第5章

模块化程序设计

【内容提要】 模块化程序设计是以“模块”为中心的程序设计方法。它将系统划分成若干功能模块，每个模块独立实现某个(些)特定的功能。然后，将所有的模块按照某种方法组装起来，成为一个整体，实现整个系统的功能。模块化程序设计通过功能分解方式控制程序的复杂度，分解出来的模块可用程序设计语言中的子程序实现。本章首先阐述模块化程序设计的概念，分析利用模块化程序设计的主要步骤、主要内容及过程。在此基础上，介绍模块化程序设计方法在C语言中的实现——函数程序的设计，包括函数的定义、信息传递方式和返回值，以及函数设计中所涉及的变量的作用域和生存期。

【学习目的和要求】 学习本章的主要目的是使学生熟悉模块化程序设计的概念、过程和主要任务，理解变量的作用域与生存期。要求掌握函数的定义方法，函数间信息传递的方法，包括虚实参数结合的方式、函数返回值，以及如何利用模块化程序设计方法进行复杂问题的求解。

【重要知识点】 功能分解；模块化设计；模块化程序设计；函数；存储类别。

5.1 模块化程序设计的概念

5.1 模块化程序设计的概念

对于规模较大的程序，结构化程序设计采用了“分而治之”的策略，即将任务分解成几个子任务分别进行处理。出于该原因，编程者往往将程序分解成一个个小模块。这些小模块可以按照一定的顺序来执行，最终完成总体任务。由于模块相互独立，在设计其中一个模块时，不会受到其他模块的牵连，可以将原来较为复杂的问题简化为一系列简单模块的设计，这就是模块化设计。**模块化设计**(Modular Design)是来自于设计学领域的设计理论和设计方法，指在进行产品设计时，不是自己生产每个部件，而是先将某些要素进行组合，构造出具有特定功能的通用子系统(模块)，然后将这些通用模块与其他要素进行组合，设计具有不同功能或相同功能但具有不同性能的系列产品。如果将模块化设计的思想应用到程序设计领域，就是模块化程序设计。

一般的高级程序设计语言都提供了一种抽象机制——**子程序**(Subprogram)技术。程序设计中，如果其中有些操作内容完全相同或相似，为了简化程序，可以把这些重复的程序段单独罗列出来，并按一定的格式编写成子程序。即编程者可以针对某些基本任务，基于子程序机制编写一个程序块，将程序块的实现细节封

装在该程序块内部，以子程序名以及入口参数和返回值的形式提供调用接口。相较于其他代码，子程序具有相对独立性。主程序在执行过程中如果需要使用某一子程序的功能，可通过调用指令来调用该子程序。主调程序在被调用子程序执行期间被挂起，即在任一给定时刻仅有一个子程序在执行。子程序执行结束后将控制权交还给主调程序，继续执行主调程序中的其余部分。子程序技术是“自顶向下、逐步求精”设计策略的基础，它将 what(做什么)和 how(怎么做)相分离。具体设计时，先按照自顶向下策略，为了完成某个操作，编程者只需关心要“做什么”(调用什么功能)即可，而不必关心“怎么做”，从而可以集中精力处理好高层架构。然后在逐步求精的过程中，再去处理“怎么做”(设计子程序本身)，设计具体的算法细节。

子程序机制下，一个子程序就是一个程序**模块**(Module)，编程者以模块化方式设计程序。**模块化程序设计**(Modular Programming)采用“组装”的办法简化程序的设计过程，是一种程序设计技术，它将一个系统分解成多个模块，然后对各个模块分别进行编程，最终实现整个系统的功能。

基于以上理由，模块化程序设计被认为是结构化设计的重要基础和内容之一。即自顶向下、逐步求精的结构化程序设计方法从问题本身开始，经过逐步细化，将解题步骤分解为由基本程序结构模块构成的结构化框图。

综上所述，模块化程序设计是指在设计程序时，将由复杂功能构成的系统按照功能划分为若干小部分(程序模块)，每个模块实现相对独立的功能，并在这些模块之间建立必要的联系，通过模块的互相协作实现整体功能。其设计机理可概括为“**功能独立、易于集成**”，它采用了分而治之、自顶向下、逐步分解的策略，将一个较大的程序按照功能分割成一些小模块，各模块相对独立、结构清晰、接口简单。

