普通高等教育“十一五”国家级规划教材

21世纪高等学校计算机规划教材

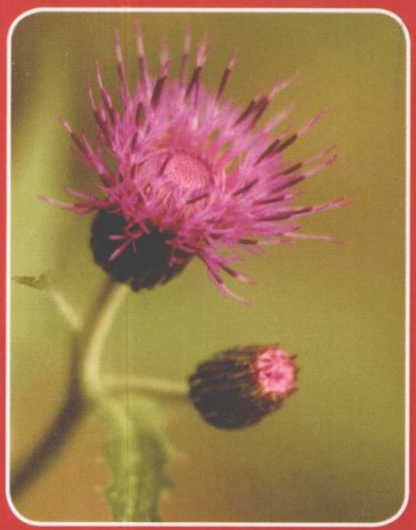
21st Century University Planned Textbooks of Computer Science

C/C++语言程序设计教程——从模块化到面向对象（第3版）

The C and C++ Programming Language——from Modular to Object Oriented (3rd Edition)

李丽娟 编著

- 掌握模块化程序设计的思想和方法
- 了解C和C++的关系与区别
- 掌握面向对象程序设计的基本方法



精品系列

人民邮电出版社
POSTS & TELECOM PRESS

图书在版编目（CIP）数据

C/C++语言程序设计教程 ：从模块化到面向对象 /
李丽娟编著. -- 3版. -- 北京 ：人民邮电出版社,
2012.2
普通高等教育“十一五”国家级规划教材 21世纪高
等学校计算机规划教材
ISBN 978-7-115-27317-8

Ⅰ. ①C… Ⅱ. ①李… Ⅲ. ①C语言－程序设计－高等
学校－教材 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2011)第273877号

内 容 提 要

本书要求读者具有了C语言的基本知识，已经掌握了C语言的基本表达式语句、分支结构语句及循环结构语句，能够用这些基本知识解决一些简单的问题。本书从C语言模块化的程序设计方法入手，过渡到C++程序设计基础，完成从面向过程的程序设计到面向对象的程序设计的学习。

全书内容分为三部分，共9章。第一部分为第1章，是C语言模块化程序设计基础，主要介绍如何通过自定义函数进行模块功能设计的基本方法，这部分内容是模块化程序设计的基础。第二部分为第2章～第6章，是应用程序设计基础，主要介绍数组、指针、结构、文件和位运算等基础知识，通过学习这部分的知识，使读者更加熟练地掌握模块的功能设计，采用更多更丰富的方法处理程序的复杂数据，学会使用不同的数据存储方式和数据提取方式，逐步认识模块化程序设计的思想，掌握模块化程序设计的方法。第三部分为第7章～第9章，是C++程序设计的基础，主要介绍从C语言过渡到C++的新增语法功能和面向对象程序设计的基本方法。通过学习，使读者了解到C语言和C++语言的关系，了解面向对象程序设计的基本方法，进一步提高分析问题和解决问题的能力，为后续的深入学习奠定基础。语言简洁，通俗易懂，内容叙述由浅入深。

本书适合作为大学本科和专科院校的教材，也可供一般工程技术人员参考。

普通高等教育“十一五”国家级规划教材

21世纪高等学校计算机规划教材

C/C++语言程序设计教程——从模块化到面向对象（第3版）

◆ 编　　著　李丽娟
责任编辑　易东山
执行编辑　代晓丽

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街14号
邮编　100061　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京隆昌伟业印刷有限公司印刷

◆ 开本：787×1092　1/16
印张：18　　2012年2月第3版
字数：443千字　　2012年2月北京第1次印刷

ISBN 978-7-115-27317-8

定价：35.00元

读者服务热线：(010)67119329　印装质量热线：(010)67129223
反盗版热线：(010)67171154
广告经营许可证：京崇工商广字第0021号

前言

程序设计是理工类各专业重要的基础课程之一，该课程在培养学生的思维能力和动手能力方面起到了重要的作用，程序设计的理念和方法有助于培养学生分析问题和解决问题的能力。为适应社会发展对大学生的素质要求，我们对程序设计课程的教学内容进行了适当的调整，其目的是给学生提供更多更普遍的知识信息，有利于学生后续课程的学习，同时为学生的自学提高奠定一定的基础知识。

