

C/C++程序设计

C/C++程序设计

(2019年修订)

● 主编 李彦明
● 主审 王连相

C/C++程序设计

C/C++
Chengxu Sheji



策划编辑：高 锐
责任编辑：谷建湘
特约编辑：李 健
装帧设计：黄燕美



定价：49.00元

国防科技大学出版社

国防科技大学出版社

前 言

本书是面向独立学院工科类专业学生编写的《C/C++ 程序设计》教材。

随着我国高等教育的大众化和办学层次的多样化,因材施教已成为当前教学改革和课程建设的重要内容之一。本书根据国家质量工程全面提高本科生素质教育的指导思想,结合工科本科计算机基础教学的基本要求,在独立学院多年教学经验的基础上编写而成。近年来的教学实践与研究表明,独立学院的计算机基础教育必须与独立学院的人才培养层次和模式紧密联系。因而本书的编写不仅强调学生掌握 C 语言的基本概念、基本方法,而且强调培养学生利用所学的编程技巧解决一些实际的工程问题。

C 语言是当今世界上最流行的程序设计语言之一,它以功能强大、表达灵活、使用方便和移植性好而受用户青睐。无论在设计系统软件、进行数据处理、数值计算及教学等领域都得到了广泛的应用。21 世纪对信息技术人才的综合能力要求越来越高,需求也越来越多。随着现代编程技术的广泛应用与发展,C/C++ 程序设计必将更为普及。学习和掌握 C/C++ 语言已成为现代软件开发人员的必备知识之一。本书针对现代教育教学改革理念,结合独立学院的人才培养层次和模式,在提高教学效率的同时,力求提高学生综合实践的能力。我们是在多年软件开发和 C/C++ 程序设计教学实践经验的基础上,根据现代高校教学改革特有的情况,探讨了现代计算机教学的规律,收集分析了大量的教学文献,并基于实际应用编写了本书和配套的《C/C++ 程序设计上机指导与习题集》。本书内容翔实,概念清晰,通过实例分析,可以引导读者尽快掌握 C/C++ 程序设计的实际应用技巧。作为教学用书,本书内容编排比较紧凑,可以帮助读者系统全面地了解 C/C++ 程序设计的重点和要点。

本书共分为 10 章,其中第 1 章由龚翔负责编写;第 2 章、第 3 章、第 5 章、第 6 章由李彦明负责编写;第 4 章和第 9 章由杨书鸿负责编写;第 7 章由陈文娟负责编写;第 8 章由马生菊负责编写;第 10 章由张清负责编写。

本书在编写过程中得到了兰州理工大学技术工程学院的大力支持和帮助。全书由李彦明统稿并定稿,甘肃省高等学校计算机教学指导委员会委员王连相主审;谢黎明教授对本书的编写给予了很大的指导和帮助。在此一并表示衷心的感谢。

本教材是由编者在多年从事计算机基础教育的基础上编写而成,由于编者能力有限,书中难免存在错误和不当之处,在此恳请广大读者批评指正,并提出宝贵意见。

编 者

目 录

第 1 章 C 语言概述	1
1.1 C 语言的起源	1
1.2 C 语言是中级语言	1
1.3 C 语言是结构化语言	2
1.4 C 语言是面向程序员的语言	3
1.5 编译和解释	4
1.6 C 语言程序结构	5
1.6.1 库和链接	6
1.6.2 分别编译	7
1.6.3 编译 C 语言程序	7
1.6.4 C 语言的内存映象	8
1.7 术语	8
习题	9
第 2 章 数据类型和表达式	10
2.1 数据类型	10
2.2 保留字和标识符	11
2.2.1 标识符	11
2.2.2 保留字	11
2.2.3 选择合适的标识符	12
2.2.4 常量与变量	12
2.3 基本数据类型	13
2.3.1 数据的内部表示	14
2.3.2 字符型	15
2.3.3 整型	17
2.3.4 浮点类型和双精度类型	18
2.3.5 字符串常量	20
2.3.6 符号常量	20
2.4 运算符与表达式	21
2.4.1 表达式	21
2.4.2 算术运算符及算术表达式	22
2.4.3 关系运算符及关系运算表达式	23

2.4.4	逻辑运算符及逻辑表达式	24
2.4.5	赋值表达式	24
2.4.6	逗号表达式	25
2.4.7	表达式的运算顺序	25
2.4.8	条件运算	25
2.4.9	sizeof 运算	26
2.4.10	运算符优先级与结合性质	27
2.5	数据类型转换	28
2.5.1	隐式类型转换	28
2.5.2	强制类型转换	29
2.6	位运算	29
2.7	数组和指针	32
2.7.1	数组	32
2.7.2	指针	33
2.7.3	指针和数组的简单用法	35
2.8	简单应用程序举例	38
	本章小结	40
	习题	40
第3章	结构化程序设计	43
3.1	顺序结构程序设计	43
3.1.1	赋值语句	43
3.1.2	数据输入	45
3.1.3	数据输出	48
3.2	选择结构程序设计	55
3.2.1	条件语句	55
3.2.2	开关选择语句	61
3.2.3	goto 语句	64
3.3	循环结构程序设计	66
3.3.1	3 种循环结构的流程表示及执行过程	66
3.3.2	循环的嵌套结构	69
3.3.3	循环程序举例	71
3.3.4	循环辅助控制	79
	本章小结	82
	习题	82
第4章	函数	85
4.1	C 语言程序的结构	85

4.2	函数的定义和调用	85
4.2.1	函数的定义	86
4.2.2	函数的调用	87
4.3	变量的存储类型和使用范围	90
4.3.1	变量的存储类型	90
4.3.2	变量的作用范围	92
4.4	函数间的传值	93
4.5	函数的嵌套调用	98
4.6	函数的递归调用	98
4.6.1	递归函数及其执行特点	98
4.6.2	实现递归调用的两种结构	100
4.6.3	递归结构的函数用法举例	100
4.7	函数的应用举例	102
	本章小结	106
	习题	106
第5章	编译预处理	108
5.1	宏定义	108
5.1.1	不带参数的宏定义	108
5.1.2	带参数的宏定义	110
5.2	文件包含	111
5.3	条件编译	114
	本章小结	116
	习题	116
第6章	复合数据类型	118
6.1	指针类型	118
6.1.1	指针的声明	118
6.1.2	指针的引用	119
6.1.3	指针的运算	119
6.2	数组类型	121
6.2.1	一维数组的声明	121
6.2.2	一维数组元素的引用与初始化	122
6.2.3	数组作为函数的参数	123
6.2.4	二维数组的声明与使用	125
6.2.5	二维数组元素的引用与初始化	126
6.2.6	指针与数组	127
6.2.7	指针数组与数组指针	128

6.3	字符串	131
6.3.1	字符串常量与变量	132
6.3.2	字符串数组	133
6.3.3	有关字符串操作的系统库函数	135
6.3.4	字符串的用法举例	137
6.4	结构类型	140
6.4.1	结构类型	140
6.4.2	访问结构体成员	142
6.4.3	结构指针和结构数组	144
6.4.4	结构指针作为函数的参数	149
6.4.5	结构体类型及结构指针的应用举例	150
6.5	关于指针的另外几种用法	161
6.5.1	指向函数的指针	161
6.5.2	指针类型的函数	164
6.5.3	指向指针的指针(二级指针)	166
6.6	main()函数的参数	167
6.7	联合、枚举和类型别名	171
6.7.1	联合类型	171
6.7.2	枚举类型	174
6.7.3	类型别名	175
	本章小结	176
	习题	176
第7章	文件及其操作	181
7.1	C语言文件的概念	181
7.1.1	ANSI C的I/O和UNIX C的I/O	181
7.1.2	流和文件	182
7.1.3	C语言文件	184
7.2	文件系统的基础	185
7.2.1	定义文件结构体类型的指针	185
7.2.2	用于文件操作的函数	186
7.3	建立文件的基本步骤	202
7.4	读取文件的基本步骤	204
7.5	非缓冲文件系统	205
7.5.1	open()函数	206
7.5.2	close()函数	206
7.5.3	creat()函数	207
7.5.4	read()和 write()函数	208