模块化程序设计的主要特点：

(1) 模块化设计使得模块间保持相互独立性，程序逻辑结构清晰，每个模块可以独立地被编写、被测试和被修改，使得程序易写易读易懂。

(2) 允许编程者有效地对过程进行抽象。即将模块的具体实现细节隐藏在模块内部，这样在程序中调用时就不再需要描述详细的步骤。这给模块的使用带来了很大的便利，也使程序内部细节不被知晓和破坏，可隔离错误，有效防止错误在模块之间扩散蔓延，有利于信息隐藏，提高了程序的可靠性。

(3) 模块作为整体方案的一小部分，每个模块独立开发和测试，测试更方便，编程者可以单独测试程序的每个模块，以确定它是否正确实现了预定的功能。

(4) 由于能够进行功能分解，使得程序易于团队合作并行开发，也有利于后期维护、升级和管理。另外，由于将重复的处理放在一个函数中，只需编写一次，从而可以实现代码的复用，既可以节省内存空间也节省了编码时间。

(5) 模块的独立性为扩充已有的系统、建立新系统带来很大的方便，编程者可以充分利用现有的模块做“积木式”扩展。

5.2 模块化程序设计过程

5.2 模块化程序设计过程

在模块化程序设计中，每个子功能被设计成独立的程序模块。如第 2 章所述，模块是源程序及计算机系统的基本构成单元，即整个系统由若干源程序组成，每个源程序又由一些小

的功能模块组合而成，每个模块由更小的子模块构成。这样一来，一个计算机程序就可以分成多个程序模块，编程人员可采用面向过程/对象方法构建程序模块。

模块化程序设计的主要步骤如下。

（1）分析并明确拟解决的问题；

（2）逐步进行任务分解与细化；

（3）确定模块之间的关联关系；

（4）实现程序模块之间的调用。

上述过程中，程序设计人员需要完成的工作内容如下。

（1）程序模块的功能分解；

（2）程序模块间语句的划分；

（3）程序模块间的通信方式。

5.2.1　程序模块的功能分解

模块是为了执行特定任务而存在于程序中的一组语句，分解是指在解决复杂问题时，按照欲实现的功能将一个系统划分为多个功能模块，然后对各个功能模块分别进行编程，采用“组装”模块的办法来简化程序设计过程。通过模块分解，各模块之间形成了上下层的关系，上层模块的功能需要通过调用下层模块来实现。因此，模块之间的关系可以用层次图或功能结构图来说明，这时无须考虑模块内部操作的详细信息。

合理的模块分解，应遵循“**高内聚、低耦合、数适中**”的原则：

（1）一般按功能进行分解，分解时注意保持模块的内聚性（独立性），使一个模块尽可能只完成一项或者少数几项功能。

（2）减少模块间的耦合性（关联性），保证模块内部的信息处于相对封闭的状态，尽量不和其他模块发生或少发生联系。

（3）模块数目要适中。模块数不宜过少或过多，过少意味着每个模块的功能将比较复杂，模块规模会很庞大。但随着模块数的增加，模块间的接口设计的复杂性和工作量将随之增加。统计表明，模块的代码一般不要超过 50 行。

1. 内聚性

高内聚性是模块化设计的一个重要要求。内聚性（Cohesion）又称**块内联系**，是指程序模块内各个元素之间相互关联的紧密程度，它也是评判程序设计质量的指标。内聚性越高，模块内部各元素之间关联得越紧密或越集中，模块的独立性越好。通常情况下，多采用功能内聚或过程内聚的方式，即为了完成一项具体的功能而将相关的元素组合在一起，或将相关元素按照特定的执行次序而予以组合。

内聚性的道理比较简单，就是“志同道合”“道同相谋”。同一组织内部，大家应该心往一块想、智往一起谋、力往一处使，互相紧密合作，齐心协力。如果某人离心离德，就要想办法开除他。

2. 耦合性

一般情况下，一个程序中的各模块之间存在着或多或少的关联关系，包括控制关系、调用关系、数据传递关系，所谓“你连着我、我连着你”。模块的耦合性（Coupling），又称**块间联系**，用来衡量模块之间的依赖关系，其强弱取决于模块间接口的复杂性、调用模块的方式以

