北京希望电脑公司

# 标准C语言

# 与

# C++编程基础

张玉亭　胡祥义
田立宪　郭爱民　编写

# 标准C语言
# 与
# C++编程基础

张玉亭 胡祥义
田立宪 郭爱民 编写



北京希望电脑公司

## 内 容 简 介

本书以大量实例详细地讲解了C编程的各个方面，内容包括：语法、I/O编程基础及编程技巧、标准库函数的使用、扩展特性、可移植代码及ANSI C等，并全面地介绍了C编程环境以及C++编程基础。

本书的特点在于由浅入深，通过对某一题目列举短而精的程序实例，使读者易于掌握C语言的实质以及包括ANSIC和C++在内的标准C编程。

本书是学习C语言必备的参考工具书，不仅适于高等院校有关专业的师生阅读，也适于作自学者和技术人员的自修与培训教材。

**需要本书的用户，请直接与北京8721信箱联系，电话2562329，邮码100080。**

**标准C语言与C++编程基础**

张玉亭 胡祥义 田立宪 郭爱民 编写

希望 审校

*

北京希望电脑公司 建华印刷厂印刷

开本：787×1092 1/16 印张：30.875 字数：739千字

京准印字：3313—90313

内部成本价：17.00元

# 编 者 的 话

本书将介绍一种极适合创建和管理计算机环境的语言——C语言。C语言可作为高级汇编语言，它易于控制计算机软件，并以一种适合快速编写丰富而复杂程序的强功能高级语言而著称。简而言之，用C语言可以编写从商用软件到操作系统的各种程序。

通过阅读本书，可了解现代C编程语言的所有基础内容。具体讲，本书具有以下几个特点：

〈1〉以一定的深度涉及C语言的基础内容，使读者学会编写C程序，或在原基础上有进一步的提高。

〈2〉本书没有列举大而复杂的程序，而只是针对某一标题列举短而精的程序段，从而更易于理解和掌握。

〈3〉本书更侧重于ANSI C，并以此获取理解所有C编程所需的基本概念。因为，只有掌握了这些概念，才能为进一步学习C++打下坚实的基础，当然，本书还要讲解有关C++的内容。

总之，本书最为重要的目的是介绍包括ANSI C和C++在内的标准C编程，从而有助于提高读者的编程水平。可以说，本书是学习C语言必备的参考工具书。

在本书的出版过程中，得到北京希望电脑公司的大力支持和帮助，在此表示衷心感谢。

编　　者

# 目　录

## 第二部分 使用标准C函数库

## 第三部分　C++编程基础

# 第一部分 介绍标准C语言

## 第一章 C语言概述

如果想要学习标准C编程语言的话，那么从本书中便可获取到所需的一切内容。对C编程的初学者来说，读过本章后便可对C语言有个大致的了解，而对有经验的C编程人员来说，则可跳过这一章。

### 1.1 C语言介绍

要掌握一种新的语言，就必须完整地了解该语言的全貌以及正确的信息。通过本章的学习，我们可以掌握以下内新：

- □ C语言从K&R到C++的历史。
- □ C语言与其它语言之比较。
- □ C语言程序的形式。
- □ C程序的基本要素。
- □ 编写C程序的过程。
- □ 编写C程序需要哪些工具。

#### 1.1.1 C语言的简要历史

C语言从何处而来？为什么它得以发展和普及？它适用于什么？是否了解这些问题的答案并不会直接影响学习C语言，但简要地了解这些内容则有助于我们更好地掌握C语言。

1978年，Prentice-Hall出版了由Brian W.Kernighan和DennisM.Ritchie编写的《C编程语言》一书。自此，开始了C的开发。

起初，Dennis Ritehie开发C语言是用于UNIX操作系统，在DEC PDP-11计算机上运行，除编译程序本身外，UNIX操作系统和大多数实用程序都是用C编写的。

C的前身是BCPL语言，后者在欧洲目前还有人使用。BCPL语言是由Martin Richards设计的。该语言深刻地影响了Ken Thompson设计的B语言风格。B语言于1970年发展成为C语言。