2002 年，作者曾编写出版了《C 程序设计基础教程》。随着教学要求的变化，2005 年对该书进行了修改，于 2006 年入选国家“十一五”规划教材，并出版了《C 语言程序设计教程》，2009 年对该教材进行了修订，出版了《C 语言程序设计教程》（第 2 版），该教材在程序设计语言的教学中起到了积极的作用。

本书在继承前两种教材特色的基础上，结合作者多年的教学经验，特别根据近几年教学改革的实践以及对人才培养的高标准要求，对其内容做了进一步的优化、补充和完善。将教学内容分成两个不同的阶段，同时将教材也分成了两大部分：第一部分为初级阶段，第二部分为中级阶段。第一部分的教学内容为 C 语言的基本变量和基本表达式、基本程序语句、分支结构语句、循环结构语句。这部分内容作为 C 语言的基础，使学习者对 C 语言有一个初步的了解，并对简单的程序设计有较好的了解和掌握。第二部分的教学内容除了 C 语言的函数、数组、指针、结构、文件、位运算等知识外，还加入了 C++的基础知识，这样可使程序设计的知识有较好的延续，为其后续的深入学习打下一定的基础。

本教材的起点是程序设计的中级阶段，学习者应该已经具有了程序设计初级阶段的基础知识，本书将 C 语言中级阶段的知识分成以下 3 个循序渐进的部分。

第一部分是模块化程序设计基础，由第 1 章组成。主要介绍 C 语言程序基本单元的设计方法，这也是模块化程序设计的基本方法。这部分的内容奠定了 C 语言模块化程序设计的基础，通过学习，读者可以设计具有独立功能的函数，有利于培养解决问题的能力。

第二部分是应用程序设计基础，由第 2 章～第 6 章组成。主要介绍数组、指针、结构、文件和位运算等基础知识，为程序中数据的存储和提取提供更多更方便的元素和方法。通过学习这部分的知识，读者可以进一步掌握自定义函数的设计，采用更多更丰富的方法处理程序的复杂数据，学会使用不同的数据存储方式和数据提取方式，掌握基本的算法设计，并能将算法通过程序来实现，培养分析问题的能力，为应用软件的程序设计奠定基础。

第三部分是C++程序设计基础，由第7章～第9章组成。主要介绍从C语言过渡到C++语言的新增语法功能和面向对象程序设计的基本方法。通过学习，使读者了解到C语言和C++语言的关系，了解面向对象程序设计的基本方法，进一步提高分析问题和解决问题的能力，为后续的深入学习奠定基础。

本书具有以下特色：

1．层次清晰，设计方法由浅入深。

如在第1章中，通过对C语言程序的基本单元的了解，掌握模块化程序的基本方法，充分认识C语言程序的优越性，深入了解C语言程序从编辑到程序调试、运行的基本过程，强化C语言程序模块化设计的概念。

2．案例丰富，启发性强。

本书精选了丰富的程序案例，所有程序都在Visual C++ 6.0环境下通过验证，并且对程序的结构、函数的设计、变量的设置进行了恰当的注释和说明。其中大量的程序案例留有可进一步探讨的余地，给教师的教学和读者的自学留下了广阔的空间，可以启发读者思考，从中发现问题，寻找解决问题的方法。从而不断激发读者的学习兴趣，激发想象力和创新思维能力。

3．C语言与C++融合，顺利地从C语言过渡到C++。

由于C++语言是在C语言的基础上发展起来的，因此，在有了C语言的基础之后，可以很顺利地过渡到C++，从面向过程的程序设计到面向对象的程序设计，通过不同的程序设计理念，掌握多途径分析问题和解决问题的能力，培养读者独立思考和创新思维的能力。

为了巩固所学的理论知识，本书每章都附有习题，以帮助读者理解基本概念，通过理论联系实际进行书面练习和上机编写程序，熟练掌握C语言模块化程序设计方法和C++面向对象程序设计方法，学会简单算法设计并实现，提高程序设计能力。

与本书配套的习题解答与实验指导给出了本书中习题的全部参考答案和学生上机实验的内容。在实验中，读者可以通过编写程序，然后编译、运行，查看程序的运行结果，根据程序的运行结果验证程序的正确与否，逐步掌握程序设计的基本方法和基本技能。

本教材的课程教学建议学时为80，其中课堂教学学时为48，上机实验学时为32，各章的学时数安排可大致如下表所示。实际教学中可以根据具体情况予以调整，适当减少或增加学时数。