7.5.5	unlink()函数	209
7.5.6	lseek()函数	209
7.5.7	缓冲文件系统和非缓冲文件系统的区别	211
本章小结		211
习题		211
第8章 基本应用综合举例		213
8.1	数组元素的查找和排序	213
8.1.1	排序及应用	213
8.1.2	查找	219
8.2	数值积分	222
8.2.1	矩形法	222
8.2.2	梯形法	223
8.2.3	辛普生法	224
8.3	非线性方程的求根	226
8.3.1	牛顿迭代法	227
8.3.2	二分法	228
8.3.3	弦截法	229
8.4	C语言动态数据结构及其用法	229
8.4.1	动态存储分配	230
8.4.2	线性动态数据结构及链表的应用	231
8.4.3	非线性动态数据结构及二叉树的应用	237
8.5	文件的应用	240
8.5.1	用户数据文件的应用	241
8.5.2	用户文本文件的应用	242
习题		250
第9章 C++程序设计基础		251
9.1	从C到C++	251
9.2	面向对象程序设计	252
9.2.1	面向对象的概念及程序结构	252
9.2.2	面向对象程序设计的实现	253
9.3	C++程序开发过程	254
9.4	C++程序示例	255
9.5	C++对函数功能的增强	257
9.5.1	C++用函数组织程序	257
9.5.2	函数驱动C++程序	257
9.6	C++对数据类型的扩充	259

9.6.1 基本数据类型	260
9.6.2 自定义数据类型	260
9.6.3 从结构到类	260
9.6.4 类与结构的区别	261
9.7 C++对运算符的扩充	262
9.7.1 基本运算符	262
9.7.2 运算符重载	263
9.7.3 运算符作为成员函数	265
9.7.4 转换运算符	268
9.7.5 赋值运算符	270
本章小结	273
习题	274
第 10 章 C++语言基础	275
10.1 C++的类	275
10.2 C++的输入与输出	277
10.3 类与对象	278
10.3.1 类的定义与对象的引用	278
10.3.2 构造函数与析构函数	281
10.3.3 函数重载	286
10.3.4 友元	287
10.4 对象指针	291
10.5 派生类与继承类	298
10.5.1 派生类及其定义	298
10.5.2 单继承的派生类	299
10.5.3 多继承的派生类	307
10.6 虚拟函数与多态性	312
10.6.1 虚拟函数使用方法	313
10.6.2 虚拟函数与重载函数	314
10.6.3 虚拟函数的继承	315
本章小结	316
习题	317
参考文献	321

第 1 章 C 语言概述

C 语言是由 Dennis Ritchie 创造并首先在配备 UNIX 操作系统的 DEC PDP-11 计算机上实现的。C 语言是早期计算机语言 BCPL(目前在欧洲还有应用)发展过程中的产物。BCPL 由 Martin Richards 开发,并影响了 Ken Thompson 首创的 B 语言,B 语言促进了 C 语言在 20 世纪 70 年代的发展。本章主要介绍 C 语言的概况和起源、C 语言的程序构成原则和应用。

1.1 C 语言的起源

多年来,UNIX 系统配备的 C 语言一直是 C 语言的公认标准,由 Brian Kernighan 和 Dennis Ritchie 在《C 语言程序设计》(The C Programming Language,Prentice-Hall,1987)一书中描述。随着微型计算机的普及,出现了大批 C 语言系统,幸运的是,其中多数系统接受的源程序都能高度兼容。然而,没有统一标准必然存在差异,为了克服这种不利局面,ANSI (American National Standards Institute,美国国家标准学会)于 1983 年初夏成立了一个委员会,由该委员会制定了一个定义 C 语言的标准。本书写作之际,ANSI 已经停止正式接受修改建议,所有主要 C 语言编译程序都实现了 ANSI C 标准。

1.2 C 语言是中级语言

C 语言通常被称为中级计算机语言。中级语言并不意味着功能差或难以使用,或者比 BASIC、PASCAL 等高级语言原始,也不表明与汇编语言相似,会给使用者带来类似的麻烦。C 语言之所以被称为中级语言,是因为它把高级语言的成分同汇编语言的功能结合起来。

作为中级语言,C 语言允许对位、字节和地址这些计算机功能中的基本成分进行操作,而且 C 语言具有良好的可移植性。可移植性是指能够把为某种计算机写的软件通过改编应用于另一种计算机上。例如,如果为苹果机写的一个程序,能够方便地改为可以在 IBM PC 上运行的程序,则称之为可移植的。

所有的高级语言都支持数据类型概念,一种数据类型定义了该类变量的取值范围和在其上操作的一组运算。常见的数据类型有整数类型、实数类型和字符类型。虽然 C 语言有 5 种固有的基本数据类型,但与 PASCAL 或 ADA 相比,却不是强类型语言。C 语言几乎允许所有类型转换。例如,字符类型和整数类型数据能够自由地混合在大多数表达式中,C 语言不进行数组边界和函数参数的相容性等运行时的错误检查,检查运行时的错误是程序员的责任。

同样地,C 语言的早期版本也不检查函数参数与调用变量之间的兼容性。例如,在早期

版本中,用指针变量调用定义成期望浮点数变量的函数时,编译程序不产生出错信息。ANSI C 标准引入了函数原型(function prototype)的概念,能够报告与上例类似的潜在错误,函数原型将在本书的第 4 章进行具体讨论。

C 语言的特色:允许对位、字节、字和指针进行直接操作,因此,非常适合用于实现上述操作的系统程序的设计。C 语言的另一个重要特点是,仅有 32 个关键字(27 个来源于 Kernighan 和 Ritchie 的公认标准,另 5 个是 ANSI 增加的),这些关键字就是构成 C 语言的命令。和 IBM PC 的 BASIC 相比,后者包含 159 个关键字。

1.3 C 语言是结构化语言

严格地说,C 语言并不完全是块结构语言(block-structured language),一般称为结构化语言(structured language)。C 语言与 ALGOL、Pascal 和 Modula-2 等结构化语言非常类似。从技术上看,一个块结构语言允许在过程(procedure)和函数(function)中定义其他过程和函数。这样,通过使用作用域规则(scope rules),扩展了全局(global)和局部(local)的概念。作用域规则确定变量、过程、函数的可见性(visibility)。因为 C 语言不允许在函数内再创建函数,所以不能正式称为块结构语言。

结构化语言的显著特征是代码和数据的封装(compartmentalization)。这种语言能够把执行某个特殊任务所需的指令和数据从程序的其余部分分离出去并隐藏起来。获得隔离的方法通常是调用使用局部(临时)变量的子程序。通过使用局部变量,用户能够编制对程序其他部分没有副作用的子程序,容易编写共享代码段的程序。如果开发了一些分离良好的函数,在引用时仅需要知道函数做什么,不必知道它如何做。切记,过度使用全局变量(可以被全部程序访问的变量)会由于意外的副作用而在程序中引入错误,设计过 BASIC 语言程序的人对这个问题都会有深刻体会。

结构化语言提供了大量程序设计功能,直接支持若干循环结构,如 while、do-while 和 for 等。在结构化语言中禁止或不提倡使用 goto 语句,不能像 Basic 语言和 FORTRAN 语言那样把它作为常用的程序流程控制语句。结构化语言允许将语句缩进,但又不要求遵守 FORTRAN 语言那种严格的程序格式。

表 1-1 所示是结构化语言和非结构化语言的例子。

表 1-1 结构化语言和非结构化语言

非结构化语言	结构化语言
FORTRAN 语言	Pascal 语言
Basic 语言	ADA 语言
COBOL 语言	C 语言

现代语言都是结构化的,而陈旧的计算机语言的标志之一就是非结构性。结构化语言比非结构化语言更便于程序设计,用结构化语言编写的结构清晰的程序更易于维护。这已是人们普遍接受的观点。

C 语言的主要结构成分是函数,在 C 语言中,函数是一种构件(程序块),程序的所有操

作都在其中发生。函数允许一个程序中的任务被分别定义和编码,使程序模块化。可以确信,设计好的函数能在各种情况下正确工作,不会对程序的其他部分产生副作用。能否设计出独立的函数在大型项目中是至关重要的,在这种项目中,一个程序员的代码不会意外地影响其他人的程序。