随着微型机的大众化，各种C语言的实用版本纷纷推出。在不同的版本上，C语言的源程序可保持高度兼容。另一方面，由于没有一个标准，它们彼此之间也有一定的差别。为此，美国国家标准局成立了一个委员会，于1983年夏初建立了一个C语言标准（ANSI C）。

随着C语言的日益普及以及设计任务不断复杂，技术要求也随之更高。为此，Bjarne Stroustrup开发了面向对象语言C++。

C++并不完全偏离ANSI C。事实上，ANSI C可看作是C++的一个子集，因为在概念和方法上两者互有借鉴。不过，C++支持一些附加工具：封装、类、数据隐藏和函数及算符重载。

#### 1.1.2 C语言与其它语言之比较

象其它语言一样，C语言有优点，也存在不足。下面我们简要地将C语言与其它语言做

一比较。

1.1.2.1 C语言与汇编语言

在生成快速运行时代码方面，无任何高级语言可与汇编语言相媲美。不过，在所有高级语言中，C也许最接近于汇编语言。起初，C作为系统程序员的一种语言与系统的低级成份保持联系，并可用作高级汇编语言。

C语言的灵活性和强功能使其速度降低，因此要比汇编语言慢。C运行时代码始终比可兼容的汇编语言程序长，这就影响了程序的执行速度。

1.1.2.2 C语言与Pascal语言

起初，Pascal是作为一种工具，用于讲解程序设计基础，并讲解硬件和操作系统环境的概念。目前有很多人使用Pascal，但该语言与C和汇编语言不同，很少用于开发像操作系统这样的项目。

另一方面，由于Pascal与C语言极为相似，所以Pascal程序员学习C语言不会遇到太多的麻烦。

1.1.2.3 C语言与COBOL

C与COBOL之间几乎不存在语言和构造的相似之处。C程序始终比COBOL产生更快且更紧凑的运行时代码。

不过，对商用编程任务来说，使用COBOL具有一大优点：内部可获取十进制数据并进行十进制运算。不过，目前Turbo C++也有内部类，支持十进制数据和运算。

### 1.1.3 C形式与标点

读C程序是掌握C语言的最重要步骤。如果不能读程序，就不能编写程序。

1.1.3.1 C形式

假设我们坐在计算机旁，且文本编辑程序正在运行C程序，那么，我们可以看到这样或那样的显示内容。

我们可以非常准确地定义C源程序为一系列说明。说明告知编译程序所需知道的内容。我们可以编写一个说明告知编译程序如何执行如下内容：

- □ 认可所创建的名字。
- □ 为变量或常量这样的数据对象分配内存，并有选择地初始化其值。
- □ 定义函数的功能。
- □ 解释特殊的数据类型。

严格地说，语句（执行操作的指令）仅是函数说明的一部分。它们绝不出现在函数外，且不是说明。函数是语句集合。从编译程序的角度看，整个程序就是一个说明集合。该集合告知编译程序如何生成目标代码，而不是告知编译程序如何执行程序所要执行的主要任务。编译程序不知道我们如何想，只知道我们告诉它的内容。

1.1.3.2 C标点

C中标点是极为重要的。我们可以把标点用作了解程序功能一个线索。现在，我们可以看一看例子1.1 first.c，看一看标点的作用。

---

**快速参考：C标点**

所有C标点都是预定义的。只有如下标点是有效的：

[ ] ( ) { * } , : = ; … # ##

我们只能把这些字符用作标点，当然在字符串中可用作数据。

＃和＃＃实际上是预处理算符。它们并不属于标点列表。

---

首先应注意到，举例1.1中的多数行以分号（；）结尾。分号用于结尾C中的所有单语句和说明。例如，在如下语句中：

```
i=i * 2;
```

i的值乘2并放回到变量的存储单元中。

我们可以标识语句块。语句块由一个或多个单语句组成，并用花括号括起来。如下函数体便是语句块的一个例子：

```
int calc( int i )
{
  i = i * 2;
  i++;
  return i;
}
```

