第 5 章　分而治之——模块化程序设计

学习目标：


电子教案

- 理解模块化程序设计的基本思想。
- 理解函数的定义、函数原型的概念。
- 掌握函数调用的方法。
- 理解函数调用的过程和函数参数传递机制。
- 学会使用标准库中的函数。

迄今为止，我们已经学习了三种控制程序的结构：顺序结构、选择结构和循环结构。如果能灵活运用这三种控制结构，使用堆叠嵌套技术，毫不夸张地说，你已经可以解决绝大多数问题了。前几章解决的问题都相对比较简单，问题的求解算法都比较明显，写出的程序一般是十几行，最多不过几十行，大多还不到一页纸的长度。但是很多问题往往求解算法比较复杂，常常要采用自顶向下、逐步求精的方法确定出算法，而且有很多问题的实现代码可能多达几百行，几千行，甚至几万行。这种规模的代码还能像前几章那样都写在一个 main 函数里吗？回答应该是可以！但是，设想一下成千上万行代码都写在一个 main 里，该有多么难以控制啊！

一个规模比较大的问题，往往都要分解成若干个相对比较小的子问题，每个子问题还可以再分成若干个更小的子问题，这个过程是自顶向下、逐步求精的过程，而且是逐层进行的。自顶向下、逐步求精的过程为模块化实现提供了具体的方法。每个子问题都可以对应一个独立的功能模块（函数）。要求解的问题经过模块化之后，在主函数 main 里只包含顶层分解的模块函数调用，每个被调用的模块都独立于主函数之外，作为主函数的工具。

三种程序控制结构与模块化相结合才是真正的结构化程序设计。

本章详细讨论如何建立**函数模块**，如何使用函数模块，函数模块之间是如何联系在一起的，如何有效地管理众多的函数模块——**文件模块**，如何建立自己的**函数库**，定义用户使用的**接口**，比较系统地研究一下模块化程序设计的基本方法。

本章要解决的问题：

- 再次讨论猜数游戏模拟问题；
- 是非判断问题；
- 递归问题；
- 简单的计算机绘图问题；
- 学生成绩管理系统初步。


视频

5.1　再次讨论猜数游戏模拟问题

问题描述：

问题同 4.7 节的问题描述，这里略。输入输出样例也请参考 4.7 节。

问题分析：

在4.7节已经采用自顶向下、逐步求精的方法研究了这个问题。已经认识到要解决一个比较复杂的问题，不可能一开始就确定一个十分精细的解决方案，一般要经历一个从抽象到具体，逐步明晰的过程。开始先勾画出求解方案的一个比较粗糙的轮廓，确定一个比较抽象的概念，然后再逐步细化，把抽象的东西逐渐细化到可以实现的具体步骤。自顶向下、逐步求精的过程是一层层分解的过程。如果设猜数游戏模拟的顶层是问题的原始抽象描述，经过第一次分解问题变成了两个子问题。

主模块算法：

① 让计算机“想”一个数；

② “猜数”过程模拟；

③ 问是否继续玩，回答y/Y，返回到①，否则程序结束。

其中的①和②的每一步还都比较抽象，但问题已经被模块化，即整个问题的求解分成两个子模块①和②，分别命名为makeMagic和guessNumber。makeMagic模块的功能是计算机“想”一个数，guessNumber模块的功能就是模拟一次猜数游戏的全过程，直到猜中为止。至于makeMagic怎么想，guessNumber怎么猜现在先不用考虑，可以假设它们都已经解决了。这样，main函数就变得非常简单，就是顺序的调用makeMagic模块和guessNumber模块。这个比较粗糙/抽象的主模块算法，如果用流程图表示则称为**主流程**，如图5.1所示。如果把main函数认为是一层，两个子问题对应的子模块认为是第二层，这样模块之间形成了一个层次结构，如图5.2所示。这个层次结构表明了主函数和子模块之间的层次调用关系，图中单向的箭头线表示调用，意识是猜数游戏main函数调用makeMagic和guessNumber模块。