及通过接口传递数据量的大小。耦合性愈小，表明模块间联系越少，模块的独立性就愈好。

程序模块的耦合性取决于以下因素：

(1) 各个模块之间接口的复杂程度；

(2) 模块调用另外一个模块的方式；

(3) 向另一个模块传递的数据状况；

(4) 一个模块对另一个模块的控制。

通过降低模块间的耦合性可以减少模块间的影响，防止出现对某一模块修改所引起的“牵一发动全身”的水波效应和“副作用”。

关于这种情况，有人拿人力资源部部长与项目经理之间的谈话打比方。

部长：如果再给你增加一个人，项目什么时候可以完工？

项目经理：一年吧！

部长：那增加两个人呢？

项目经理：8 个月。

部长：三个人呢？

项目经理：6 个月。

部长：10 个人呢？

项目经理：两年。

部长：100 个人呢？

项目经理：那我将永远无法完成任务！

这段玩笑说明，程序并不因为模块数越多质量就越好，因为更多的程序模块，意味着更多的交互与通信开销，以及更大的“副作用”。特别是对于功能比较复杂以及多人协作开发维护的程序，修改一个小地方会引起本来已经运行稳定的模块发生错误，严重时会导致恶性循环，问题永远改不完。

在 C 语言中，当组成程序的函数之间的调用关系太多时，各个函数就组成了一个复杂的调用网络，将很难厘清它们之间的关系。例如，一个程序由 50 个函数构成，这个程序编译、链接后执行得非常好。然而，一旦修改其中一个函数，可能其他 49 个函数都需要做相应修改，这就是高耦合的后果。

因此，合理的模块划分，应该是模块内部凝聚力强，功能单一，模块之间的联系越少越好。

5.2.2 程序模块间语句的划分

这是模块化程序设计应该解决的第二个问题。

为了更好地理解该问题，首先看一个例子。这个例子要求设计一个程序，实现从键盘上输入两个整数、输出其中较大值的功能，并且要求程序包括一个主函数、一个子函数，子函数的主要功能是比较两个数，然后由主函数调用该子函数。

为了实现上述功能，下面给出了几种设计好的程序代码，读者可自行比较一下，看看哪种方法更为合理？

方法一：

```
#include<stdio.h>
void f1()                                    //将 I、P、O 操作全部放到 f1()函数中
```

```
{
  int x, y, z;
  scanf("%d%d", &x, &y);
  if(x>y)   z=x;
  else   z=y;
  printf("The big is %d!\n", z);
}
void main()
{
  f1();
}
```

方法二：

```
#include<stdio.h>
void f2(int a, int b)                          //将 P、O 操作放到 f2()函数中
{
  int c;
  if(a>b)   c=a;
  else   c=b;
  printf("The big is %d!\n", c);
}
void main()                                    //将 I 操作放到 main()函数中
{
  int x, y;
  scanf("%d%d", &x, &y);
  f2(x, y);
}
```

方法三：

```
#include<stdio.h>
int f3(int a, int b)                           //将 P 操作放到 f3()函数中
{
  int c;
  if(a>b)   c=a;
  else   c=b;
  return c;
}
void main()                                    //将 I、O 操作放到 main()函数中
{
  int x, y, z;
  scanf("%d%d", &x, &y);
  z=f3(x, y);
  printf("The big is %d!\n", z);
}
```

依据第 2 章中介绍的程序的 IPO 结构，可总结出模块间语句划分的一般原则如下：

(1) 方法一中，f1()函数承担了全部的 I、P、O 操作，大包大揽，责任重大。该方法主要

适用于要求所有 I、P 和 O 均在模块内完成的情形。

(2) 方法二将 P、O 操作放到 f2()函数中，而将 I 操作放入 main()函数，对于有些要求在模块内进行处理的同时实现结果输出的情形，可采用该种方法。

(3) 对于功能比较复杂且需要多次调用的情形，可采用方法三，即将 I、O 操作放到主调函数中，数据 P 操作放到子函数中。此时，由于 I 和 O 操作相对简单，P 操作逻辑复杂一些，各模块负载总体上是平衡的。如果 I、O 操作比较复杂，也可以考虑将 I、O 和 P 分别定义为独立的子函数。