仔细看一看前几行。尽管每条单语句都用一个分号结尾，但语句块在花括号（{ }）后并没有以分号结束。

接下来，我们可清楚地分辨出编译程序伪指令。注意，它们都以＃符号开头，如：

```
#include <stdlib.h>
```

编译程序伪指令不属于要转换为目标代码的源代码。它们告知编译程序如何执行翻译进程。由于在执行任意实际编译之前必须发现这些伪指令并进行解释，所以它们称为预处理伪指令。伪指令的结尾并不需要分号。如果语句含有一个分号，那么应另有别的原因（详见第四章）。

最后，注释在C程序可读性方面具有重要作用。注释由特殊的字符序列分隔。该字符序列既不是标点也不是C算符。注释以/*开头并以*/结尾（见举例1.1的前两行）。

注释的唯一目的就是有助于程序的可读性。在进行翻译之前编译程序用空白取代整个注释。

---

**字符、数组和串：**

不提到串就不可能讨论C语言。为了理解串，我们还必须对字符和数组有所了解。这三种数据类型有如下特性：

- ☐ 字符是数据变量，可以打印、显示或控制显示设备。字母、数字和标点都是显示字符。控制字符与显示字符具有相同的内部结构，但一般来说它们是不可见的，而控制着其它字符在显示设备上所出现的形式。第四章将详解字符。
- ☐ 数组是一组数据变量，都存储在计算机内存中。数组可由任意其它类变量构成，其中包括字符。例如，整型数组可这样定义：

  ```
  int counts[10]; /* An array of 10 integers */
  ```

- ☐ 串是一种特殊数组，由字符构成。我们定义串就象定义任何其它数组一样。例如，我们可以定义一个姓名串：

  ```
  char name[40];
  ```

□ 该串最多只包含39个显示字符，因为最后一个数组元素（位置）必须为null（空）字符保留。null字符是一个特殊字符，其数值为零。null字符表示串结束且不可显示。要用文字值初始化串，不必含有null字符，如下所示：

```
char name[40]= "John Aloysius Doe";
```

## 1.2 C程序的各个部分

既然我们已知道C程序的形式，那么我们现在看一看C程序的各个成份。举例1.1是程序first.c的源代码。这是一个短小的程序，但它包含了程序的基本成份。

例1.1 first.c程序

```
1  /* -----        FIRST.C        ----- */
2  /* ----- A short C program ----- */
3
4  #include <stdio.h>
5  #include <stdlib.h>
6
7  int calc(,int i );
8
9  main()
10 {
11   char number_in[4], number_out[5];
12   int i,j;
13
14   puts( "Enter up to a two digit
15   gets( number_in );
16   i = atoi( number_in );
17   j = calc( i );
18   itoa( j, number_out, 10 );
19   puts( "The calculations on the integer yield:" );
20   puts( number_out );
21 }
22
23 int calc( int i )
24 {
25   i = i * 2;
26   i++;
27   return i;
28 }
```

如果初次接触C程序，最好花一定时间看一看例1.1，以了解C程序的形式。first.c程序获取键入的一个字符串，并把它转换为一个数字。之后，它调用一个函数并返回结果。最后，新值转换为字符串并在控制台上显示。下面，我们根据行号对该程序进行分析：

□ 第1～2行是注释。它提供有关该程序的某些基本信息。

□ 第3行说明我们可在程序的任何地方留出空行，以提高其可读性。

□ 第4～5行是编译程序伪指令。在该程序中，它告知编译器，在命名文件中可发现某些非常重要的信息。这些文件称为头文件，因为它们通常出现在程序开头处。

□ 第7行含有函数原型。它告知编译程序，该程序含有calc函数的定义。该函数接收整型参数并将整型值返回到程序中的任何地方。函数一旦定义就可在程序的许多地方多次调用（但以后我们会看到，不是从任何地方调用）。

□ 第9～21行包含程序的main（）函数。每个程序都有一个main（）函数。当运行程序时，首先要执行main函数。在举例1.1中，main（）函数获取输入数据，将其转换为数值，将该数值传送给calc（），以便处理，接收结果并在屏幕上显示结果。