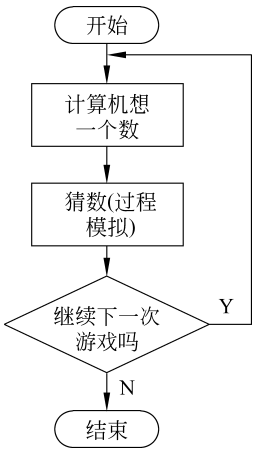

图5.1 猜数游戏的主流程

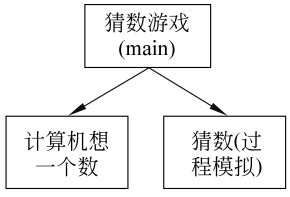

图5.2 猜数游戏程序的层次结构

主函数模块的实现见程序清单5.1。

程序清单 5.1

```
#001 guessnumberNew.c
#002 int main(void){
#003     char a;
```

源码 5.1

```
#004    int magic;
#005    srand(time(NULL));          //设置随机数的种子
#006    printf("Welcome to GuessNumber Game\n");
#007    do{
#008        magic=makeMagic();
#009        printf("I have a magic number between 1 to 1000, please guess:");
#010        guessNumber(magic);
#011        printf("Continue or no? Y/N\n");
#012        a=getch();              //输入一个字符赋给 a
#013    }while( a=='Y' || a=='y');
#014    return 0;
#015 }
```

其中,两个关键步骤#008行和#010行分别调用了函数makeMagic()和guessNumber(magic),在这里不管它们是如何实现的,只关心它们的功能,它们的实现代码是在main函数之外的其他某个地方。

主流程和层次结构清楚之后,接下来就可以进一步研究每个子问题(即每个模块)的解决方法了。如果子问题还很复杂、抽象,那就还要把每个子问题再次进行分解,每个分解出来的子子问题又对应一个更小的模块,它们合起来构成本模块的解决方案。如果经过第一次划分之后的子问题已经很容易求解了,就直接设计这个子问题的详细算法即可。对于猜数游戏模拟问题,第一次分解求精得到的模块"让计算机想一个数"已经比较简单了,可以直接给出它的算法如下。

makeMagic 模块算法

1.1 使用随机函数rand()产生一个1～1000的数number

对于猜数过程模拟模块,第二步判断是否猜中,又加细求精了一次,也可以做成一个第三层的模块,由于它过于简单,所以就把它细化在猜数模块中展开了,即

guessNumber 模块算法

1.2.1 接收用户猜的数guess

1.2.2.1 如果guess>number,提示too high返回到1.2.1

1.2.2.2 如果guess<number,提示too low返回到1.2.1

1.2.2.3 如果猜中,转到1.2.3

1.2.3 输出祝贺信息

在C/C++中,每个功能模块的实现程序用自定义的"函数"表示,其形式与main类似,但是各自都有自己的名字。这两个函数的定义如下:

makeMagic 函数定义:


源码

```
#001 /*
#002 * 函数功能: 产生一个 1~1000 的随机数,并反馈已经产生的信息
#003 * 入口参数: 无
#004 * 返回值: 计算机"想"好的随机整数
#005 */
#006  int makeMagic(void){
#007    int magicNumber;
```

```
#008     magicNumber=rand()%1000 +1;   //产生随机数
#009     return magicNumber;
#010  }
```

guessNumber 函数定义：

```
#001 /*
#002 * 函数功能：猜数过程模拟
#003 * 入口参数：整型 magic,计算机想好的数
#004 * 返回值：无
#005 */
#006 void guessNumber(int magic){
#007     int guess;
#008     do{
#009        scanf("%d",&guess);
#010        if(guess >magic)
#011              printf("Wrong, too high! try again!\n");
#012        else if(guess <magic)
#013              printf("Wrong, too low! try again!\n");
#014     }while(guess <magic || guess >magic);
#015     printf("Congratulation! You are right!\n");
#016 }
```