章	内　　容	课堂教学学时	上机实验学时
1	函数	8	6
2	数组	8	6
3	指针	8	4
4	结构	4	2
5	文件	4	2
6	位运算	4	2

（续表）

章	内　　容	课堂教学学时	上机实验学时
7	C++新增语法功能	4	2
8	类与数据抽象（一）	4	4
9	类与数据抽象（二）	4	4
合计		48	32

本书可以作为普通高等院校计算机专业及理工类各专业本科、专升本的教材，也可作为研究生入学考试和各类认证证书考试的复习参考书，还可供计算机应用工作者和工程技术人员参考阅读。

本书由李丽娟任主编。感谢吴蓉晖、杨小林、洪跃山、李根强、银红霞、谷长龙、李小英等为本书提出的宝贵意见。

由于作者水平有限，书中难免存在不妥与疏漏之处，敬请广大读者批评指正。

李丽娟

2011 年 12 月于岳麓山

目 录

第一部分 模块化程序设计基础 …… 1

第 1 章 函数与宏定义 …… 3

1.1 函数的概念 …… 3

1.1.1 函数的定义 …… 3

1.1.2 函数的声明和调用 …… 5

1.1.3 函数的传值方式 …… 5

1.2 变量的作用域和存储类型 …… 8

1.3 内部函数与外部函数 …… 11

1.4 递归函数的设计和调用 …… 13

1.5 预处理 …… 17

1.5.1 宏定义 …… 17

1.5.2 文件包含 …… 20

1.5.3 条件编译及其他 …… 21

1.6 综合范例 …… 24

1.7 本章小结 …… 32

习题 …… 33

第二部分 应用程序设计基础 …… 39

第 2 章 数组 …… 41

2.1 一维数组的定义和初始化 …… 41

2.1.1 一维数组的定义 …… 41

2.1.2 一维数组的初始化 …… 43

2.2 一维数组的使用 …… 44

2.3 多维数组 …… 46

2.3.1 二维数组的概念 …… 46

2.3.2 二维数组的定义 …… 47

2.3.3 多维数组的定义 …… 48

2.3.4 二维数组及多维数组的初始化 …… 49

2.4 字符数组 …… 52

2.4.1 字符数组的初始化 …… 53

2.4.2 字符串的输入 …… 54

2.4.3 字符串的输出 …… 55

2.4.4 二维字符数组 …… 56

2.5 数组作为函数的参数 …… 61

2.5.1 数组元素作为函数的参数 …… 61

2.5.2 数组名作为函数的参数 …… 62

2.6 程序范例 …… 66

2.7 本章小结 …… 73

习题 …… 73

第 3 章 指针 …… 78

3.1 指针的概念 …… 78

3.1.1 指针变量的定义 …… 79

3.1.2 指针变量的使用 …… 79

3.1.3 指针变量与简单变量的关系 …… 80

3.2 指针的运算 …… 81

3.2.1 指针的算术运算 …… 81

3.2.2 指针的关系运算 …… 83

3.3 指针与数组的关系 …… 83

3.3.1 指向一维数组的指针 …… 83

3.3.2 指向多维数组的指针 …… 86

3.3.3 字符指针 …… 92

3.3.4 指针数组 …… 93

3.4 指针作为函数的参数 …… 95

3.5 函数的返回值为指针 …… 98

3.6 指向函数的指针 …… 99

3.7 main()函数的参数 …… 101

3.8 指向指针的指针 …… 103

3.9 图形处理模式 …… 104

3.10 程序范例 …… 108

3.11 本章小结 …… 116

习题 …… 117

第 4 章 构造数据类型 …… 122

4.1 结构体数据类型 …… 122

4.1.1 结构体的定义 …… 122

4.1.2 结构体变量的定义 …… 123

4.1.3 结构体变量的初始化 …… 124

4.1.4 结构体变量成员的引用 …… 125

4.1.5 结构体变量成员的输入/输出 …… 127

4.2 结构体数组 ……128
4.2.1 结构体数组的定义 ……128
4.2.2 结构体数组成员的初始化和引用 ……128
4.3 结构体变量与函数 ……129
4.3.1 函数的形参与实参为结构体 ……129
4.3.2 函数的返回值类型为结构体 ……130
4.4 联合体数据类型 ……132
4.5 枚举数据类型 ……135
4.6 链表的概念 ……136
4.6.1 动态分配内存 ……137
4.6.2 单链表的建立 ……138
4.6.3 从单链表中删除节点 ……142
4.6.4 向链表中插入节点 ……145
4.7 程序范例 ……149