在C语言中,实现结构化和代码隔离的另一种方法是使用复合语句。一个复合语句是作为一个语句处理的并在逻辑上相互关联的一组语句。在C语言中,复合语句就是处于一对大括号之间的语句序列。例如:

```
if(x<10)
{
    printf("too low,try again.\n");
    scanf("%d",&x);
}
```

在上面的语句中,如果x小于10,位于if语句之后、在一对大括号之间的两个语句将都被执行。这两个句子连同大括号表示一个代码块,是一个逻辑单位,其中所有语句必须一起执行。代码块不仅能使许多算法得到清晰且有效的实现,而且有助于程序员抓住程序的本质。

1.4 C语言是面向程序员的语言

令人遗憾的是,计算机程序设计语言并非都是为程序员设计的。以典型的非程序员语言COBOL语言和Basic语言为例。COBOL语言的设计目标不是改善程序员的环境,不是增加代码的可靠性,也不是提高编程的速度。它的部分目标是使非程序员能够阅读并理解程序。Basic语言主要是供非程序员编制计算机程序、解决简单问题的。

与此形成鲜明对比的是C语言。C语言是由实际工作中的程序设计人员创造、影响并测试的,最终向程序员提供了他们期望的一切,即很少限制、很少强求、块结构、独立函数和简明扼要的关键字集合。通过C语言编写的程序可以基本达到汇编代码的效率,程序又有ALGOL语言和Modula-2语言的块结构。因此,C语言自然深受一流专业程序设计人员的欢迎。

程序设计人员广泛使用C语言的一个主要因素是,常能用C语言代替汇编语言。汇编语言使用的是汇编指令,是能够直接在计算机上执行的二进制代码的符号表示,它的每个操作都映射为计算机执行的单一任务。虽然汇编语言使程序员达到最大灵活性和发挥最高效率的潜力,但开发和调试汇编语言程序的困难也是程序员难以忍受的。进一步说,由于汇编语言是非结构化的,最后完成的程序肯定像一团面条——纠缠不清的跳转、调用和变址,这种非结构化使汇编语言难于阅读、改进和维护。更重要的是,汇编语言编写的程序不能在使用不同中央处理器(CPU)的机器之间移植。

最初,C语言主要用于系统程序设计(system programming),系统程序是计算机操作系统及操作系统支持的实用程序的一部分。以下列出几种常用的系统程序。

- 操作系统
- 翻译程序
- 编辑程序
- 汇编程序
- 编译程序
- 数据库管理程序

随着 C 语言的普及和发展,更由于它的可移植性和高效率,许多程序员开始用 C 语言设计各类程序。几乎所有计算机上都有 C 语言编译程序,使用户可以很少改动,甚至不加改动地将为一种机器写的 C 语言源程序在另一种机器上编译执行。可移植性节省了大量的时间和财力,而且各种 C 语言编译程序均可产生非常紧凑、执行快捷的目标码,比任何一种 Basic 语言编译程序的目标码都紧凑和快速。

也许 C 语言被用于各类程序设计的主要原因在于程序员的偏爱,C 语言提供了汇编语言的速度和 FORTRAN 的可扩充性,而很少有 Pascal 和 Modula-2 的限制。C 程序员可以创建并维护一个专门的函数库,库中的函数根据个人需要配置,可以在各种程序中使用。由于允许(更确切地说是鼓励)分别编译,C 语言可以使程序员方便地管理大型项目,最大限度地减少重复劳动。

1.5 编译和解释

编译(compile)和解释(interpret)是两种程序执行的方式。理论上,任何程序语言都能编译执行又能解释执行,但特定语言一般仅取两者之一。例如,Basic 语言一般是解释执行的,而 C 语言常为编译执行的(最近,C 语言解释程序作为调试手段也受到了欢迎)。程序的执行方式不由它的编程语言确定,解释程序和编译程序只是作用于源代码上的复杂程序。

解释程序一次读进一行源代码,然后执行一串由这行源代码确定的指令。编译程序一次读完全部程序,并把它转换成目标码(object code)。目标码是源代码转换成能够在计算机上直接运行的代码,常指二进制码或机器码。程序被编译之后,源代码对程序的执行就毫无意义了。

使用解释程序时,每次运行程序都调用解释程序。例如,每次运行 Basic 语言程序时必须首先运行 Basic 语言的解释程序,然后装入源程序,再键入 RUN,Basic 语言解释程序检查、纠正并运行该程序。这种缓慢的处理过程在每次运行程序时都重复一次。相反地,编译程序一次将整个源程序转换为可直接在计算机上运行的目标码,因为编译程序一次性地完成转换工作,所以每次直接运行程序时只要键入其名字即可。因此,编译是一次性开销,而解释则每次运行时都引起额外开销。

运行编译程序时需要花费额外时间,但这种开销容易被每次运行程序所节省的时间抵消。被编译的程序运行速度比被解释的程序快得多。只有一种例外,即程序很短(不足 50 行)并且没有循环语句。本书和编译手册中常见的两个术语是编译时(compile time)和运行时(run time),前者指发生在编译期间的事件,后者指发生在运行期间的事件。读者通常只在讨论错误时(如在“编译错”和“运行错”这类短语中)看到这两个术语。

1.6 C 语言程序结构

C 语言的关键字都是小写的。在 C 语言中区分大小写,如 else 是关键字而 ELSE 则不是,在 C 语言程序中,关键字不能用于其他目的,既不能作为变量名也不能作为函数名。C 语言关键字如下。

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

以上 32 个关键字,与标准 C 语言句法结合,构成程序设计语言 C 语言。其中,27 个关键字由 C 语言的原始版定义,而 enum、const、signed、void 和 volatile 这 5 个关键字由 ANSI 委员会定义。

C 语言程序都由一个或多个函数组成,其中必须出现的函数是 main(),通常把它称为 C 语言的主函数。它在程序开始运行时首先被调入执行。在一个良好的 C 语言程序中,main()总是包括程序操作的基本轮廓,它可以由一系列函数调用组成。main()被看作 C 语言的基本组成部分,虽然技术上并非如此。另外不要把 main 作为变量名字,因为可能破坏编译程序的正常操作。

下面举 3 个 C 语言程序的实例,以便了解并认识 C 语言程序的结构。

【例 1.1】 一个简单的 C 语言程序。

```
/* C program _1 */
main()
{
    printf("C program _1\n");
}
```

这是一个简单而完整的 C 程序。第一行“/*”开头并以“*/”结尾的内容是 C 语言程序的注释信息。在程序的任何地方都可以加上注释的内容对程序的功能进行注解和说明。在 QC(QUICK C)中表示注释有两种方法,一种是用“/*”和“*/”将注释的信息括起来;另一种是在注释信息行的前面加“//”,如【例 1.3】中的表示。可以看出,【例 1.1】这个程序只有一个 main()函数,函数中只有一条“printf()”函数调用语句,它的功能是输出引号内除“\n”以外的所有字符。“\n”是换行符,作用是在输出“C program _1”这串字符后进行回车换行。

【例 1.2】 计算半径为 10 的圆的面积的 C 语言程序。

```
/* C program _2 */
#define PI 3.14159
main()
{
    float r,area;
    r=10;
```

```

    area=PI * r * r;
    printf("r= %f,area= %f\n",r,area);
    getch();
}

```

请留意圆周率 π 的表示方法。

【例 1.3】 含有两个程序块的 C 语言程序结构。

```

//C program _3
//main function
#define N 10                                /* C 语言的宏定义 */
main()                                       /* 主函数块 */
{
    int i,p;
    p=0;
    for(i=1;i<=N;i++)
        p=p+f(i);                          /* 调用 f(int x)函数 */
    printf("p=1+(1+2)+(1+2+3)+.....= %d\n",p);
}
//f sub function                           /* f(int x)子函数块 */
int f(int x)
{
    int m,y;
    for(y=0,m=1;m<=x;m++)                  /* 功能:循环求和 */
        y=y+m;
    return y;                               /* 返回函数值 */
}