关于自定义函数的具体细节在本节进行详细讨论。

5.1.1 模块化思想

现在我们总结一下刚才的讨论。在自顶向下、逐步求精的过程中，最初是比较粗糙的算法，把比较抽象的猜数游戏模拟归结为两个子问题，每个子问题对应一个模块。如果每个子问题能够很容易地得到解决，整个问题便得到了解决，如果子问题还比较抽象，就继续拆分成更小的子问题，又产生很多更小的模块，以此类推，直到很具体地能够解决为止。这个从抽象到具体的过程**蕴藏着一种层次结构、模块结构**，它所体现的就是模块化程序设计的基本思想，它是解决复杂问题或大规模问题的一种行之有效的策略——“分而治之”策略。

模块化程序设计使得一个比较大的问题转化为若干个相对独立的、规模较小的子问题，这给软件开发带来了很多方便和好处。

- 整个系统的开发可以由一个团队合作完成，每个成员只需完成其中的一部分；
- 开发一个模块不必知道其他模块的内部结构和编程细节，只需知道它所需要的那些模块的接口，每个模块可以独立开发；
- 模块之间通过特别的消息传递机制（函数调用，参数传递）有机地结合在一起；
- 模块化系统具有层次结构，降低了复杂性，因此具有易读性，容易阅读和理解；
- 模块化系统具有可修改性，对系统的修改只涉及少数部分模块；
- 模块化系统具有易验证性，每个功能模块可以独立测试验证，而且由于功能单一、规模较小，所以容易验证；
- 模块具有可重用性，每个功能模块可以反复使用，因此也称可复用性。

注意：模块化程序设计要求每个功能模块的规模不要过大，模块的功能应该单一，即遵循模块功能的**高内聚**基本原则。模块的接口(名字和参数)应该尽可能简明，不同模块之间应尽可能少关联，即遵循**低耦合**的基本原则。

5.1.2 函数定义

C/C++结构化程序设计把每个功能模块用**函数**表示，这个函数从功能上来看有点像数学函数，但表现形式和实现方法都与数学截然不同。从第一个Hello程序开始，我们就已经接触到C/C++的函数了，一个是main函数，它是每个软件或程序必须有的；另一个是用来输出信息的printf函数，后来又陆续接触了scanf、sqrt等。不管是哪个函数，它们定义的结构都是一样的，都像我们已经看到的main函数的样子。**函数定义**(**简称函数**)的一般形式如下：

```
/*
*   函数注释
*/
返回值类型 函数名 ( 参数列表 )                    //函数头
{
    声明语句部分;                                 //语句注释
    执行语句部分;                                 //语句注释
    [return(表达式);]
}
```

从整体上来看，**函数定义**是一段**有注释、有名字的程序代码**，它由三部分组成：**函数注释**、**函数头**和**函数体**(大括号括起来的部分，声明语句＋可执行语句＋return)。下面分别详细讨论。

1.函数头

一个函数定义的头也由三部分构成：

(1) **返回值类型**，这个类型用来说明函数使用之后能够返回什么类型的值，因此也有人说它是函数类型。编译器的各种内置数据类型，如整型(int)，浮点型(float、double)，字符型(char)，逻辑型(bool)等均可以作为函数类型，甚至第8章要学习的自定义的各种类型都可以作为函数类型。如果函数使用之后没有要返回的值，就要用void代替。

(2) **函数名**，它是函数的标识符，其命名规则同变量名的命名规则类似，在C语言中两个不同的函数不能同名(C++允许，称为函数重载)。

(3) **参数列表**，它是逗号隔开的一些列表项，每一项说明一个参数。它声明了一组参数，参数之间用逗号隔开。这组参数是将来函数被调用时函数能够接收的参数。声明格式为

```
参数类型 参数名称,参数类型 参数名称,…
```