5.2.3 程序模块间的通信方式

程序模块的通信，又称模块间信息的传递，它是指主调模块将信息发送给被调用模块，同时接收来自被调用模块的处理结果。被调用模块接收来自主调模块的已知信息，同时在处理完成后将结果返回给主调模块。因此，模块间的这种通信是双向的。

对于模块化程序设计，尽管推崇高内聚、低耦合，强调各程序模块功能尽量单一和保持独立性，但这并不意味着各个模块完全孤立，与其他程序模块老死不相往来。实际上，同一个程序的各个程序模块之间往往存在着数据传递关系。

程序模块间信息传递的方式分为值传递、地址传递和引用传递。

1. 值传递

值传递(Pass-by-Value)指一个程序模块向另一个程序模块传送了一个数值，其实现方式可以是通过参数的对应关系传递，也可以是凭借全局变量传递，数值的类型可以是整型、实型或字符型等。

2. 地址传递

地址传递(Pass-by-Address)指主调模块向被调用模块传送变量的地址、数组的首地址或子程序的入口地址。地址传递其实就是多开辟了一个存储空间，并使其指向拟处理的变量、数组或子程序的地址，进而在被调用模块中改变该地址对应空间中的内容。

3. 引用传递

引用传递(Pass-by-Reference)与地址传递类似，但又有所不同，它为主调模块中的变量另起一个名字(别名)，在被调模块中可以直接按照值传递的方法使用该名字。引用传递其实就是相当于直接把那个实参本身传过去，修改的就是那个实参的值，节省了开辟形参的操作，同时能提高效率，而且可以避免涉及指针的操作。

总之，模块化程序设计一般采用自顶向下方法，将 I、P 和 O 设计成独立模块，通过模块间相互交换信息达到协调彼此间的同步及协作关系。

模块化程序设计也给了编程者以启示：首先，模块化程序设计讲究“分工合作”，因此进行程序设计必须具备团队精神和合作意识，俗话说得好，“人多力量大、众人拾柴火焰高”。在实现中国梦的伟大征程中，作为新时代的大学生，必须时刻牢记团结就是力量，合作才能共赢。其次，在共享经济时代，软件行业从业者必须具备一定的工程经济意识，遵守软件行业标准，积极通过开源软件、共享软件、代码复用等达到节约人力物力财力的目的。

【扩展阅读】 Pascal 之父：尼古拉斯·沃斯。

尼古拉斯·沃斯，1934 年生于瑞士温特图尔，世界知名计算机科学家。1958 年，沃斯从苏黎世工学院取得学士学位后到加拿大拉瓦尔大学深造，于 1960 年获得硕士学位，之后进

入美国加州大学伯克利分校学习，并于 1963 年获得博士学位。1963—1967 年，他担任了斯坦福大学刚刚成立的计算机科学部助理教授。他于 1967 年回到祖国瑞士，先后在苏黎世大学、苏黎世工学院担任信息学教授。1968 年，他创建了当时世界上最受欢迎的语言之一——Pascal 语言。

有趣的是，沃斯开发 Pascal 语言的初衷是为了创建一个适用于教学用的计算机语言。但该语言一经推出，由于其语法简洁明了，具有丰富的数据结构和控制结构，为程序员提供了极大的便利与灵活性，使得其在市场上大受推崇。后来，沃斯的学生 Philipe Kahn 和 Anders Hejlsberg(Delphi 之父)创办了 Borland 公司，该公司靠 Turbo Pascal 起家，成为当时全球最大的开发厂商。Pascal 语言整整影响了几代程序员，在 C 语言问世以前，它是风靡全球、最受欢迎的编程语言之一，创下了发行拷贝数最多的世界纪录。20 世纪 70 年代中期，为适应并发程序设计的需要，沃斯又成功开发了一个获得广泛应用的语言 Modula。Modula 除了提供并发程序设计功能之外，还引进了模块、进程(Process)等与并发程序相联系的重要概念。他还是 Euler 语言的发明者之一。1976 年，沃斯再次远赴美国，在施乐(Xerox)公司的 Palo Alto 研究中心参与 Alto 计算机的设计与开发工作。1984 年，沃斯获得了图灵奖，他是瑞士学者中唯一获此殊荣的人。他于 1987 年获得计算机科学教育杰出贡献奖，1988 年获得计算机先驱奖。