□ 第23～28行含有calc（）的函数定义。

### 1.2.1 包含头文件

例1.2 的第4和5行提供了#include预处理伪指令的一个例子。

#### 1.2.1.1 包含源文件

#include伪指令使编译程序执行我们所希望的操作：包含另一个源文件。当预处理时，在开始翻译为目标代码之前，编译程序寻找尖括号（小于和大于号）中命名的源文件，读取并插入到include位置处的当前源中（见图1.1）。

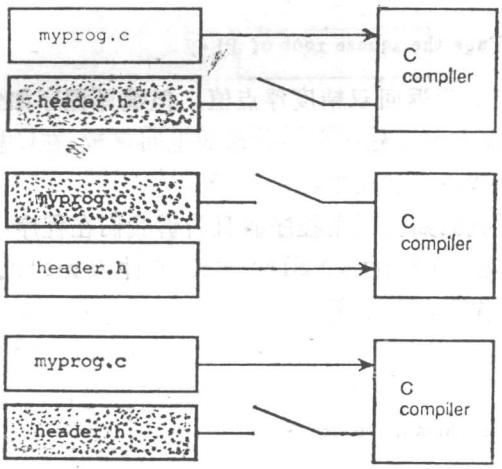


图 1.1 include伪指令

我们可以把长的源程序分为多重文本文件并用#include语句引用它们，以便编译。不过应注意，这不是处理复杂项目的很有效方法。在本章后面将谈及如何建立复杂项目，并在第五章详细讨论。

#### 1.2.1.2 头文件和内部函数

举例1.1中的两个#include伪指令说明了编译程序使其库函数和其它功能为我们所用。这些资源对我们编写良好而有用的程序来说是极为重要的。

---

**快速参考：头文件的典型用法**

Bjarne Strosutrup（C++的创始人）建议了几种头文件的使用方法。下面是头文件应包含的内容列表：

```
类型定义：  Struct parts{int stocknum, loc};
函数说明：  int area (int length, int width) ;
数据说明：  extern unsigned master-switch;
常量定义：  const int maxsize  32767;
    枚举：enum toys{top, jacks, slinky};
```

包含伪指令：#include < graphics.h >
宏定义：#define TRUE 1
注释：/* This is a header file */

---

当我们购买了编译程序，我们会获得许多程序。其中，我们可以获取到功能很强的函数库。这些函数不属于C语言，而是装入一个软件包中。

我们拥有了这些函数可以干些什么呢？编译程序手册会告诉我们有关这些函数的一切：什么是函数，它们如何操作，如何设置对它们的调用。理解这些内容是至关重要的。函数有时像变量一样操作；它们返回拥有某种类型的值。这就是为什么我们可以在赋值语句的右边使用函数的原因，如下所示：

```
/*What will happen here? */
double x;          /*Define a variable to use */
...
x=sqrt(3.1415926); /*Take the square root of PI*/
```

平方根函数sqrt( )定义为返回双精度浮点值，但编译程序如何知道这一点呢？如果我们不说明函数便使用之，那么编译程序会假设函数返回整数。在C中，整型是缺省类型。从第三章开始，我们会看到哪些是合法类型。

编译程序手册不仅告诉我们如何调用程序而且告诉我们在程序中包含了哪些头文件，以便函数可正确运行。平方根函数的ANSI标准头文件是math.h。我们可通过添加#include伪指令修改上述代码，如下所示：

```
#include < math.h >
...
double x;      /*define a variable to use */
...
x=sqrt(3.1415926); /*Take the square root of PI */
```

1.2.1.3 放置包含文件

我们把#include伪指令放在程序中的什么地方呢？C语法允许放置在任何地方。不过，C程序员希望我们在使用之前对其加以说明。最好把该伪指令放在源文件的顶端。

如果我们不使用特殊头文件所说明的任意函数、宏或全局变量，那么不必包含之，但这样做也无关紧要。例如，我们常习惯于包含stdlib.h和stdio.h。如果不需要这两个头文件的话，也不会影响程序。