注意，这个参数列表必须放在一对小括号之内。一对小括号是函数的重要特征，只有看到小括号，才能确定其前面的名字是函数名。例如：

```
int max2(int a, int b);
```

函数的参数列表是“int a,int b”，说明当使用函数max2时可以接受两个整型参数。

每个参数的声明格式“**参数类型 参数名称**”很像一般变量的声明格式。但它只是在形式上给出了参数的类型说明，因此它叫**形参**。在未使用那个函数的时候，形参并不是什么变量，只有这个函数被使用的时候形参才作为那种类型的变量自动产生，才有实际值。既然形参是形式上的参数，因此用什么名称应该无关紧要，但是**参数的顺序**必须明确，如函数 void printRectangle(int h, int w)的功能是打印一个 h 行 w 列的字符图案，第一个参数的意义是行数，而第二个参数的意义是列数，两个参数如果交换一下顺序，意义就不同了。

参数列表也可以为空，但一般要用 void 表示，如果没有用 void，而是空白，C 语言编译器认为任何参数都可以接受。如大家熟悉的 main，如果没有参数，标准的写法都应该写成 int main(void)，尽管写成 int main()也没有出错，但还是存在潜在的危险。7.6.1 节将看到 main 函数也允许有参数。

总之，一个函数定义的参数列表必须明确它**有几个参数**，**每个参数的类型**是什么，各个**参数的顺序**如何。

2. 函数体

一对大括号括起来的所有语句是函数定义的主体，简称**函数体**。函数体是一些语句的集合，就像我们在 main 函数中看到的一样，各种语句都可以出现在函数体中，包括变量声明语句和各种可执行语句，如赋值语句、输入/输出语句、函数调用语句、判断选择语句、循环语句等。要实现一个由函数头确定的函数的功能必须有正确的函数体，也就是必须熟练掌握前面已经学过的变量如何声明，变量的输入输出语句，三种程序控制结构的语句(常常包含复合语句)。

当函数的返回值类型为**非 void** 时，一般至少要包含一个**返回语句**：

```
return(表达式);
```

它把表达式的值返回给将来使用它的语句——函数调用语句。其中，表达式的值的类型与函数的返回类型一致。return 中的表达式可以有各种各样的形式，只要它有适合于返回类型的值即可。例如，“return 1;”或“return a;”或“return (a+b);”均可。return 表达式两边的括号也可以省略。当返回值类型是 void 的时候，可以省略 return 语句，这时右大括号“}”起到返回的作用。

return 语句常常位于函数体的末尾，但有时也在函数体的内部，一般都是当某个条件为真时提前结束函数的执行。

函数头和函数体一起构成了编译器识别的函数定义。从函数的定义可以看出，一个函数实际上就是一组语句被封装在了一起，用一个名字来表示，这组语句在被执行之前会通过参数带进一些信息，在被执行之后将完成一个特定的任务，其结果通过 return 语句或其他形式返回。因此，可以说一个函数就是具有某种特别功能的一种工具。一般来说，使用函数的人比较关心的是要提供给函数什么样的参数它才能执行，函数执行后其结果会是什么，并不太关心函数内部到底是怎么工作的、如何实现的。因此，人们常常把函数看成是一个黑盒子，如图 5.3 所示。

图 5.3 函数是一个黑盒子

3. 函数的注释

大家都知道，注释部分是被编译器忽略的，它仅仅为了方便阅读而存在，但它是比较重

要的一部分。作为一个程序员或者说函数的设计者,不仅要能够设计出符合上述C/C++语言格式的、好用的函数,还要考虑代码的可读性。因此,一方面,必须熟悉前面几章学过的基本程序结构,按照大家公认的程序结构风格写出正确的函数体代码(含各种语句的注释);另一方面,一个好的函数定义,还应该包含一些必要的注释。常用的注释风格如

```
/*
* 函数功能:
* 入口参数:
* 返回值:
*/
```

又如

```
/*
* 函数名称:
* 使用方法:给出具体的调用形式
* ------------------------------
* 函数功能描述(参数说明,返回值说明)
*/
```