4.8 本章小结 ……155
习题 ……156
第 5 章 文件操作 ……161
5.1 文件的概念 ……161
5.2 文件的操作 ……161
5.2.1 文件的打开与关闭 ……162
5.2.2 文件操作的错误检测 ……164
5.2.3 文件的顺序读写 ……165
5.2.4 文件的随机读写 ……170
5.3 程序范例 ……174
5.4 本章小结 ……177
习题 ……178
第 6 章 位运算 ……182
6.1 按位取反运算 ……182
6.2 按位左移运算 ……184
6.3 按位右移运算 ……186
6.4 按位与运算 ……188
6.5 按位或运算 ……190
6.6 按位异或运算 ……192
6.7 复合位运算符 ……195
6.8 程序范例 ……195
6.9 本章小结 ……198
习题 ……198
第三部分 C++程序设计基础 ……203
第 7 章 C++中新增语法功能 ……205
7.1 引言 ……205
7.2 新增基本语法 ……205
7.3 新增函数功能 ……209
7.4 程序范例 ……220
7.5 本章小结 ……221
习题 ……221
第 8 章 类与数据抽象（一） ……226
8.1 引言 ……226
8.2 回顾 C 语言中的结构数据类型 ……227
8.3 C++中的数据类型——类 ……229
8.4 类成员的访问和作用域 ……231
8.5 访问函数和工具函数的意义 ……233
8.6 接口和实现分离的设计方法 ……235
8.7 程序范例 ……237
8.8 本章小结 ……242
习题 ……242
第 9 章 类与数据抽象（二） ……251
9.1 引言 ……251
9.2 构造函数的初始化功能 ……251
9.3 const 对象和 const 成员函数 ……254
9.4 析构函数的作用 ……258
9.5 类的复合——类可以作为其他类的成员 ……259
9.6 类的静态成员 ……263
9.7 程序范例 ……265
9.8 本章小结 ……270
习题 ……271
附录 常用的 C 语言库函数 ……276

第一部分
模块化程序设计基础

第 1 章　函数与宏定义　3

第1章 函数与宏定义

1.1 函数的概念

在C语言中，函数可分为两类，一类是由系统定义的标准函数，又称为库函数，其函数声明一般是放在系统的 include 目录下以.h 为后缀的头文件中，如在程序中要用到某个库函数，必须在调用该函数之前用#include<头文件名>命令将库函数信息包含到本程序中。

有关各类常用的库函数及所属的头文件请查阅附录，有关#include 命令将在 1.5.2 小节介绍。

另一类函数是自定义函数，这类函数是根据问题的特殊要求而设计的，自定义的函数为程序的模块化设计提供了有效的技术支撑，有利于程序的维护和扩充。

C语言程序设计的核心之一就是自定义函数，每一个函数具有独立的功能，程序通过函数的组合构成一个个模块，各模块之间的协调工作可以完成复杂的程序功能。

1.1.1 函数的定义

一个函数就是一些语句的集合，这些语句组合在一起完成一项操作，返回所需要的结果。C语言还允许程序设计人员自己定义函数，称之为自定义函数。

自定义函数的形式有如下两种。

1．现代形式：

```
[存储类型符] [返回值类型符]  函数名([形参说明表])
{
  函数语句体
}
```

2．古典形式：

```
[存储类型符] [返回值类型符]  函数名([形参表])
形参说明;
{
  函数语句体
}
```

关于函数定义的几点说明。

（1）[存储类型符]指的是函数的作用范围，它只有两种形式：static 和 extern。static 说明函数只能作用于其所在的源文件，用 static 说明的函数又称为内部函数；extern 说明函数可被其他源文件中的函数调用，用 extern 说明的函数，又称为外部函数。默认情况为 extern。

（2）[返回值类型符]指的是函数体语句执行完成后，函数返回值的类型，如 int，float，char 等，若函数无返回值，则用空类型 void 来定义函数的返回值。默认情况为 int 型（有些编译器不支持默认情况）。

（3）函数名由任何合法的标识符构成。为了增强程序的可读性，建议函数名的命名与函数内容有一定关系，以养成良好的编程风格。

（4）在第 1 种函数定义的形式中，[形参说明表]是一系列用逗号分开的形参变量说明。如：int x, int y, int z 表示形参变量有 3 个：x, y, z。它们的类型都是 int 型的，不能写成：int x, y, z。