如果忘记了一个必要的头文件怎么办？那么，要么在编译时，要么在链接时会出现奇怪的报错信息。

1.2.1.4 包含的两种方法

例1.1仅说明了编写#include伪指令的一种方法。要包含的文件写在尖括号( < > )中，这尖括号也可用作大于和小于算符。要特别注意，不要把任意额外空格放在括号内。如果在括号内有多余空格，编译程序就找不到该文件。例如：

```
#include < stddef.h > /* The correct method */
```

与如下形式不一样

```
#include(stddef.h)/* wrong! */
```

ANSI标准规定，用尖括号写一个包含文件使编译程序查找命名文件。因此，仅在特定目录中查找，这些目录假设仅包含头文件。请见图1.2。

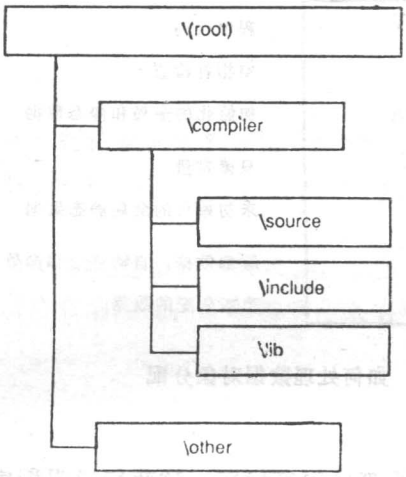


图 1.2 头文件和库目录

如图1.2所示，/source目录是当前目录。我们可以用DOS SET命令定义PATH，INCLUDE和LIB目录说明。

编写#include的第二种方法是用双引号括起文件名，如：

#include "myhdir.h"

注意，像第一种方法一样，引号中不要包含额外空格。

有关多重源文件的注释：当开始编写复杂程序时，需了解如何把程序部分包含在分别编译和链接的文件中。当分别编译一组函数时，需为它们创建一个可包含在其它源文件中的头文件，这样其它函数就知道如何存取这些函数和公用变量。

### 1.2.2 定义C数据

不管是用C还是用其它语言编写的所有程序都要做一件事：处理数据。这一数据的某些内容仅在程序内；其它内容由输入和输出函数放入RAM中。不管是哪种情况，编译程序必须知道如何设置程序内存，以序放该数据。在C文件中，数据项常称为对象。

#### 1.2.2.1 为数据分配内存

是说明一个变量还是使用一个常量，编译程序都应分配内存。这种内存成为程序的一部分。当编译程序向磁盘写运行时代码时，这种数据（写在函数外的所有常量和变量）就要占用文件中的某些空间。

通过使用malloc( )，calloc( )，realloc( )和相关函数，我们可在运行程序时动态地为数据分配内存。这组函数从操作系统请求内存块。动态分配的数据并不占用磁盘上程序文件中的空间。

不过，对各种数据，编译程序必须构造程序，这样，运行时程序代码可跟踪任意内容。因此，有些数据放在特殊的内存部分中。在80x86处理器系列中，数据可放在代码段，数据

段或堆栈段中。这要根据如何使用数据而定。图1.3说明了编译程序如何在80x86处理器上为数据分配内存。在图1.3所示的IBM及其兼容机上小存储模式的段结构中，数据存储在_DATA，CONST，_BSS，STACK和Heap段中。

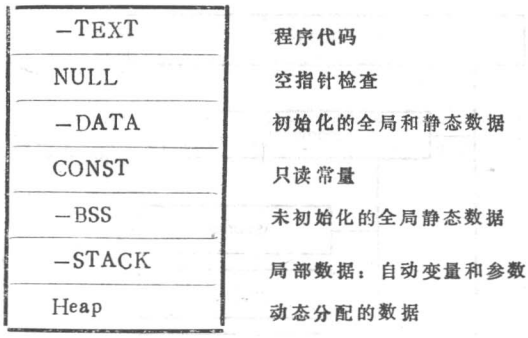


图 1.3 如何处理数据对象分配

1.2.2.2 把它放在何处？

见first.c（举例1.1）中的第11和第12行。这两行说明程序要用数据对象。第11行说明两个串（它们是字符数组——可显示的字符序列）；第12行说明两个整数。

现注意这两个说明的位置——它们在main（）函数体内。只有main（）函数才可直接存取这些变量，要么对它们进行检查，要么进行修改。我们还可在函数外说明变量。这样的变量可由源文件中的所有函数存取。在C中有全局变量和局部变量之分：