下面看几个函数定义的例子。

【例5.1】 定义一个函数,求两个整数的最大值。

定义一个函数要从函数头的三要素出发,即要确定函数名称,函数的参数列表及函数的返回类型。此函数的功能是对任意给定的两个整数,求它的最大值。这里两个整数是变化的,因此必须让函数具有两个整型参数。函数的结果是一个整数,所以返回类型应该是整型。函数的命名应该尽量有意义,应该看到名字就能基本知道函数的功能,因此给这个函数命名为max2int,意思是2个整数的最大值。完整的函数定义如下:

```
#001 /*
#002 * 函数功能:求两个整数的最大值
#003 * 函数参数:两个整数
#004 * 返回值类型:整数
#005 */
#006 int max2int(int a, int b){
#007     return (a>b? a:b);                    //返回a和b中的较大者
#008 }
```

这个函数的函数体非常简洁,只有一个语句,但功能已经实现。它直接返回了条件运算表达式的值,结果是两个参数值的较大者。这个函数的具体使用方法见5.1.3节。

【例5.2】 定义一个函数,打印h行w列的矩形图案。

这个函数要打印一个h行w列的图案,显然函数的参数列表应该是两个整型参数,第一个参数表示行数或者是高度,第二个参数表示列数,也可认为是宽度。它的功能是打印图案,没有计算的结果要返回,因此返回值类型应该是void。函数取名为printRectangle,完整的函数定义如下:

```
#001 /*
```

```
#002 * 函数功能：打印矩形
#003 * 函数参数：两个整数
#004 * 返回值类型：无
#005 */
#006 void printRectangle(int h, int w){
#007    int i,j;
#008    for(i=0;i<h;i++){
#009       for(j=0;j<w;j++)
#010          printf("*");                    //循环打印*号
#011       printf("\n");
#012    }
#013 }
```

【例 5.3】 定义一个函数，显示一个菜单界面。

当一个问题比较复杂时，经过分解会有很多子问题或者具有很多功能模块，这时可以给用户一个菜单界面提示，供用户选择，用户选择不同的功能就去执行不同的功能模块。而且这个界面是始终要显示在用户面前的，需要多次调用才能达到这样的效果。因此，有必要定义一个显示菜单界面的函数，函数命名为 menu，函数定义的完整实现如下：

```
#001  /*
#002   * 函数功能：显示系统主界面
#003   * 入口参数：无
#004   * 返回值：无
#005   */
#006  void menu(void){
#007  printf("         |-----------------------------------------|\n");
#008  printf("         |            请输入选择的数字字符          |\n");
#009  printf("         |-----------------------------------------|\n");
#010  printf("         |       1--添加信息(append record)        |\n");
#011  printf("         |       2--显示信息(list record)          |\n");
#012  printf("         |       3--删除信息(delete record)        |\n");
#013  printf("         |       4--修改信息(modify record)        |\n");
#014  printf("         |       5--查询信息(search record)        |\n");
#015  printf("         |       0--退出 (exit)                    |\n");
#016  printf("         |-----------------------------------------|\n");
#017  printf("                                                    \n");
#018  }
```

它是一个既无入口参数又无返回值的函数，调用它就会在屏幕上显示学生成绩管理系统的主界面菜单。

【例 5.4】 定义猜数游戏模拟的 makeMagic 函数。

它的功能是产生一个随机“想”出来的数，不需要参数，返回类型是一个整型，见本节开始的函数定义。

【例 5.5】 定义猜数游戏模拟的 guessNumber 函数。

它的功能是模拟猜数过程，需要知道计算机想的数是什么，因此要有一个整型参数，没

有无返回值。见本节开始的函数定义。

从前面的讨论可以知道,任何函数都是自己独立的。即函数的定义是不能交叉,也不可以嵌套的。下面两个函数的定义 func1 和 func2 彼此嵌套在一起是不允许的。

```
#001 void func1(void){
#002     …
#003     void func2(void){
#004        …
#005     }
#006     …
#007 }
```