沃斯因其所提出的名言“**算法＋数据结构＝程序**”而闻名于世，在结构化程序设计方面做出了重要的贡献。1971 年，沃斯基于其开发程序设计语言和编程的实践经验，在 *Communications of ACM* 上发表了论文《通过逐步求精方式开发程序》(*Program Development by Stepwise Refinement*)，进一步完善了“结构化程序设计”的概念，故结构化程序设计方法又称“自顶向下”或“逐步求精”法。结构化程序设计方法在计算机领域引发了一场革命，成为程序开发的一个标准方法。1983 年 1 月，ACM 在纪念 *Communications of ACM* 创刊 25 周年时，从其 25 年来所发表的论文中评选出 25 篇具有“里程碑意义的研究论文”，每年一篇，沃斯的这篇论文就名列其中。

沃斯的学术著作很多，包括《系统程序设计导论》《算法＋数据结构＝程序》《算法和数据结构》《Modula-2 程序设计》《PASCAL 用户手册和报告》《Oberon 计划：操作系统和编译器的设计》《Oberon 程序设计：超越 Pascal 和 Modula》等。

5.3　基于 C 语言的模块化程序设计

不同的编程语言可以通过差异化的方式实现子程序。C 语言也不例外，它提供了对模块化程序设计的较好支持。C 语言中的模块化功能通过函数实现，即事先编好一批能实现各种不同功能的函数，每个函数用来实现一个(组)特定的功能，并将这些函数在源程序中实现或保存在函数库中，需要的时候直接调用，从而减少重复编写代码的工作量。

对于像 C 语言这种函数式的语言，其本质上是由若干函数组成。因此，程序设计是围绕函数来进行的，功能的实现过程表现为一系列的函数应用，如图 5-1 所示。

函数(Function)是对数学函数的模型化，用来定义新的操作，具有较强的组织数据结构的能力。利用函数可以减少重复编写程序段的工作量，非常方便地实现模块化设计。实际应用中，一般做法是把规模较大的程序分解为若干程序模块，每个模块包含一个或多个函

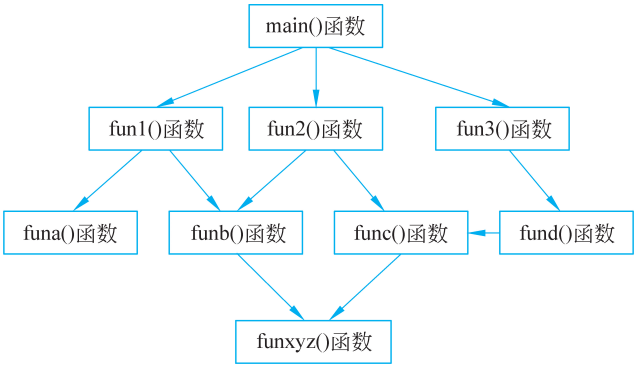

图 5-1　C 程序中的函数调用关系

数，每个函数实现一个(组)特定的功能。可以调用库函数，也可自己新建函数。对于每个程序，其执行的入口点是主函数 main()，通过 main()函数调用其他函数，这些被 main()函数调用的函数又可以调用另外的函数，或者互相之间进行调用，一个函数可被一个或多个函数调用任意多次。

5.3.1 函数的定义及声明

5.3.1　函数的定义与声明

1. 函数定义

模块化程序设计的第一步就是要确立程序模块——函数的功能，即进行函数定义。**函数定义**(Function Definition)用于描述函数抽象的接口和行为，即从无到有确立函数的功能，它主要解决如下问题：