```

该程序由两个程序块组成,完成 $\sum_{i=1}^n (\sum_{k=1}^i k)$ 的计算。

观察这 3 个程序,可以看出 C 语言程序有以下特点。

(1)C 语言程序的关键字和变量习惯上使用小写英文字母。而宏定义的宏名通常用大写字母。

(2)C 语言程序是由一个个语句组成的,一行可以书写多个语句,每个语句均由分号“;”结尾,因此,分号是 C 语言语句的终止符或分隔符。

(3)C 语言程序用大括号“{ }”表示程序的范围和层次关系。每个函数块需要用一对大括号括起来,同样,一段复合语句也需要用大括号括起来。

(4)程序中的注解说明信息可用“/* */”或“//”进行注释。

1.6.1 库和链接

从技术上讲,程序员可以仅用自己编制的函数构成实用的完整程序,但在实际中,这种情况十分罕见。在 C 语言的实际定义中,未提供任何实施输入/输出(I/O)操作的方法,因

此,大多数 C 语言程序都调用 C 语言标准库中的各种函数。

C 语言编译程序都带有标准函数库,由此完成最常见的任务。ANSI C 语言标准定义了标准库中必须包括的函数的最小集合。读者的标准库中,可能包括 ANSI C 语言标准并未规定的图形函数等。

在 C 语言的某些实现中,函数库以单一的大文件形式出现,在 C 语言的另一一些实现中,函数库以若干小文件的形式出现,提高了效率和实用性。

C 语言编译程序的实现者已经编写了大部分常见的通用函数。当调用一个别人编写的函数时,编译程序“记忆”它的名字。随后,链接程序(linker)把用户编写的程序同标准函数库中找到的目标码结合起来,这个过程称为链接(link),某些 C 编译程序带有自己的链接程序,有些则使用操作系统提供的标准链接程序。

保存在函数库中的是可重定位(relocatable)的函数,意味着其中机器码指令的内存地址并未绝对确定,只有偏移量是确定的。把程序与标准函数库中的函数相链接时,内存偏移量被用来产生实际地址。某些技术手册和参考书中更为详细地讲解了这一处理过程,但对于用 C 语言编程而言,读者并不需要进一步理解重定位的实际操作。

编程所需的许多函数都在标准库中,作为构成程序的基本构件。程序员自己编写并反复使用的函数,也可以放在一个库中。有些编译程序允许用户向标准库中插入自己的函数,有些则要求程序员另建一个库,无论如何,库中的代码都可以反复使用。

如前述,ANSI C 语言标准只定义了一个最小标准库。许多编译程序提供的库都比 ANSI C 语言定义的标准库更丰富,由于冗余,ANSI C 语言没有定义原始 UNIX C 语言的某些函数。

1.6.2 分别编译

多数短程序都可以完全放进一个源文件,然而,随着程序长度的增加,编译时间也大幅度增加(编译时间的增加常使人无法忍耐),因此,C 语言允许将一个程序分解为若干块,装入若干文件,每一个文件可单独编译。一旦所有的文件编译完毕,就可以将它们与函数库中的函数链接,形成完整的目标码。分别编译的优点是,当一个文件中的代码改变以后,不必重新编译全部程序。除最简单的项目之外,这种办法能节省大量时间。

1.6.3 编译 C 语言程序

编译 C 语言程序包括以下 3 步。

- (1)程序设计。
- (2)程序编译。
- (3)程序与所需要的库函数链接。

某些编译程序提供一个集成开发环境,其中包括编辑程序。另一些编译程序没有集成开发环境,程序员必须用独立的编辑程序编程。编译程序只接受输入的标准文本文件。用某些字处理软件产生的文件包含控制码和非打印字符,编译程序将拒绝接受。

读者编译自己编写的源程序时,所用方法依实际使用的编译程序而定。同样,链接的实现也随编译程序和系统环境有所不同,详情见用户手册。

1.6.4 C 语言的内存映象

一个已完成编译的 C 语言程序取得并使用 4 块在逻辑上不同且用于不同目的的内存区域,如图 1-1 所示。第一块区域含有程序代码,相邻的一块内存区域存放全局变量,其他两块分别是栈(stack)和堆(heap)。栈用来处理程序运行的许多事务,保存函数调用时的返回地址、函数的变量、局部变量以及 CPU 的当前状态。堆是一个自由内存区域,程序可利用 C 语言的动态分配函数,取得用于链接表和树等结构所需要的内存。

程序的实际布局随编译程序和环境的不同而有所变化。由于 8086 体系的分段内存结构,多数适用于 8086 系列处理器的编译程序都有 6 种组织内存的方法。虽然 4 块区域的实际物理布局随 CPU 类型和编译程序的实现不同而有所不同,但图 1-1 仍然从概念上描述了 C 语言程序在内存中的形态。



图 1-1 一个 C 语言程序在内存中的映像

1.7 术 语

1. 源代码(source code)

用户可阅读的程序文本,通常称为程序,源代码是 C 语言编译程序的输入。

2. 目标码(object code)

由源代码转换而来的机器码,计算机可直接读入并执行。目标码是链接程序的输入。

3. 链接程序(linker)

可把分别编译的函数链接为完整程序的程序,能够把 C 语言的标准库函数与程序人员编写的代码结合起来,链接程序的输出是可执行代码。

4. 函数库(library)

存放可被用户程序引用的标准函数。这些函数包括所有 I/O 操作及其他有用的子程序。

5. 编译时(compile time)

程序被编译时出现的事件。在编译期间出现的错误通常是语法错误。

6. 运行时(run time)

程序在运行时出现的事件。

习 题

1. 简述你对计算机语言和程序设计的认识。
2. C语言为什么受到程序员的欢迎？它有哪些突出优点？
3. 结合自己使用的计算机系统环境和C语言编译系统，说明C语言程序的上机调试过程。
4. 能将高级语言源程序转换成目标程序的是()。
A. 调试程序 B. 解释程序 C. 编译程序 D. 编辑程序

第 2 章 数据类型和表达式

数据是程序处理的对象,程序的主要任务就是对数据进行加工和处理。数据类型是按性质、表示形式、占用存储空间的大小和构造特点划分的。C 语言程序中使用的每一个数据都属于某一种类型,必须正确声明和使用。C 语言提供了许多运算符,可以对不同类型的数据进行处理,运算符和数据结合后便形成了表达式。本章最后还介绍了位运算、数组和指针的基本知识及简单用法。

2.1 数据类型

在 C 语言中,对于一个给定的字节 01000011,对应的八进制、十六进制与十进制表示分别是 0103、0x43 和 67。这个字节表示的到底是什么数据呢?这可能有多种解释。如果将它理解为整数,则这个字节表示的是正整数+67;如果将它理解为一个 ASCII 编码的字符,这个字节表示的又是大写英文字母 C,还可能与相邻字节结合为其他类型的数据。由此可见,要理解一个数据到底表示的是什么含义还必须依赖于这个数据应作何解释,这种对数据的解释称之为数据类型。

一般来说,每一种程序设计语言都支持一些数据类型,但并不是所有程序设计语言都严格地约束数据类型及其上的操作。如果编译程序在语法分析阶段就能够严格检查数据上的操作是否满足数据所属类型的要求,这种程序设计语言就称为强类型的(strong typing),否则称为弱类型的(weak typing)。Ada 和 Pascal 等语言都是强类型语言,而 SmallTalk、Basic 等语言则是弱类型语言。C 语言的类型检查机制处于两者之间。

C 语言提供了 5 种基本数据类型:字符型、整型、浮点型、双精度型和空值,此外还包括 4 种复合数据类型:指针类型、数组类型、函数类型和构造类型。

在程序设计中,数据类型有着重要的意义,程序中所有的数据都属于特定的类型,数据的表示方式、取值范围以及对数据可以使用的操作都由数据所属的类型决定。

类型可以帮助编译程序生成高效率的目标代码。为了让编译程序了解数据所属的类型,通常程序设计语言都遵循“先声明、后使用”的原则,即在使用一个数据之前必须先声明它属于哪种数据类型。这样,编译程序在生成目标代码时可以知道需要分配多大的存储空间以及如何引用这个数据。