- 局部变量仅在它们所定义的块（函数或块语句）中可见。这些变量拥有动态持续时间——它们仅当块为活动的（实际在运行）时才存在。块语句是用花括号括起来的任意内容，因此，函数体是块。
- 全局变量是源文件中全局可见的，并可由其它源文件可见。在缺省情况下，全局变量拥有静态持续时间——不管程序的哪部分是活动的，它们都有效。

**注意**，即便不用花括号把整个源文件括起来，但整个文件被视为一个块。这一点有助于我们理解这样的一般规则：变量仅在它们所定义的块中可见。

### 1.2.3 main函数的作用

我们可以编写任意函数并给它们起名，但不能使用main这个名字。main（）函数是任意程序中所需的唯一C函数。例1.1的第9～21行包含first.c程序的main（）函数。需要main（）函数有多种原因。

1.2.3.1 main（）进入点

假设我们已编译了程序，未出现错误，并准备运行之。现在，问这样一个问题——计算机如何知道启动程序中的哪条可执行指令？

main（）函数的一个主要用途就是：它是程序的主要进入点。编译程序始终设置可执行文件，因此，main（）先运行。

1.2.3.2 设置环境

编译程序创建一个程序存根（短小程序）。该存根在main（）中代码之前执行，并依据所用机器和操作系统执行不同的操作。一般来说，存根存取在系统命令中所键入的命令

行参数，以执行程序。它还为程序设置系统环境。

我们所提供的内容完全依赖于我们希望程序所执行的内容。事实上，我们不必做任何事情。例如，如下程序可正确地编译并运行，而无任何错误：

```
main()/*empty.c*/
{
}
```

该程序什么也不做。

通常来说，main（）应执行依据应用程序所设置的任何操作：初始化全局变量，打开文件，调用第一个启动函数，等等。

1.2.3.3 程序结束

main（）的最后一个主要功能是提供一种机制，以终止程序的执行。正如我们在前面的程序段（empty.c）中所注意到的那样，不需要什么语句来完成此任务。请见如下程序段：

```
main()/*ending.c*/
{
.../*Some statements go here*/
exit(0);
}
```

ending.c程序终止程序的执行。exit（0）函数调用一个库函数，将控制返回给系统，并将返回代码0（或其它内容）传送给系统。

### 1.2.4 提供用户定义函数

例1.1的第23～28行包含一个用户定义函数：calc（）。这样的函数不是必不可少的，但C程序都含有这样的函数。

1.2.4.1 函数的作用就像数据一样

在我们说明并定义一个函数后，便可在表达式中引用。正如例1.1的第17行那样：

```
j=calc(i);
```

引用calc（）将占用该语句中的一个位置，这与引用变量或常量一样。即：编译程序假设函数有一个值。在calc（）中，return语句使i的计算值传送给调用例程。在这种情况下，函数的作用就像数据对象一样。

我们还可这样调用函数：

```
calc(i);
```

调用不使用其结果的函数是完全合法的。函数返回的任意值只是不用而已。

1.2.4.2 函数放在何处

将函数放在程序的什么地方取决于该函数在何处说明以及程序的哪些部分将调用该函数。

- □ 我们可以但不应该在函数说明之前引用该函数。这样做的话会导致某些错误操作。
- □ 我们可以通过定义，提供函数体和函数参数说明函数。在定义后的任何地方都可引用该函数。
- □ 通过先在程序中编写函数原型，以后再加以定义，以此说明函数。函数原型就是无体的标题行，如：