第 5 章　模块化程序设计——函数

5.1　求给定三角形的重心——模块化程序设计

【例 5.1】 求给定三角形的重心。

解：三角形的重心是三条中线的交点。如图 5.1 所示，在直角坐标系下，该题目应该按图 5.2 所示的 PAD 求解。

在图 5.2 的 PAD 中，输入和输出很简单，下面求精“求中线”和“求交点”。

求中线 AD 和 BE 的算法是一样的，以 AD 为例：求中线 AD 应该先求 BC 边的中点 D，然后求过 A、D 两点的直线方程。该过程描述为图 5.3 的 PAD。

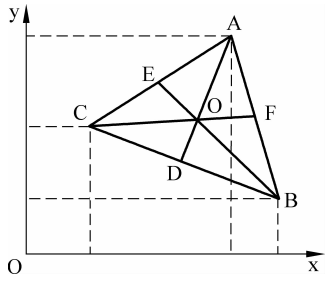

图 5.1　求三角形重心

图 5.2　求三角形重心的 PAD

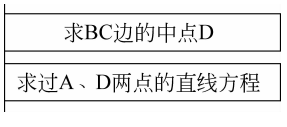

图 5.3　求中线 AD

求 BC 边的中点 D，只是两个公式计算：xd＝(xb＋xc)/2；yd＝(yb＋yc)/2。

求过 A、D 两点的直线方程也只是两个公式计算：a1＝(ya－yd)/(xa－xd)；b1＝ya－a1 * xa。这里不失一般性假设，所需直线都可以用斜截式方程表示。

求 AD、BE 交点 O，就是解方程组：

$$y = a1 * x + b1$$
$$y = a2 * x + b2$$

也只是两个公式计算：xo＝(b2－b1)/(a1－a2)；yo＝a1 * xo＋b1。

实际计算过程应该按图 5.4 的 PAD 进行。

按以前所学的知识，编出程序如下：

```
#include <stdio.h>                          //括入标准输入输出函数库头文件
int main(void) {                            //主函数
    float xa,ya,xb,yb,xc,yc;                //分别保存三角形三个顶点的 X、Y 方向坐标
```

```
    float xd,yd,xe,ye;              //分别表示中点 D、E 坐标
    float a1,b1,a2,b2;      //分别表示中线 AD、BE 的方程系数
    float xo,yo;                    //重心 O 的坐标
    printf("please input xa,ya,xb,yb,xc,yc:\n");
                                    //输入三个点 X、Y 轴坐标
    scanf("%f%f%f%f%f%f",&xa,&ya,&xb,&yb,&xc,&yc);
    xd=(xb+xc)/2;               //求 BC 边的中点 D
    yd=(yb+yc)/2;
    a1=(ya-yd)/(xa-xd);         //求过 A、D 两点的直线方程
    b1=ya-a1*xa;
    xe=(xa+xc)/2;               //求 AC 边的中点 E
    ye=(ya+yc)/2;
    a2=(yb-ye)/(xb-xe);         //求过 B、E 两点的直线方程
    b2=yb-a2*xb;
    xo=(b2-b1)/(a1-a2);         //求 AD、BE 交点 O
    yo=a1*xo+b1;
    printf("重心坐标：x=%.3f y=%.3f \n", xo, yo);
                                    //打印输出
    return 0;
}
```

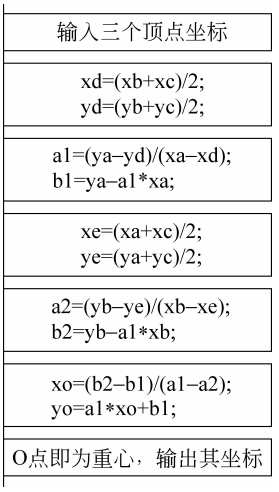

图 5.4 求三角形重心的完整 PAD

观察该程序，“求 BC 边的中点 D”和“求 AC 边的中点 E”的两段程序是一样的，在程序中重复写了一次。再回顾第 2 章例 2.4 的程序，在计算四边形周长时，计算四个边长的程序也是一样的，在程序中重复写了四次。