注意:一个函数的规模一般不要过大,一般控制在 50 行以内为宜。一个函数的功能也应尽可能单一,不要让一个函数的负担过重。

5.1.3 函数调用

函数是自顶向下、逐步求精的求解过程中分而治之的结果,每个子问题均可以定义为一个函数,函数模块之间呈现一种层次结构,最顶层的是主函数 main。一般来说,下一层的函数模块是为上一层的函数模块服务的,也就是说,上一层的函数模块要使用下一层的函数,当然如果需要的话,下一层的函数模块也可以使用上一层的函数模块,同层的函数模块也可以互相使用。除了 main 函数具有特殊的地位之外,其他所有函数都是彼此独立的,均可以彼此使用。在一个函数中使用另一个函数称为**函数调用**(**Call**),使用者称为**主调函数**(**或调用函数**),被使用者称为**被调用函数**。函数调用的一般形式是

```
函数名(实参列表)
```

其中,函数名后面的一对小括号是必需的,实参列表是逗号隔开的,它与函数定义中的形参列表一一对应,常量、变量或表达式均可以作为实参,只要类型匹配,可以提供形参需要的"值"即可。函数调用可以独立使用,形成一个函数调用语句,如调用 printRectangle 函数打印一个 5 行 10 列的矩形图案:

```
printRectangle(5,10);
```

也可以作为其他语句或表达式的一部分,如函数 makeMagic 调用的结果赋给变量 magic:

```
magic=makeMagic();
```

注意,当我们要使用某个函数时,必须从以下三个方面着手。

首先要知道那个**函数的名字**是什么,函数的功能是什么。

然后看看它**有无参数**;如果有参数,还要进一步确认:

① 有几个参数;

② 参数都是什么类型的;

③ 参数的先后顺序如何。

最后要知道函数**是否有返回值**,返回值是什么类型,返回值的意义如何。

【例 5.6】 使用函数 max2int 和函数 printRectangle。

以 max2int 函数和 printRectangle 函数为例，看看它们是如何被调用的。这两个函数都是有参数的。函数 max2int 有一个整型返回值，而函数 printRectangle 无返回值。

C/C++ 语言规定，一个函数在被调用之前**必须定义或声明**，否则编译的时候就会出现警告和错误，例如程序中使用函数 func：

```
warning: implicit declaration of function 'func'
undefined reference to 'func'
```

这与一个变量在使用它之前必须先用声明定义一样。编译器要知道被调用的函数是什么样的才允许使用。怎么让编译器知道呢？**方法之一就是把函数定义的代码放在函数调用语句代码之前**，这个“之前”未必是相邻，只要在前面即可。如果要在 main 函数中使用 max2int 和 printRectangle 函数，把 max2int 和 printRectangle 两个函数的定义放在 main 函数之前即可，见程序清单 5.2。

程序清单 5.2


源码 5.2

```
#001 /*
#002 *   useFuncs.c: 函数使用举例,函数定义在函数调用之前
#003 */
#004 #include<stdio.h>
#005 #include<stdlib.h>
#006 /*
#007 * 函数功能: 求两个整数的最大值
#008 * 函数参数: 两个整数
#009 * 返回值类型: 整数
#010 */
#011 int max2int(int a, int b){
#012     return (a>b? a:b);     //返回 a 和 b 中的较大者
#013 }
#014 /*
#015 * 函数功能: 打印矩形
#016 * 函数参数: 两个整数
#017 * 返回值类型: 无
#018 */
#019 void printRectangle(int h, int w){
#020    int i,j;
#021    for(i=0;i<h;i++){
#022       for(j=0;j<w;j++)
#023          printf("*");     //循环打印*号
#024       printf("\n");
#025    }
#026 }
#027 /*
#028 * 主函数: 使用 max2 和 printRectangle 函数
#029 */
#030 int main(void){
```