a=16.0 * arctan(1/5.0);
    double b=4.0 * arctan(1/239.0);
    //注意：因为整数相除结果取整，如果参数写为 1/5,1/239,结果就都是 0
    cout<<"PI="<<a-b<<endl;
    return 0;
}
```

运行结果：

```
PI=3.14159
```

例 3-4 寻找并输出 11～999 之间的数 m，它满足 m，m^2 和 m^3 均为回文数。

所谓回文数是指其各位数字左右对称的整数。例如：121，676，94 249 等。满足上述条件的数如 $m=11$，$m^2=121$，$m^3=1331$。

分析：判断一个数是否是回文，可以用除以 10 取余的方法，从最低位开始，依次取出该数的各位数字，然后用最低位充当最高位，按反序重新构成新的数，与原数比较是否相等，若相等，则原数为回文。

源程序：

```
//3_4.cpp
#include <iostream>
using namespace std;
```

```
//判断 n 是否为回文数
bool symm(unsigned n) {
    unsigned i=n;
    unsigned m=0;
    while (i>0) {
        m=m*10+i%10;
        i/=10;
    }
    return m==n;
}

int main() {
    for (unsigned m=11; m<1000; m++)
        if (symm(m) && symm(m*m) && symm(m*m*m)) {
            cout<<"m="<<m;
            cout<<"  m*m="<<m*m;
            cout<<"  m*m*m="<<m*m*m<<endl;
        }
    return 0;
}
```

运行结果：

```
m=11   m*m=121    m*m*m=1331
m=101  m*m=10201  m*m*m=1030301
m=111  m*m=12321  m*m*m=1367631
```

例 3-5　计算如下公式，并输出结果。

$$k=\begin{cases}\sqrt{\sin^2 r+\sin^2 s} & (r^2\leqslant s^2)\\ \dfrac{1}{2}\sin(rs) & (r^2>s^2)\end{cases}$$

其中 r,s 的值由键盘输入。$\sin x$ 的近似值按如下公式计算：

$$\sin x=\frac{x}{1!}-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\cdots=\sum_{n=1}^{\infty}(-1)^{n-1}\frac{x^{2n-1}}{(2n-1)!}$$

计算精度为 10^{-6}，当某项的绝对值小于计算精度时，停止累加，累加和即为该精度下的 sin x 的近似值。

源程序：

```
//3_5.cpp
#include <iostream>
#include <cmath>   //头文件 cmath 中具有对 C++标准库中数学函数的说明
using namespace std;

const double TINY_VALUE=1e-10;
```

```
double tsin(double x) {
    double g=0;
    double t=x;
    int n=1;
    do {
        g+=t;
        n++;
        t=-t*x*x/(2*n-1)/(2*n-2);
    } while (fabs(t)>=TINY_VALUE);
    return g;
}

int main() {
    double k, r, s;
    cout<<"r=";
    cin>>r;
    cout<<"s=";
    cin>>s;
    if (r*r<=s*s)
        k=sqrt(tsin(r)*tsin(r)+tsin(s)*tsin(s));
    else
        k=tsin(r*s)/2;
    cout<<k<<endl;
    return 0;
}
```

运行结果：

```
r=5
s=8
1.37781
```

本程序中用到了两个标准 C++ 的系统函数——绝对值函数 double fabs(double x)和平方根函数 double sqrt(double x)，它们的原型都在 cmath 头文件中定义。3.5 节将专门介绍标准 C++ 的系统函数。

例 3-6 投骰子的随机游戏。

游戏规则是：每个骰子有 6 面，点数分别为 1,2,3,4,5,6。游戏者在程序开始时输入一个无符号整数，作为产生随机数的种子。

每轮投两次骰子，第一轮如果和数为 7 或 11 则为胜，游戏结束；和数为 2,3 或 12 则为负，游戏结束；和数为其他值则将此值作为自己的点数，继续第二轮、第三轮…直到某轮的和数等于点数则取胜，若在此前出现和数为 7 则为负。

由 rollDice 函数负责模拟投骰子、计算和数并输出和数。

提示 系统函数 int rand(void)的功能是产生一个伪随机数，伪随机数并不是真正随机的。这个函数自己不能产生真正的随机数。如果在程序中连续调用 rand，期望由此可

以产生一个随机数序列，你会发现每次运行这个程序时产生的序列都是相同的，这称为伪随机数序列。这是因为函数 rand 需要一个称为“种子”的初始值，种子不同，产生的伪随机数也就不同。因此只要每次运行时给予不同的种子，然后连续调用 rand 便可以产生不同的随机数序列。如果不设置种子，rand 总是默认种子为 1。不过设置种子的方法比较特殊，不是通过函数的参数，而是在调用它之前，需要首先调用另外一个函数 void srand (unsigned int seed)为其设置种子，其中的参数 seed 便是种子。

源程序：