数据类型对于编写程序的程序员也有重要意义。由于程序中的每一个数据都明确地属于一种类型,所以这些数据的取值范围、可以操作的运算等信息就可通过数据所属的类型显式地反映出来,提高了程序的可读性和可理解性。在程序中使用数据时,会根据数据的特点(如是字符还是数值、有无小数等)与可能的数值范围决定数据的类型。一个数据被明确属于某一特定类型后,在该数据上允许操作的运算也就确定了下来。例如,两个整数可以进行加、减、乘、除、整除、取模等运算,而两个字符串则可进行比较、连接、判断子串等操作,但不

可作四则运算。如果程序员在编写程序过程中错误地将一个浮点数作取模运算或移位运算,这种错误可以由编译程序在语法分析阶段检测出来,并且帮助程序员准确地定位错误。在赋值语句中将涉及类型相容与类型转换问题。

2.2 保留字和标识符

组成 C 语言程序的最小单位是单词(token),最小单位意味着单词是语言中具有独立意义、不可拆分的最基本单位。C 语言程序中有 5 种单词:保留字、标识符、常量、运算符和分隔符。

2.2.1 标识符

现实世界中的所有事物都有自己的名字,从而可与其他事物区别开来。在程序设计语言中也采用类似的方法,允许程序员对程序中的各个元素加以命名,这种命名记号称为标识符(identifier)。在 C 语言中,标识符是由字母或下划线开头,后继字符可以是字母、数字与下划线组成的字符串。

标识符的定义如下。

- 字母开头的字母、数字、下划线串。
- 下划线开头的字母、数字、下划线串。

例如,ab、sum1、_12x、y_1、t_12 等都是 C 语言允许的标识符。

标识符可以为任意长度,但一般只识别前 31 个字符,而在外部连接时,根据不同的语言环境和编译器,用来区别不同标识符的有效长度可能还要短些。在 MS C 中,外部连接的标识符长度也是 31 个字符。

在 C 语言中,标识符对字母的大小写是敏感的。因此,count、COUNT 和 Count 被认为是 3 个不同的标识符。在程序中恰当运用下划线、大小写字母和数字组成一个有意义的标识符字符串是一个良好的习惯。

2.2.2 保留字

在 C 语言中,有一些单词具有专门的意义和作用,不能当一般的标识符使用,这些单词称为保留字(reserved word),或者称为关键字(key word)。在第 1 章中已经列出了 C 语言标准 ANSI 规定的 32 个保留字,使用中用户定义的标识符不能与保留字同名。

另外,除了第 1 章中列出的保留字外,C 语言编译器还增加了几个保留字,以使用户能更好地利用 Intel 8086 处理器的存储结构和中断处理,进行混合语言编程。由于没有 ANSI C 语言的支持,这部分保留字在不同的编译器中存在一些差别,下面列出了 MS C 扩充的保留字。

_asm _based _cdecl _export _far _fastcall _fortran _huge

_interrupt _loadds _near _pascal _saveregs _segment _segname _self

此外,main、argc、argv、envp、_emit、_setargv、_setenvp 也是 C 语言的保留字。所有的

C 语言程序都包含一个或多个函数,而编译成可执行程序的唯一不可缺少的函数名为 `main`,它在程序开始执行时第一个被调用。所有的保留字都是由小写字母构成的,而第一个字母若为下划线,则表示为 MS C 的私有扩充,其他编译器可能不支持略有不同的名字。

2.2.3 选择合适的标识符

标识符主要用于标记程序中的一些元素,而这种标记合适与否也相当重要。有人以为在一个简短的程序中,采用短的标识符(如 `a`、`b`、`c` 等)不仅容易记忆,而且容易书写。但当人们要对几周前的程序进行修改时,可能会发现原来简短而方便的程序现在是那么难以理解。此外,即使能够理解自己编写的程序,其他人也会很难理解,导致无法与其他人合作编写更大规模的程序。

正因为如此,所有介绍程序设计的书都主张采用有意义的标识符,如 `count`、`withdraw`、`deposit` 等。对于两个以上英文单词组成的标识符目前有两种流行的写法:一种是每个英文单词的首字母大写,其他字母用小写,如 `IsEmpty`、`SoundOfMusic`、`OpenWindow` 等;另一种是各个英文单词之间以下划线连接,如 `is_empty`、`sound_of_music`、`open_window` 等。无论选用哪种写法,和谐与统一的形式是必要的,不要在程序的某部分采用一种写法,而在另一部分采用另外的写法。

当标识符过长时,应该使用缩写的形式。为提高缩写标识符的可读性,缩写时可参考以下几个原则。

- (1) 英文单词的第一个字母一定要保留。
- (2) 英文单词的首部比尾部重要。
- (3) 英文单词中的辅音字母比元音字母重要。
- (4) 遵守一些英文单词的缩写习惯。
- (5) 缩写后连续单词的一些下划线可去掉。

例如,`open_window` 可缩写为 `openwin`,`account_record` 可缩写为 `acc_rec`,`double_sides` 可缩写为 `dblside`。多阅读一些风格较好的程序可以积累缩写的经验。用户定义的标识符不能与 C 语言的保留字同名,同时还要避免与 C 语言的标准库产生名字冲突。

2.2.4 常量与变量

C 语言中的数据可分为常量与变量两大类。在程序的执行过程中,值不能被改变的量称为常量(`constant`),值可以改变的量称为变量(`variable`)。无论是常量还是变量,都有其类型,并且根据所属类型规定其可进行的运算。

1. 常量和符号常量

常量的类型是根据数据的书写形式(即语法形式)决定的。例如,800、1 500 是整数类型,800.00、3.141 59 是浮点类型,‘a’、‘b’是字符类型。

有一种特殊的常量是用标识符表示的,称为符号常量。符号常量常用于帮助记忆,提高程序的可读性和可维护性。例如,程序中要用到圆周率时,可以将圆周率表示为一个符号常量 `pi` 并将其值设置为 3.14,程序中每次使用圆周率时都用 `pi` 表示,如果要提高精度,只要修改 `pi` 的初始值就可以了(如将 3.14 改成 3.141 592 6)。

2. 变量

变量是程序设计中的有力武器。从程序设计的角度看,变量由 4 部分组成:变量名、数据类型、内存地址以及变量的值。变量用标识符来命名,它对应着内存中的某个区域,该区域的位置由内存地址指明,区域中可存放数据。在程序中通过变量的名字来访问存放在这个存储区域中的数值。

在程序中使用变量时必须遵循“先声明、后使用”的原则。“声明”即是对程序中所用的变量进行类型的定义和说明。在 C 语言中,变量的类型必须由程序员进行显式声明,声明方式是类型后面列出变量的名字或逗号分开的变量表。

type 变量表;

例如:

```
int i,j,k;
```

声明了 3 个属于整数类型的变量 i、j、k。编译程序会根据变量的类型为其分配相应的存储空间,如给变量 i、j、k 各分配 2 个字节。

声明变量的意义,是向编译程序声明一个变量的名字及其类型,以便编译程序给这些变量分配相应的存储空间,并不真正地执行程序,这种语句称为声明语句或变量声明语句。

变量的值主要通过赋值(assignment)运算改变。赋值运算由赋值运算符“=”将赋值表达式分为左、右两边。左边一般是变量标识符,不能是常量、符号常量或某些表达式,称为左值;右边是一个可以具体计算出来的值,可以是常量、变量或由常量与变量经过若干运算后得出的值,统称为表达式,称为右值。赋值语句的作用就是将右边的值求出并放入左边变量名对应的存储区域中。例如,对于声明的变量 i,可以用赋值语句:

```
i=3*8;
```

将该变量的值设置为 24,即这个名字为 i 的变量对应的存储区域中存放了数值 24。

在代数中“ $x=x+1$ ”是不可解的、错误的,然而作为程序设计中的赋值运算“ $x=x+1$ ”则是完全合理的,它表示将变量 x 的值增加 1,即先取出 x 的值,然后与 1 相加,最后将相加结果再放回变量 x 相应的存储区域中。经过运算后,变量 x 的新值取代了原来的值。注意,变量名字 x 在上述赋值运算的左、右都出现了一次,名字 x 出现在“=”的右边是变量 x 的值的引用,而名字 x 出现在“=”的左边是对变量 x 的内存地址的引用。