一般程序设计语言都为这种“计算过程一致，而参与运算的数据不同”的情况，提供一种机制——子程序。在 C 中子程序体现为**函数**(function)。下面用函数来重写例 5.1 的程序。

【例 5.2】 用函数重新写例 5.1 的程序。

```
#include <stdio.h>                  //括入标准输入输出函数库头文件          //L1
float a;                            //全局量 a 传递直线方程的斜率            //L2
/*求中线：参数：三角形三个顶点 r、s、t 的 x、y 坐标*/                        //L3
float lines(float xr, float yr, float xs, float ys, float xt, float yt){    //L4
    float xu,yu;                    //中点 u 坐标                          //L5
    xu=(xs+xt)/2;                   //求 st 边的中点 u                     //L6
    yu=(ys+yt)/2;                                                          //L7
    //求过 r、u 两点的直线方程                                             //L8
    a=(yr-yu)/(xr-xu);              //计算系数 a                           //L9
    return yr-a*xr;                 //计算系数 b,并带着 b 值返回           //L10
}                                                                          //L11
int main(void) {                    //主函数                               //L12
    float xa,ya,xb,yb,xc,yc;        //分别保存三角形三个顶点的 X、Y 方向坐标 //L13
    float xd,yd,xe,ye;              //分别表示中点 D、E 坐标               //L14
    float a1,b1,a2,b2;              //分别表示中线 AD、BE 的方程系数临时变量 //L15
```

```
    float xo,yo;                    //重心 O的坐标                          //L16
    //输入三个点的 X、Y 方向坐标 346 360 416 108 116 212                    //L17
    printf("please input xa,ya,xb,yb,xc,yc:\n");                            //L18
    scanf("%f%f%f%f%f%f",&xa,&ya,&xb,&yb,&xc,&yc);                          //L19
    b1=lines(xa,ya,xb,yb,xc,yc);                //求 BC边的中线 AD          //L20
    a1=a;                                                                   //L21
    b2=lines(xb,yb,xa,ya,xc,yc);                //求 AC边的中线 BE          //L22
    a2=a;                                                                   //L23
    xo=(b2-b1)/(a1-a2);                         //求 AD、BE交点 O           //L24
    yo=a1*xo+b1;                                                            //L25
    printf("重心坐标: x=%.3f y=%.3f \n",xo,yo);  //打印输出                 //L26
    return 0;                                                               //L27
}                                                                           //L28
```

对照例 5.2 与例 5.1 的程序。例 5.2 引进函数计算中线，当具体计算某条中线时，调用该函数。程序显得干净、利索、清晰，即好读又好看，并且与原始问题有相当高的可对照性。因为这里使用了“子程序”技术。

这就是模块化程序。以模块为指导思想的程序设计过程称**模块化程序设计**。狭义地讲，模块化程序设计依赖于子程序，每个模块是一个子程序。在 C 中子程序体现为函数，程序的每个模块是一个函数。

5.2　函　　数

在例 5.2 的程序中：第 4～11 行是一个函数定义，定义函数 lines。它有 6 个自变量 xr、yr、xs、ys、xt、yt，函数的自变量称为**形式参数**(formal parameter)，简称**形参**。

第 20、22 行的

lines(xa,ya,xb,yb,xc,yc)

lines(xb,yb,xa,ya,xc,yc)

调用函数 lines，称为**函数调用**(function call)。函数调用 lines(xa,ya,xb,yb,xc,yc)计算当自变量 xr、yr、xs、ys、xt、yt 取 xa,ya,xb,yb,xc,yc 值时，函数 lines 的值。在计算函数值时，替换自变量的部分称为**实际参数**(actual parameter)，简称**实参**。这个计算过程称为“以 xa,ya,xb,yb,xc,yc 作实参调用函数 lines”。