（5）在第 2 种函数定义的形式中，[形参表]是一系列用逗号分开的形参变量，如 x, y, z 表示有 3 个形参变量，它的类型通过形参说明语句来说明，如：int x, y, z;。

[形参说明表]或[形参表]都可以缺省，缺省时表示函数无参数。

（6）函数语句体是放在一对花括号{ }中，由局部数据类型描述和功能实现两部分组成。局部数据类型描述是由类型定义语句完成的，用来说明函数中局部变量的数据类型；功能实现部分可由顺序语句、分支语句、循环语句、函数调用语句和函数返回语句等语句构成，是函数的主体部分。

（7）函数返回语句的形式有以下两种。

① 函数无返回值的情况

```
return;
```

在函数无返回值的情况下，也可以不写 return 语句，函数执行完毕后，自动回到调用函数处，继续执行下面的语句。

② 函数有返回值的情况

```
return (表达式的值);
```

在第②种情况下要注意“表达式的值”的类型必须与函数返回值的类型一致。

例如，求两个任意整数的绝对值的和，用函数 abs_sum()实现。

函数定义如下：

```
int abs_sum(int m, int n)
{
   if (m<0)
       m=-m;
   if(n<0)
       n=-n;
   return (m+n);
}
```

当然，也可以直接调用系统函数来计算 m 和 n 的绝对值之和，函数也可以写成这样：

```
int abs_sum(int m, int n)
```

```
{
    return (abs(m)+abs(n));
}
```

求整数的绝对值的函数 abs()是在头文件 math.h 中声明的。

1.1.2　函数的声明和调用

在大多数情况下，程序中使用自定义的函数之前要先进行函数声明，才能在程序中调用。

1. 函数的声明

函数声明语句的一般形式为：

```
[存储类型符]  [返回值类型符]  函数名([形参说明表]);
```

如：`int abs_sun(int m, int n);`

2. 函数调用

一个函数写好后，若不通过函数调用，是不会发挥任何作用的，函数调用是通过函数调用语句来实现的，它分为以下两种形式。

① 函数无返回值的函数调用语句：

```
函数名([实参表]);
```

② 函数有返回值的函数调用语句：

```
变量名=函数名([实参表]);
```

该变量名的类型必须与函数的返回值类型相同。

不论是哪种情况，函数调用时都会去执行函数中的语句内容，函数执行完毕后，回到函数的调用处，继续执行程序中函数调用后面的语句。

例如：

```
⋮
int x=5, y=-10;
int z;
⋮
z=abs_sum(x, y);  /*函数调用 */
⋮
```

1.1.3　函数的传值方式

在调用函数时，若函数是有参数的，则必须采用实参表将每一个实参的值相应地传递给每一个形参变量，形参变量在接收到实参表传过来的值时，会在内存临时开辟新的空间，以保存形参变量的值，当函数执行完毕时，这些临时开辟的内存空间会被释放，并且形参的值在函数中不论是否发生变化，都不会影响到实参变量的值的变化，这就是函数的传值方式。

自定义函数在程序中的使用顺序有以下两种形式。

第 1 种：先进行函数声明，再进行函数调用，函数定义放在 main()函数的后面。函数声明应放在函数调用之前，具体位置与编译环境有关。

第 2 种：函数定义放在 main()函数的前面，再进行函数调用。在这种情况下，可以不进

行函数声明。

【例 1-1】编写程序，通过调用函数 int abs_sum (int a,int b)，求任意 2 个整数的绝对值的和。

分析：2 个整数的绝对值的和仍然是整型数，函数调用时需要一个整型变量来接收函数的返回值。

程序如下：

```
/*example1_1.c 自定义函数，求两整数绝对值的和*/
#include <stdio.h>
int abs_sum(int m,int n);  /*函数声明*/
main()
{
    int x,y,z;
    scanf("%d%d",&x,&y);
    z=abs_sum(x,y);      /*函数调用*/
    printf("|%d|+|%d|=%d\n",x,y,z);
}
int abs_sum(int m,int n)   /*函数定义*/
{
      if(m<0)
           m=-m;
      if(n<0)
           n=-n;
      return m+n;
}
```