2.3 基本数据类型

C 语言提供了 5 种基本数据类型:字符型、整型、浮点型、双精度型和空值类型。C 语言提供的基本数据类型用保留字作为类型的名字(char、int、float、double、void 等),除系统提供的这些类型名外,C 语言还允许用户自定义类型名,自己可用标识符作为类型的名字。

数据类型的存储方式和取值范围与计算机硬件结构密切相关(影响最大的主要是计算机的字长),这些因素可能会影响程序的可移植性。表 2-1 列出了 C 语言的基本数据类型的存储方式与取值范围。注意,表中实数(浮点类型与双精度类型)的取值范围不是用最小值与最大值表示的,而是可以表示的最小精度与最大数值。

表 2-1 C 语言基本数据类型的存储方式和取值范围

数据类型	类 型 名	占用字节	取值范围
字符型	char	1	-128~127
整型	int	2	-32 768~32 767
浮点型	float	4	3. 4E-38~3. 4E+38
双精度型	double	8	1. 7E-308~1. 7E+308
空值类型	void	0	无值

除空值类型外,其余 4 种基本类型前面还可加上类型修饰符,用于改变基本数据类型的含义,以满足更广泛的程序设计需要。C 语言的类型修饰符有两对,分别是 signed(有符号)和 unsigned(无符号)、short(短的)和 long(长的)。这两对修饰符可同时用于修饰基本数据类型,但同一对中只允许出现一个修饰符。

对基本数据类型的修饰不允许任意组合,表 2-2 列出了基本数据类型和类型修饰符的所有组合方式。使用表中类型的缩写名字与直接使用类型名字具有同样效果。

表 2-2 基本数据类型和类型修饰符的所有组合

类 型 名	简 写 名	字节	取值范围	有 效 位
char		1	-128~127	
signed char				
unsigned char		1	0~225	
int		2	-32 768~32 767	
signed int	signed			
short int	short			
signed short int	signed short			
unsigned short int	unsigned short	2	0~65 535	
unsigned int	unsigned			
long int	long	4	-2 147 483 648 ~2 147 483 647	
signed long int	signed long			
unsigned long int	unsigned long	4	0~4 294 967 295	
float		4	3. 4E-38~3. 4E+38	6~7 位
double		8	1. 7E-308~1. 7E+308	15~16 位
long double		10	3. 4E-4932~1. 1E+4932	19 位

如果不使用 signed 或 unsigned 修饰 int,编译程序默认是 signed 类型;而对 char 不加修饰,则有些编译默认为 signed,有些则认定为 unsigned。具体规定要参考编译程序提供的使用手册。

2.3.1 数据的内部表示

在 C 语言中,整型数据的内部表示一般采用补码形式,实数采用浮点小数表示。整数有

3 种不同的尺寸:8 位的字符整数、16 位的短整数和 32 位的长整数。作为有符号数时,每种整数的最高位为符号位,如图 2-1 所示;作为无符号数时,每种整数的最高位是该数据数值表示的一部分。

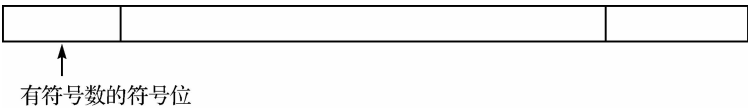


图 2-1 整数的内部表示

有 3 种不同尺寸的浮点型数据:4 字节浮点数、8 字节双精度数和 10 字节长双精度数。它们都使用 4 部分位组来确定所表示的数值,如图 2-2 所示。

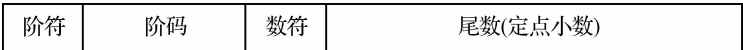


图 2-2 实数的内部表示

数符和阶符各占一位,阶码的位数决定了数据的精度范围,尾数的位数规定该数据有效位的长度(按 $2^{-1}2^{-2}\dots\dots2^{-n}$ 表示),如表 2-3 所示。

表 2-3 3 种实数数据类型的存储格式

数据类型	阶码长度	尾数长度	阶码偏移	整 数 位
float	7 位	23 位	7FH	无(1 为隐含的整数)
double	10 位	52 位	3FFH	无(1 为隐含的整数)
long double	14 位	63 位	3FFFH	1 个整数位(位 63)

在 16 位 PC 机中,多个字节构成的数据类型一般采用低字节(地址小)在前,高字节(地址大)在后的原则。

2.3.2 字符型

在 C 语言中,字符型的数据在内部存储的是该字符的 ASCII 码,即存储的是一个整型数值。

1. 字符常量

括在单引号中的字符称为字符常量,如'A'、'9'、' '(空格)、'_'(下划线)等。字母的大、小写看作是互异的,即'A'和'a'表示的是两个不同的字符常量。在计算机上编辑程序时,字符左边的单引号与右边的单引号是相同的,在键盘上均通过分号右边的键输入。

单引号本身也可以作为字符常量,那么诸如单引号字符这样的控制字符将如何表示呢?在 C 语言中要表示它们是用转义字符方式,即转变其他字符的含义来表示这些特殊字符。转义字符也由单引号括住,所有转义字符都以反斜线“\”开头。由于反斜线专门用作转义,所以反斜线符号本身也需要转义表示。表 2-4 列出了 C 语言提供的转义字符。例如,单引号字符就可以表示为'\',制表符表示为'\t',字符常量'A'就可以有两种等价的表示方法:'\101'和'\x41'。注意,'\101'和'\x41'都是单个字符,不要当作 4 个字符组成的字符串。

表 2-4 C 语言预定义的转义字符

转义字符	ASCII 码	含 义
\a	0x07	响铃符(BEL-Bell)
\n	0x0A	换行符(LF-LineFeed)
\t	0x09	水平制表符(HT-HorizontalTable)
\v	0x0B	垂直制表符(VT-VerticalTable)
\b	0x08	回退符(BS-Backspace)
\r	0x0D	回车符(CR-CarriageReturn)
\f	0x0C	换页符(FF-FormFeed)
\\	0x5C	反斜线("\")
\?	0x3F	问号("?")
\'	0X27	单引号("'")
\"	0X22	双引号(" ")
\ooo	0x00~0xff	八进制字符常数
\xhh	0x00~0xff	十六进制字符常数

字符常量的完整表示为:‘字符’。

其中,引号内的字符有两类:可显示字符和转义字符。

(1)可显示字符:指的是 ASCII 码表中除单引号、反斜线和换行以外的所有可见字符。

(2)转义字符:如表 2-4 所列举的。例如,‘a’、‘\n’、‘\\’、‘\t’、‘\234’、‘\x59’ 都是 C 语言允许的字符常量。

2. 字符变量

与其他所有变量一样,字符变量也是由标识符表示其名字的,在使用之前必须先声明该变量。字符变量的声明方式与前面介绍的变量类型的声明方式一样,是在字符类型保留字 char 后面列出表示名字的标识符,例如:

```
char choice;
```

这就是一个字符类型的声明语句,该语句声明了一个字符变量 choice(行末的分号“;”是 C 语言要求的,表示语句的结束)。这个变量的名字是 choice,编译程序会在内存中分配一个字节给这个变量,程序通过名字 choice 访问这个字节,但是在这个字节中存放的是什么值还未确定。通常在声明变量时还可为变量设置初始值,这是一种好的程序设计习惯。例如,可以在声明 choice 的同时设置 choice 的初始值为 Y(称为初始值是因为该值以后可以被修改)。

```
char choice=Y;
```

这里不但声明了名为 choice 的字符类型变量,还在与它相关联的字节中存放了字符常量 Y。由于 Y 还可用转义字符表示,所以以下 3 条声明语句都与上一条语句等价。

```
char choice='\131';           //Y 的 ASCII 码值八进制表示
```

```
char choice='\x59';           //Y 的 ASCII 码值十六进制表示
```

```
char choice=89;                //Y 的 ASCII 码值十进制表示
```