在函数 lines 内：

- 第 4 行 float lines(float xr,float yr,float xs,float ys,float xt,float yt)称为**函数定义说明符**。
  - ✓ float 定义本函数的类型为浮点型；
  - ✓ lines 是函数名；
  - ✓ 括号部分称为形式参数表。

在形式参数表中

- ✓ xr、yr、xs、ys、xt、yt 为形参；

✓ 每个形参类型为浮点类型。

- 从第 4 行的“{”到第 11 行的“}”，是函数 lines 的**函数体**(function body)。函数体由一个复合语句构成，为函数的操作部分。具体规定函数 lines 的操作及其值的计算。
- 在函数体内，第 10 行“return b;”是返回语句，它带着 return 后边表达式的值作为函数值返回。

程序例 5.1 的执行过程是：

(1) 从第 18 行主函数中 printf("please input xa,ya,xb,yb,xc,yc:\n");(13～17 行是变量声明和注释)开始执行。

输出一行提示信息：

```
please input xa,ya,xb,yb,xc,yc :
```

(2) 执行第 19 行函数调用 scanf("%f%f%f%f%f%f ",&xa,&ya,&xb,&yb,&xc,&yc);等待操作员输入数值。假设操作员输入：

```
346 360 416 108 116 212
```

当操作员输入并回车之后，变量 xa、ya、xb、yb、xc、yc 分别取得相应值。

(3) 执行第 20 行的表达式语句，首先分别以 xa、ya、xb、yb、xc、yc 为实际参数调用函数 lines，计算出当自变量取 346 360 416 108 116 212 时 lines 的函数值。调用函数 lines 的执行过程是：

① 分别计算 xa，ya，xb，yb、xc、yc 的值，得 346、360、416、108、116、212。

② 将 346、360、416、108、116、212 分别送入 lines 的形式参数 xr、yr、xs、ys、xt、yt 中，这些变量分别有给定的值。

③ 进入函数 lines，执行 lines 的操作部分。执行第 6 行，为 xu 赋值右端表达式值，为 266。

④ 执行第 7 行，为 yu 赋值右端表达式值，为 160。

⑤ 执行第 9 行，为 a 赋值右端表达式值，为 2.5。

⑥ 执行第 10 行返回语句 return yr－a * xr。

- 计算出表达式 yr－a * xr 的值为 －505。
- 带着函数值(－505)返回调用处：主程序第 20 行。

(4) 在主程序第 20 行的表达式语句内，用 lines 带回的(－505)给 b1 赋值；b1 的值为(－505)。

(5) 执行第 18 行，为 a1 赋值，a1 的值为函数 lines 中计算出的 a 值，a1 的值为 2.5。

(6) 执行第 22 行的表达式语句，首先分别以 xb、yb、xa、ya、xc、yc 为实际参数调用函数 lines，计算出当自变量取 416、108、346、360、116、212 时 lines 的函数值。调用函数 lines 的执行过程与步骤 3 相同，这里不再赘述。从函数返回后为 b2 赋值：508.259，a 的值为(－0.962)。

(7) 执行第 23 行,为 a2 赋值,a2 的值为函数 lines 中计算出的 a 值,a2 的值为 (−0.962)。

(8) 执行第 24、25 行,求 AD、BE 交点 O,计算出: xo=292.667;yo=226.667。

(9) 执行第 26 行,输出函数 printf 在屏幕显示:

```
重心坐标: x=292.667  y=226.667
```

5.2.1　函数定义

除标准库函数外,程序中使用函数必须先定义,然后才可以用“函数调用”调用它。函数定义的形式如下所示。

函数声明

```
TT F (T id, T id, ..., T id){
    ⋮
}
```

其中:

- TT 是类型说明符,具体说明函数的类型;
- F 是函数名字;
- (T id,T id,…,T id)是形参列表,具体说明本函数的各个形式参数;
- { ...}是复合语句,具体规定本函数的操作。