程序运行结果:

```
7 -12↵
|7|+|-12|=19
```

在程序中，若将函数定义放在函数调用之前，则可以不需要函数声明语句，上面的程序也可以写成如下的形式：

```
/*example1_1a.c 自定义函数，求两整数绝对值的和*/
#include <stdio.h>
int abs_sum(int m,int n)   /*函数定义*/
{
    if(m<0)
         m=-m;
    if(n<0)
         n=-n;
    return m+n;
}
main()
{
    int x,y,z;
    scanf("%d%d",&x,&y);
    z=abs_sum(x,y);      /*函数调用*/
    printf("|%d|+|%d|=%d\n",x,y,z);
}
```

上面这两个程序 example1_1.c 和 example1_1a.c 的功能是相同的。

用传值方式调用函数时，实参也可以是函数调用语句，请看下面的程序。

【例 1-2】编写程序，通过调用函数 int abs_sum(int a, int b)，求任意 3 个整数的绝对值的和。

分析：因为 2 个数绝对值得和还是整数，因此，也可以将函数调用作为函数的实参。

程序如下：

```
/*example1_2.c 调用函数求 3 个整数绝对值的和*/
#include <stdio.h>
int abs_sum(int m,int n);   /*函数声明*/
main()
{    int x,y,z,sum;
     scanf("%d%d%d",&x,&y,&z);
     sum=abs_sum(abs_sum(x,y),z);    /*函数调用*/
     printf("|%d|+|%d|+|%d|=%d\n",x,y,z,sum);
}
int abs_sum(int m,int n)   /*函数定义*/
{     if(m<0)
           m=-m;
      if(n<0)
           n=-n;
      return m+n;
}
```

程序运行结果：

```
-7 12 -5↵
|-7|+|12|+|-5|=24
```

当然，解决这个问题也可以通过两次调用函数来求得 3 个数绝对值的和：

```
sum=abs_sum(x,y);
sum=abs_sum(sum,z);
```

还可以通过设计一个新的函数来实现求 3 个数绝对值的和：

```
int abs_sum(int a,int b,int c);
```

另外，若函数有返回值，调用时又没有把它赋给某个变量，C 语言的语法并不报错，程序仍然可以执行，但函数的返回值有可能会被丢失，在程序中要防止这种情况的发生。请看下面的例子。

【例 1-3】编写程序，求任意两数的乘积。

分析：自定义一个函数 double mul(double a, double b)，用于求 2 个数的乘积，函数的返回值为 double 型。

程序如下：

```
/*example1_3.c  求 2 个数的乘积*/
#include <stdio.h>
float mul(float a,float b);   /*函数声明*/
main()
{
     float x,y,z;
     printf("Please enter the value of x and y:\n");
     scanf("%f %f",&x,&y);
     z=mul(x,y);    /* 1. 函数调用，有变量接收返回值*/
```

```
        printf("1--x=%4.1f,y=%4.1f, ",x,y);
        printf("(%4.1f)*(%4.1f)=%4.1f\n",x,y,z);
        x=x+10;
        y=y-10;
        printf("2--x=%4.1f,y=%4.1f, ",x,y);
        mul(x,y);       /* 2. 函数调用，无变量接收返回值*/
        printf("(%4.1f)*(%4.1f)=%4.1f\n",x,y,z);
        x=x*2;
        y=y*2;
        printf("3--x=%4.1f,y=%4.1f, ",x,y);
        printf("(%4.1f)*(%4.1f)=%4.1f\n",x,y,mul(x,y)); /* 3. 函数调用，函数的返回值作为
参数*/
    }
    float mul(float a,float b)   /*函数定义*/
    {
        return a*b;
    }
```

程序运行结果：

```
Please enter the value of x and y:
5 6↵
1--x= 5.0,y= 6.0, ( 5.0)*( 6.0)=30.0
2--x=15.0,y=-4.0, (15.0)*(-4.0)=30.0
3--x=30.0,y=-8.0, (30.0)*(-8.0)=-240.0
```

在上面这个程序中，第1处函数调用将函数的返回值赋给变量z，得到正确的计算结果；第2处调用函数后没有将函数的返回值赋给任何变量，函数的返回值被丢失，无法将函数的计算结果输出；第3处调用函数是将函数的返回值作为printf()函数的参数，得到正确的计算结果。

对于有返回值的函数，在调用函数时，一般是要用相应的变量来接收函数的返回值，也可将函数作为另一个函数的参数。

1.2 变量的作用域和存储类型

变量的作用域指的是在程序中能引用该变量的范围，针对变量不同的作用域，可把变量分为局部变量和全局变量。

C语言程序中根据变量的作用域不同，可分为局部变量和全局变量两种。

1. 变量的作用域

局部变量：在函数内部或某个控制块的内部定义的变量为局部变量，局部变量的有效范围只限于本函数内部，退出函数，该变量自动失效。局部变量所具有的这种特性使程序的模块增强了独立性。