通过用这4种不同值对变量 choice 进行初始化,可以进一步了解和掌握字符常量的表示方法。

也可以一次声明多个变量,例如:

```
unsigned char sel1,sel2;
```

它声明了两个无符号字符变量 sel1 和 sel2。

注意,C语言的字符类型变量不仅可用于存储字符的 ASCII 编码,还可存储某一范围内的整数值。如上面的第4种表示,就是给变量 choice 赋值一个十进制的整数89,整数的范围在-128~127之间;同理变量 sel1 或 sel2 的无符号字符变量还可存储0~255之间的整数值。

【例 2.1】 字符变量声明和赋值举例。

```
/* character example_1 */
main()
{
    char c1=89,c2,c3,t='\t';
    c2='\130';
    c3='\x5a';
    printf(" %o%c%x%c%d%c%c\n",c1,t,c1,t,c1,t,c1);
    printf(" %o\t%x\t%d\t%c\n",c2,c2,c2,c2);
    printf(" %o\t%x\t%d\t%c\n",c3,c3,c3,c3);
}
```

这里请大家写出程序运行后的输出结果,特别注意第一条打印语句中变量 t 的输出结果。

2.3.3 整型

整数类型又简称为整型,是表示整数的集合。注意整数类型与数学中的整数概念有所不同,它表示的只是整数的一个有限的子集,因为计算机只能表示一定范围内的整数。虽然在数学的意义上整数可以作为实数的一个子集,但是将整数类型独立出来更符合日常的习惯,也便于编译程序生成高效率的目标代码。

1. 整数常量

C语言的整数常量支持3种数制:十进制、八进制和十六进制。**【例 2.2】**给出了相同整数数值的3种表示。在C语言中,123与0123表示的是不同的数值。

【例 2.2】 同一整数数值的3种表示方法。

十进制	八进制	十六进制
0	0	0x0
7	07	0x7
9	011	0x09
83	0123	0x53
123	0173	0x7B

整数常量的类型由整数值所属的范围决定,C 语言编译程序会将其类型解释为能存储该数的最小类型。例如,123 是有符号整数类型,65 000 可解释为无符号整数类型,而 5 500 000 则应解释为有符号长整数类型。程序员也可在整数常量后面加上后缀,以显式指明该常量是何种类型的整数,如 123L 是长整数类型、123U 是无符号整数类型等。

定义 C 语言的整数类型常量如下。

十进制常量[整型后缀] 八进制常量[整型后缀] 十六进制常量[整型后缀]

其中,整型后缀有如下两种。

无符号后缀 u|U //u|U 表示字母 U 的小写或大写

长后缀 l|L //l|L 表示字母 L 的小写或大写

例如,023、234l、33u、0x24、-25、067、0xa9、10L 都是 C 语言允许的整型常量。

2. 整数变量

声明整数变量用保留字 int,可以加入类型修饰符。例如:

```
int age,work-years; //声明了两个整型变量 age 和 work-years
```

```
long int population; //声明了一个长整型变量 population
```

```
short days=7; //声明了一个短整型变量 days 并设置初始值为 7
```

根据表 2-2,编译程序会给名字为 age、work_years 和 days 的整型变量分别分配 2 个字节,而为变量 population 分配 4 个字节。最后一个变量的初始值 7 就存放为 days 分配的那 2 个字节中,允许程序执行时修改这个值,修改时是通过变量名字 days 来访问的。下面给出一个对整型变量进行声明、赋值和输出的例子。

【例 2.3】 整型变量的定义和赋值举例。

```
// integer number example
main()
{
    int x,y,z,t;
    x=32768u;
    y=0111;
    z=0x23;
    t='\t';
    printf(" %o%c%x%c%u%c%d\n",x,t,x,t,x,t,x);
    printf(" %o%c%x%c%u%c%d\n",y,t,y,t,y,t,y);
    printf(" %o%c%x%c%u%c%d\n",z,t,z,t,z,t,z);
}
```

请分析程序的输出结果。

2.3.4 浮点类型和双精度类型

与整数类型相似,浮点类型与双精度类型也只是实数的一个子集,取值范围取决于计算机型号。计算机表示的实数是一个近似值,这对程序设计会产生影响。例如,应避免直接比较两个浮点数是否相等,10.0×0.1 不一定等于 1.0。

1. 浮点常量

C语言允许程序员按日常使用小数点的习惯书写实数,并且小数点前或后只要有一个数字即可。这些实数只能是十进制数。例如:

2.0 3.14159 0.025 +33.3 -4.275 2. .025 -.5

这些实数均是浮点常量。

对于一些特别大的数,如天文数字或特别小的,如原子、电子中的微量,如果采用上述记数法则不便书写与阅读。科学家们一般都把这些数据写成一个实数乘以10的若干次幂的形式,这种记法称为科学记数法。例如,电子的静电质量可记为 9.10956×10^{-28} 。C语言也支持程序员使用类似于科学记数法的指数形式表示浮点常量。例如:

9.10956E-28 18E10 2e-5 -27e4

这些均是C语言中的浮点常量。注意,两种写法在浮点常量中间都不允许插入空格,否则会产生语法错误。

注意,数3.14称为浮点常量,但其类型为双精度类型,占8个字节。这样规定是为了在运算时获得更大的精度,在本章最后学习了C语言的类型转换规则后就可理解这点。也可将浮点常量加后缀,强制某一浮点常量为某一类型。例如,数3.14L显式指明该常量是长双精度类型,占10个字节;数3.14F是浮点类型,占4个字节。

从表2-2可见,浮点类型数据的有效位为7位,双精度类型数据的有效位为15位,长双精度类型数据的有效位为19位。这些位数均指十进制数的位数。

定义C语言的浮点常量如下。

[符号] 小数常量 [指数部分] [浮点后缀]

[符号] 数字序列 [指数部分] [浮点后缀]

其中,小数常量:[数字序列]. 数字序列

指数部分:e [符号] 数字序列、E [符号] 数字序列

符号:+、-

浮点后缀:f、F、l、L

例如,2.3、-0.003、3e+5、.234e-2、3.14L、6.78f等都是C语言允许的浮点常量。

2. 浮点变量与双精度变量

声明浮点变量或双精度变量分别用保留字float和double,可以加入类型修饰符。例如:

```
float balance;           //声明了一个浮点变量 balance
double weight=0;         //声明了一个双精度变量 weight 并初始化为 0
long double distance;    //声明了一个长双精度变量 distance
```

根据表2-2,编译程序会为变量balance分配4个字节,为变量weight分配8个字节,为变量distance分配10个字节。

【例2.4】 实型变量的声明和打印举例。

```
//float number example
main()
{
```

```
float x,y,z,t;
x=1001;y=2.3456f;
z=88.9L;
printf(" %5.1f\n",x);
printf(" %5.1f\n",y);
printf(" %5.1f\n",z);
printf(" %d\n",sizeof 88.9L); //打印出常数 88.9L 所占字节数
printf(" %d\n",sizeof(2.3f)); //打印出常数 2.3f 所占字节数
printf(" %d\n",sizeof(y)); //打印出变量 y 的字节数
}
```

2.3.5 字符串常量

字符串常量是括在双引号内的字符序列,简称字符串。字符中左、右两边的双引号是相同的,在键盘上均用 Shift 键加单引号键输入。在字符串中可以包含空格、转义字符以及中文。例如:

```
"This is a string constant."
"His name is\"John\".\n"
"注意:\x07\n 张三正在开会.\n"
```

第 1 行的字符串是一个典型的字符串;第 2 行的字符串中 John 的名字要用双引号括住就必须采用双引号的转义字符,此外,该字符串的末尾接了一个换行符,在屏幕上显示该字符串时,会在显示完字符串后将光标移到下一行,在打印机上打印该字符串时,会在打印完该字符串后移到下一行;第 3 行的字符串中包括了中文且有一个响铃字符与两个换行字符,在屏幕上显示时会发出吡一声并分两行显示。由于双引号专用于括住字符串,所以在字符串内使用双引号必须使用转义字符。

要注意字符串中转义字符的用法,C 语言规定,在字符串中使用八进制或十六进制转义字符时,按最长有效长度对该转义字符进行解释。例如:字符串"Hi,\x7come on."中并不包含响铃字符,而解释为:

```
"Hi,|ome on."
```