1. 函数类型

函数类型(function type)指明所定义的函数的结果类型。

缺省结果类型为“int”类型,结果类型不能是数组类型、函数类型。

有些函数是无值的,也可以说是“无类型”的,这可能是问题的算法本身决定的。无类型函数在函数定义时,其结果类型为空类型符“void”。如图 1.11 中的 hello 函数。

2. 参数列表

函数定义说明符的**参数列表**(parameter-list)由一个个**参数声明**(parameter-declaration)组成,各个参数声明之间以逗号“,”分隔。每个参数声明具体说明一个形式参数的特性(类型),形式如下:

函数形参列表

```
(T id, T id,..., T id)
```

其中:

- id 是标识符,为一个形式参数(简称“形参”);
- T 是类型说明,它指出紧跟其后的形式参数 id 的类型。

C 允许使用无参函数,无参函数的参数列表为空,或使用“空类型”的类型说明符“void”。

C 参数种类十分简单,只有值参。值参表示形参是一个局部于函数的变量;当

调用函数时，把实参值复制到形参变量中，函数内部的运算则是在形参上操作，不影响实参。

3. 复合语句

复合语句(compound-statement)在第 2 章已经介绍过，由若干声明和语句组成。其声明部分具体说明本函数内使用的其他量；语句部分规定在本函数中要执行的算法动作，即描述本函数的具体实现算法。

5.2.2 函数调用

在 C 中，当调用一个函数时，

- 首先顺序计算函数调用时**实参表**(actual-parameter-list)中各实参的值；
- 其次把各个实参值转换成形参表中相应形参的类型；
- 然后把这些转换后的值顺序传入形参表的相应形参中去；
- 最后进入函数执行复合语句。

这与数学中计算一个函数值十分类似，首先用一些值替换函数定义中的函数自变量，然后再计算函数值。

函数调用

```
F(U, U,..., U)
```

其中：

- F 是函数标识符，是欲调用函数的名字；
- 每个 U 都是表达式，分别为一个实参；
- 实参表 U,U,…,U 列出调用函数时传入相应函数中的信息。若为无参函数，该部分可空。

函数调用的目的是计算一个函数值，然后将这个值用于表达式并参与进一步的运算及操作，调用标准函数也是函数调用。如例 5.2 程序中的

```
lines(xa,ya,xb,yb,xc,yc);
lines(xb,yb,xa,ya,xc,yc);
printf("please input xa,ya,xb,yb,xc,yc:\n");
scanf("%f%f%f %f%f%f",&xa,&ya,&xb,&yb,&xc,&yc);
```

等都是函数调用。

调用函数时，形参用实参带进来的信息参与进一步的运算。调用函数的执行过程如图 5.5 所示。首先保存主程序当前运行环境状态；然后为被调用函数开辟运行空间，形实参结合，执行被调用函数体，得到函数值(可能没有)；最后被调用函数返回，释放其运行空间，并将函数运行结果返回到调用点；恢复函数调用前的运行环境，继续执行下边的代码。

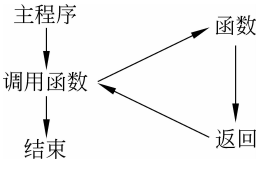

图 5.5 调用函数的执行过程

1. 函数返回

函数调用进入函数执行后，到一定步骤应该返回到调用处。C 函数返回有两种方式：

- 函数运行到复合语句末尾。则当函数执行到复合语句末尾（最后那个闭花括号“}”）后，既返回到调用处。这种返回方式适用于函数返回类型是“空类型”即“void”。
- 执行返回语句。

函数返回语句

```
return;
```

或

```
return e;
```

其中：

- return 是保留字，标明是函数返回语句；
- return；适合无返回类型的函数，一般无返回类型函数原型为 void f(…)；
- return e；适合有返回类型的函数，e 是表达式，是函数的返回值；e 的类型要与函数返回类型赋值兼容。