全局变量：在函数外面定义的变量称为全局变量，全局变量的作用域是从该变量定义的位置开始，直到源文件结束。在同一文件中的所有函数都可以引用全局变量。全局变量所具有的这种特性可以增强各函数间数据的联系。

局部变量和全局变量的作用域如图1-1所示。

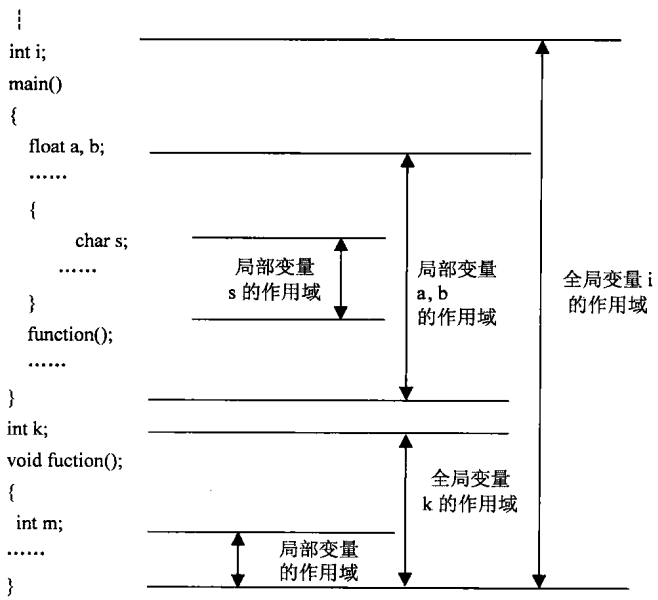


图 1-1　局部变量和全局变量的作用域

2. 变量的存储类型

变量的存储类型指的是变量的存储属性，它说明了变量占用存储空间的区域。在内存中，供用户使用的存储区由程序区、静态存储区和动态存储区 3 部分组成。变量的存储类型有 auto 型、register 型、static 型和 extern 型 4 种。

auto 型变量存储在内存的动态存储区；register 型变量存储在寄存器；static 型变量和 extern 型变量存储在静态存储器。

局部变量的存储类型默认值为 auto 型，全局变量的存储类型默认值为 extern 型。

auto 型变量 register 型只用于定义局部变量。

static 型既可定义局部变量，又可定义全局变量。定义局部变量时，局部变量的值将被保留，若定义时没有赋初值，则系统会自动为其赋 0 值；若定义全局变量时，其有效范围为它所在的源文件，则其他源文件不能使用。

【例 1-4】 了解变量作用域。阅读下面的程序，注意区分局部变量和全局变量的作用域。

```
/*example1_4.c  了解变量的作用域*/
#include <stdio.h>
void a( void );
void b( void );
void c( void );
int x = 1;
int main()
{
    int x = 5;
    printf("x in main is %d\n", x );
    {
        int x = 7;
        printf( "x in inner scope of main is %d\n", x );
    }
    printf( "x in main is %d\n", x );
    a();
```

```
    b();
    c();
    a();
    b();
    c();
    printf( "\nx in main is %d\n", x );
    return 0;
}
void a( void )
{
    int x = 25;
    printf("\nThis is function a:\n");
    printf( "x in a is: %d\n", x );
    ++x;
    printf( "++x in a is: %d\n", x );
}
void b( void )
{
    static int x = 50;
    printf("\nThis is function b:\n");
    printf( "static x in b is: %d\n", x );
    ++x;
    printf( "static ++x in b is:%d\n", x );
}
void c( void )
{
    printf("\nThis is function c:\n");
    printf( "global x in c is:%d\n", x );
    ++x;
    printf( "global ++x in c is:%d\n", x );
}
```

程序运行结果：

```
x in main is 5
x in inner scope of main is 7
x in main is 5

This is function a:
x in a is: 25
++x in a is: 26

This is function b:
static x in b is: 50
static ++x in b is:51

This is function c:
global x in c is:1
global ++x in c is:2

This is function a:
x in a is: 25
++x in a is: 26

This is function b:
static x in b is: 51
static ++x in b is:52
```