(1) 函数的名称是什么？

(2) 函数参数名及类型是什么？

(3) 函数返回值的类型是什么？

(4) 函数要实现的功能是什么？

函数定义的语法格式：

```
类型名　函数名(形参列表)
{
    函数体
}
```

从上面语法形式可以看出，每个函数定义由函数首部和函数体两部分组成。其中，函数首部包括函数类型、函数名，以及函数形参列表，一般将函数的参数配置与函数返回值类型合起来称为**函数协议**(Function Protocol)。这里，“**函数名**”符合标识符的命名规则，即由字母、数字和下画线组成，且必须以字母或下画线开头；“**类型名**”指定了函数值(一般通过被调用函数中的 return 语句带回)的类型，简称**函数类型**，可以是 int、float、double、char、指针或结构体类型等，缺省时为 int 类型，但不允许是数组或函数类型，如果明确将来通过函数调用不需要带回数值或地址值等，可直接指定函数类型为 void 类型；“**形参列表**”用于指定本函数和主调函数之间交换信息的形式(函数配置)，即传递的信息数量是多少，传递什么类型的信息等，这些参数应一一列出，每个参数均应指明其类型和参数名，各个形参之间用逗号

分隔；“函数体”是由“{”和“}”括起来的复合语句，用于定义函数动作，由局部变量定义、全局变量的声明和语句组成，在语句中指出了拟在本函数中实施的操作（功能）。

C 语言中的函数分为有参函数和无参函数，它们的定义形式如下。

1）无参函数

无参函数即主调函数和被调函数之间没有参数传递。无参函数定义的语法格式：

```
类型名　函数名()
{
    函数体
}
```

或

```
类型名　函数名(void)
{
    函数体
}
```

无参函数一般用于输出文本信息、绘制图形，或者直接在函数中输出结果等情形，函数调用时不用利用参数带入数值信息，执行后可以带回也可以不带回返回值给主调函数。如果函数没有返回值，则返回值的类型可以指定为 void。

【例 5-1】　定义函数 Add，在屏幕上输出两个整数的和。

解：

本题比较简单，虽然有数据传入，但由于不需要带回数据，因此可以直接定义一个 void 类型的函数，在该函数中直接输出两个整数的和。程序代码如下：

```
#include<stdio.h>
void Add(int x, int y)
{
  printf("%d+%d=%d\n", x, y, x+y);
}

int main()
{
  Add(3, 5);
  return 0;
}
```

2）有参函数

有参函数定义的语法格式为

```
类型名　函数名(数据类型 参数 1, 数据类型 参数 2, …, 数据类型 参数 n)
{
    函数体
}
```

此时，主调函数与被调函数之间具有数据传递关系，可以传递一至多个数据给被调函数。定义函数时参数表列中的参数称为“形式参数”，简称形参，它是主调函数与被调函数间信息传递的接口。与上面一样，应分别标出每个参数的数据类型，各参数之间以“，”进行分隔。函数执行以后，可以带回值也可以不用带回值。

【例 5-2】 定义函数 Sum()，输出从整数 a 加到 b 的最终结果。要求在主函数 main 中从键盘上输入 a 和 b 的值，并在 main()函数中显示运算的结果。

解：

本题中，将函数 Sum()的定义放到主函数 main()之前。这里，涉及了程序模块间的数据传递关系，需要将 a、b 的值传送到 Sum()函数中。因此，Sum()函数应定义两个整型形参变量，用于接收来自 a 和 b 的值。

相应的程序代码如下：

```
#include<stdio.h>
int Sum(int x, int y)
{
  int i, s=0;
  for(i=x; i<=y; i++)   s+=i;
  return s;
}

int main()
{
  int begin, end, result;
  printf("请输入开始值 a 和结束值 b:");
  scanf("%d%d", &begin, &end);
  result=Sum(begin, end);
  printf("The sum from %d to %d is %d\n", begin, end, result);
  return 0;
}
```

程序运行结果：

```
请输入开始值a和结束值b: 1 100
The sum from 1 to 100 is 5050
```

函数一经定义，将在编译阶段由系统给该函数在内存中分配一段连续的空间，在该空间存储了该程序的相关指令等。这段空间的首地址由函数名标识，即函数名代表了函数在内存中的首地址。

2. 函数声明

函数调用需要满足下面两个条件之一：①函数定义位于主调函数之前；②当函数返回值的类型为 int 时，不是所有系统都要求函数定义一定位于调用语句之前。除了这两个条件，一般情况下需对被调函数进行声明。

以例 5-2 为例，当函数定义位于 main()函数之后时，程序在 Code::Blocks 环境下可以顺利进行编译、链接和运行。但在 VC++ 6.0 环境下，编译时将会显示如图 5-2 所示的信息。