2. 函数值

如果函数有值，应该把函数值带回调用处。C 使用带表达式的返回语句向调用函数的主程序传递函数值。为了将函数值传回函数调用处（为了带回一个值），在复合语句中，应该至少有一个（当然可以有多个）带表达式的返回语句。并且，当函数返回时，必须执行某个带表达式的返回语句。带表达式的返回语句的执行过程是：

- 计算表达式的值；
- 把表达式值转换成函数的结果类型；
- 用类型转换后的值作为函数值，并带着它返回到调用该函数处。在函数调用处，作为运算分量，参与进一步运算。

如果函数执行不带表达式的返回语句；或没有执行返回语句，而是执行到复合语句最后，遇到闭花括号“}”，才返回到函数调用处。这两种情况函数都无值可以带回。如果是无类型函数，在函数调用处不需要函数值，这种返回是正常的；如果是有类型函数，在函数调用处极可能正需要函数值参与进一步运算，这将带来不可预料的结果，读者一定注意。

5.2.3　函数原型

【例 5.3】 用函数重新写第 4 章例 4.11，打印 100 以内素数。

```
/* PROGRAM writeprime */
#include <stdio.h>
bool prime(int n) {
    int j;
```

```
    for (j=n/2; j >=2; j--)
        if (n%j==0)  return false;
    return true;
}
int main(void) {
    int i;
    for (i=2;i <=100; i++)
        if (prime(i))
            printf("%5d\n", i);
    return 0;
}
```

对于这个例子，需要读者注意的是代码第5～7行的结构和例4.11的区别。

例4.11代码如下：

```
for (j=i/2; j >=2; j--)
    if (i %j==0) flag=false;
```

这段代码是要j在[2,i/2]的整数闭区间遍历一次，对每一个数都进行i%j的运算，判断余数是否为零。例如对于整数12来说，这个循环j是要从6变到2，进行5次求余运算；才会得到结果flag为假。

例5.3的代码是

```
for (j=n/2; j >=2; j--)
    if (n%j==0)  return false;
return true;
```

这段代码的意义是j在[n/2,2]之间的整数闭区间遍历，如果遇到某一个j使得i%j的运算结果为0，则函数退出，返回false作为函数值；如果整个区间都遍历，没有j使得i%j的运算结果为0，说明是素数，返回true。同样是整数12，遇到6即可整除，循环只执行1次就结束，函数返回false。

请读者仔细推敲例4.11和例5.3在素数判断上的不同。

前边讲述的程序例子，从行文上看，任何函数的函数调用都在相应函数定义之后，这是有意安排的。因为C规定任何标识符都必须先声明后使用，但不是所有程序都能做到这点，有可能有一些函数的调用在其定义之前出现。有些即使能做到这点，程序也不清晰，例如上述例5.3的程序。

为了解决这个问题，C引进“**函数原型**(function prototype)”的概念。函数原型放在函数调用之前，先声明相应函数的特性，满足了C标识符先定义后使用的要求。这样相应函数的定义就可以放在任何位置了。例5.3程序使用函数原型可以写成如下例5.4的样子。

【例5.4】 引进函数原型重写例5.3程序，打印100以内素数。

```
/* PROGRAM writeprime */
#include <stdio.h>
```

```
bool prime(int);                    //函数原型,说明标识符 prime
int main(void) {
    int i;
    for (i=2;i <=100; i++)
        if (prime(i))
            printf("%5d\n", i);
    return 0;
}
bool prime(int n) {
    int j;
    for (j=n/2; j >=2; j--)
        if (n%j==0)   return false;
    return true;
}
```

函数原型形式

```
TT F (T; T; ...; T);
```

或

```
TT F (T id; T id; ...; T id);
```

其中：

- TT 是类型说明符,具体说明函数的类型；
- F 是函数名字；
- T 形参类型,id 是形参名字。

函数原型与函数定义的首行十分类似,可以说把函数定义中的复合语句换成分号“;”就是函数原型。一般情况下,函数原型使用第一种形式。第二种形式的函数原型,相应参数标识符也不起作用,因为函数原型只需要说明参数个数和每个参数的特性,而不关心相应参数是什么名字。如下两个函数原型等价：

float lines(float,float,float,float,float,float);

float lines(float xr,float yr,float xs,float ys,float xt,float yt);