```
//3_6.cpp
#include <iostream>
#include <cstdlib>
using namespace std;

//投骰子,计算和数,输出和数
int rollDice() {
    int die1=1+rand()%6;
    int die2=1+rand()%6;
    int sum=die1+die2;
    cout<<"player rolled "<<die1<<"+"<<die2<<"="<<sum<<endl;
    return sum;
}

enum GameStatus {WIN, LOSE, PLAYING};

int main() {
    int sum, myPoint;
    GameStatus status;

    unsigned seed;
    cout<<"Please enter an unsigned integer: ";
    cin>>seed;                    //输入随机数种子
    srand(seed);                  //将种子传递给 rand()

    sum=rollDice();               //第一轮投骰子,计算和数
    switch (sum) {
    case 7:                       //如果和数为 7 或 11 则为胜,状态为 WIN
    case 11:
        status=WIN;
        break;
    case 2:                       //和数为 2,3 或 12 则为负,状态为 LOSE
    case 3:
    case 12:
        status=LOSE;
```

```
        break;
    default:  //其他情况,游戏尚无结果,状态为 PLAYING,记下点数,为下一轮做准备
        status=PLAYING;
        myPoint=sum;
        cout<<"point is"<<myPoint<<endl;
        break;
    }

    while (status==PLAYING) {          //只要状态仍为 PLAYING,就继续进行下一轮
        sum=rollDice();
        if (sum==myPoint)              //某轮的和数等于点数则取胜,状态置为 WIN
            status=WIN;
        else if (sum==7)               //出现和数为 7 则为负,状态置为 LOSE
            status=LOSE;
    }

    //当状态不为 PLAYING 时上面的循环结束,以下程序段输出游戏结果
    if (status==WIN)
        cout<<"player wins"<<endl;
    else
        cout<<"player loses"<<endl;

    return 0;
}
```

运行结果 1：

```
Please enter an unsigned integer: 8
player rolled 5+1=6
point is 6
player rolled 6+6=12
player rolled 6+4=10
player rolled 6+6=12
player rolled 6+6=12
player rolled 3+2=5
player rolled 2+2=4
player rolled 3+4=7
player loses
```

运行结果 2：

```
Please enter an unsigned integer: 23
player rolled 6+3=9
point is 9
player rolled 5+4=9
player wins
```

2. 嵌套调用

函数允许嵌套调用。如果函数 1 调用了函数 2，函数 2 再调用函数 3，便形成了函数的嵌套调用。

例 3-7　输入两个整数，求它们的平方和。

分析：虽然这个问题很简单，但是为了说明函数的嵌套调用问题，在这里设计两个函数：求平方和函数 fun1 和求一个整数的平方函数 fun2。由主函数调用 fun1，fun1 又调用 fun2。

源程序：

```
//3_7.cpp
#include <iostream>
using namespace std;

int fun2(int m) {
    return m * m;
}

int fun1(int x,int y) {
    return fun2(x)+fun2(y);
}

int main() {
    int a, b;
    cout<<"Please enter two integers(a and b): ";
    cin>>a>>b;
    cout<<"The sum of square of a and b: "<<fun1(a, b)<<endl;
    return 0;
}
```

运行结果：

```
Please enter two integers(a and b): 3 4
The sum of square of a and b: 25
```

图 3-1 说明了例 3-7 函数的调用过程，图中标号标明了执行顺序。

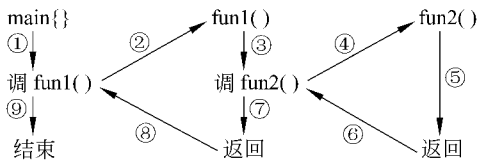

图 3-1　例 3-7 函数的调用过程

3. 递归调用

函数可以直接或间接地调用自身，称为递归调用。