5.3　程序设计实例

【例 5.5】 用函数重新写第 2 章例 2.4 的程序,求植树棵数。

```
#include <stdio.h>                          //括入标准输入输出函数库头文件
#include <math.h>                           //括入标准数学函数库头文件
/*计算 r、s 两点距离：参数：r 点 x、y 坐标,s 点 x、y 坐标*/
float lines(float xr,float yr,float xs,float ys){
    return sqrt((xr-xs)*(xr-xs)+(yr-ys)*(yr-ys));
}
```

```
int main(void) {                                    //主函数
    float xa,ya,xb,yb,xc,yc,xd,yd;                  //分别保存四个点的 X、Y 方向坐标
    float ab, bc, cd, da;                           //分别表示四边形的四条边边长
    float s;
    int m;                                          //计算用变量;ss 表示周长;m 表示植树棵树
    //输入四个点的 X、Y 方向坐标
    printf("please input xa,ya,xb,yb,xc,yc,xd,yd:\n");
    scanf("%f%f%f%f%f%f%f%f",&xa,&ya,&xb,&yb,&xc,&yc,&xd,&yd);
    //计算边长
    ab=lines(xa,ya,xb,yb);                          //边 AB 长
    bc=lines(xb,yb,xc,yc);                          //边 BC 长
    cd=lines(xc,yc,xd,yd);                          //边 CD 长
    da=lines(xd,yd,xa,ya);                          //边 DA 长
    s=ab+bc+cd+da;
    m=s/2;                                          //计算总植树棵数
    printf("总植树棵数：%10d\n",m);                  //打印输出
    return 0;
}
```

【例 5.6】 已知玉米每亩产量 650 公斤。如图 5.6 所示，现有一个近似四边形的地块位于南北方向路东侧，东西方向路北侧。其一个顶点距离南北方向路 547 米，距离东西方向路 411 米；另一个顶点距离南北方向路 804 米，距离东西方向路 77 米；第三个顶点距离南北方向路 39 米，距离东西方向路 208 米；第四个顶点距离南北方向路 116 米，距离东西方向路 332 米。若该地块种植玉米，求该地块玉米产量。

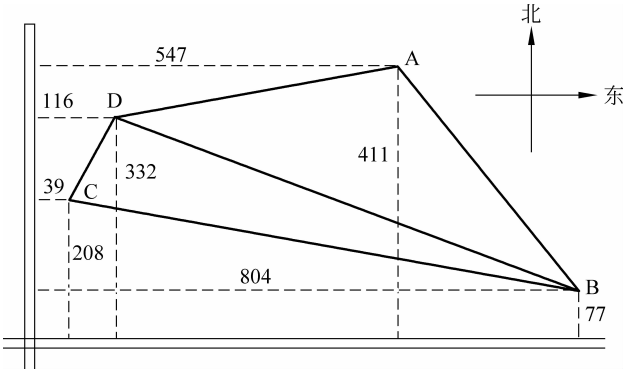

图 5.6 四边形地块

解：求总产量，应该用总面积乘以单位面积产量，关键问题是怎样求总面积。把南北方向路定义为 Y 方向坐标轴，北为正；把东西方向路定义为 X 方向坐标轴，东为正；把四个顶点分别记为 A、B、C、D；把四个顶点到路的距离分别定义为相应点的坐标值，如图 5.6 所示。其中，“折合成亩”和“求总产量”两步计算很简单。

计算地块面积。根据数学知识，可以把四边形的 B、D 两顶点相连，构成两个三角形，然后分别计算两个三角形面积并相加，由于计算三角形 ABD 和 BCD 的过程一样，不同的