因为其中字符“c”可以结合到十六进制转义字符序列中,形成一个比响铃更长的字符\x7c(即字符|)。

字符串中包括字符的个数称为字符串的长度(虽然转义字符用多个符号表示,但只当作一个字符计算)。字符串存储在内存中时,除字符串中的每一个字符外,还在最后接一个“\0”字符表示串的结束,所以长度为 n 的字符串需要 $(n+1)$ 个字节来存储。“A”与 A 的区别在于前者是字符串常量,占两个字节,后者是字符常量,占一个字节。

C 语言用于存放字符串的量是数组,所以在 C 语言中的字符数组与字符串是同等概念。有关字符数组的用法将在第 6 章进行详细介绍。

2.3.6 符号常量

前面已经讨论了符号常量的许多优点。为支持程序员使用符号常量,C 语言提供了存

取修饰符 `const`,用于声明变量时约束变量的存取方式。但 C 语言本身并没有提供统一的符号常量供程序员使用,程序员要自己动手定义符号常量。

在变量声明语句中,可以用保留字 `const` 规定一个变量的值在程序中不能再被改变,这种变量称为符号常量或 `const` 变量。由于符号常量的值不可进一步改变,所以在声明符号常量时就必须对符号常量进行初始化。例如:

```
const float pi=3.14159;  
const int model=486;
```

这里声明了两个符号常量 `pi` 和 `model`,它们在程序的作用域内始终代表了这两个初始化的值,若在程序其他地方改变符号常量 `pi` 或 `model` 的值(例如,对 `pi` 或 `model` 进行赋值),系统将会给出一个语法错误的提示。

对于所有基本数据类型的 `const` 变量,编译程序处理起来与常量非常相似。在 C 语言中,`const` 只约束了如何使用这些变量,并没有规定如何分配这些变量的存储空间。有些编译程序可以利用这一特点提高效率。由于编译程序已知常量值,所以没有必要再为它动态地分配存储空间;如果使用表达式初始化该变量,可在必要阶段完成该表达式的求值。

从符号常量可以看出 C 语言的一个特点,即虽然在初始化与赋值时均使用运算符“=”,但两者的含义是完全不同的。符号常量可以初始化,但不允许赋值。稍后将进一步看到初始化与赋值的区别。

2.4 运算符与表达式

基本数据类型不仅限定了常量与变量的存储方式、取值范围,同时还约束了在数据之间可进行的操作。对基本数据类型的常量与变量进行加工的过程称为运算,表示各种不同运算的符号称为运算符,参与运算的数据称为操作数。C 语言的基本运算包括算术运算、关系运算、逻辑运算、位运算、赋值运算和逗号运算等。按操作数的个数,运算符可分为一元运算符、二元运算符和三元运算符。

在众多程序设计语言中,C 语言的表达式十分复杂、灵活,C 程序的基本运算主要由表达式完成。因而,学习 C 语言的表达式需要循序渐进,应先对 C 语言的表达式有一个总的了解,然后通过学习各种运算符进一步掌握表达式的概念与用法。

2.4.1 表达式

表达式(expression)是由操作数和运算符按一定语法形式组成的符号序列。每个表达式经过运算后都会产生一个确定的值(这一运算过程又称为表达式求值),称为表达式的值。经过运算求出表达式的值是属于某种类型的,该类型称为表达式的类型。表达式的类型由运算以及参与运算的操作数的类型决定(空值类型的表达式例外)。

一个常量或变量是最简单的表达式,其值即该常量或变量的值;表达式的值还可以用作其他运算的操作数,形成更复杂的表达式;用圆括号括住的表达式还是一个表达式,其值和类型与未加括号时相同。例如:

```
3.14  
balance  
2 * r * pi  
a * (b + c) + d
```

这些均是 C 语言中合法的表达式。

2.4.2 算术运算符及算术表达式

算术运算符用于各类数值运算,由算术运算符和括号将常量、变量、函数等连接起来的式子称为算术表达式。

1. 算术运算符

C 语言的算术运算符有:“+”“-”“*”“/”“%”“++”“--”。

其中,一元算术运算符是“-”,其作用是取操作数的相反数。二元算术运算符包括“+”、“-”、“*”、“/”和“%”,这些运算与通常数学意义下的运算是相同的。“%”、“++”、“--”运算符只可用于整型,其他运算符可用于整型、浮点型和双精度型。

对于两个都是整型的操作数,“/”表示的是整除的概念,即结果舍弃小数部分,只取整数部分。例如,15/3、16/3 与 17/3 的求值结果均为 5,而 1/3 和 2/3 的求值结果均为 0。“%”表示取模的概念,即只取余数。例如,17%3 的求值结果为 2,28%5 的求值结果为 3。取模的结果不可能大于或等于第二个操作数。

“++”为加 1(或自增)运算符,“--”为减 1(或自减)运算符。用法可以是前置或后置。例如,a++(后置)或++a(前置),其功能是完成每次加 1 的计算。该运算符有以下两种使用情况。

(1)作为独立的表达式语句方式使用,如“i++;”和“++i”,这时的前置和后置效果相同。

(2)作为运算元素与其他元素一起参与运算,这时前置和后置对表达式所产生的结果不同,如下所示。

```
a=3;b=2;  
c=a++ * b;
```

以上语句是先计算 $c=a*b$,然后再计算 $a++$,执行完后, $a=4,c=6$ 。

```
c=++a * b;
```

该语句先计算 $++a$,然后再计算 $c=a*b$,执行完后, $a=4,c=8$ 。

2. 算术表达式

两个表达式通过加(+)、减(-)、乘(*)、除(/)、取模(%)等算术运算符作算术运算,组成一个新的表达式,称为算术表达式。参加算术表达式的操作数可以是字符型、整型、浮点型、双精度型等多种,算术表达式的求值结果也可以是这些类型。例如,下列表达式均是算术表达式。

```
x * (y - z)  
(a - 2) / (b + 2) + 50
```


3. 重要说明

在程序设计中要注意算术运算溢出问题。在 C 语言中,除以零以及浮点运算溢出均会产生一个运行时错误并导致程序终止,但指数运算溢出则不会产生错误。这是因为有时需要利用这种运算溢出设计程序,如查看某设备的状况标志、设计一个伪随机数发生器等。

要注意的另一个问题是整数运算的符号连续,操作数的不同符号类型会使相同表达式的值产生不同的内部表示。例如,若 x 是短整数, $x = -3$; 表达式 $x/2$, 当 x 为有符号数时, 结果为 -1 , 其二进制数为 1111111111111111 , 若 x 为无符号数, 结果为 $32\ 766$, 二进制数为 0111111111111110 。采用补码表示的数据,在进行 $+$ 、 $-$ 、 $*$ 运算时一般不必考虑符号连续,只考虑符号类型的不同产生的溢出问题, $/$ 运算不必考虑溢出,但可能存在运算结果的不同内部表示数值。

4. 算术赋值运算

算术运算与赋值表达式结合在一起形成算术赋值运算,在 C 语言提供了更简单的表达方式来描述这类运算,如下所示。

$x = x + y$ 可表示为 $x += y$

$x = x - y$ 可表示为 $x -= y$

$x = x * y$ 可表示为 $x *= y$

$x = x / y$ 可表示为 $x /= y$

$x = x \% y$ 可表示为 $x \% = y$

$x = x + 1$ 可表示为 $x++$ 或 $++x$

$x = x - 1$ 可表示为 $x--$ 或 $--x$

这种简单的表达方式不仅简化了表达式的写法,更重要的是提高了编译程序实现的效率。例如, $x++ = y$ 比 $x = x + y$ 少求了一次 x 的原值; $x++$ 比 $x + 1$ 效率还高,因为编译程序可以用机器语言的增量语句实现。

作为一个熟练的 C 语言程序员,应该可以充分利用这类运算符。但程序设计初学者可暂时不采用这类运算符,以便将注意力集中在其他关键机制上。

2.4.3 关系运算符及关系运算表达式

关系运算符用于比较两个量的大小关系,将用关系运算符把两个表达式连接起来,进行关系运算的式子称为关系表达式。

1. 关系运算符

关系运算符全部是二元运算符,包括以下几种。

<	小于
<=	小于或等于
>	大于
>=	大于或等于
==	等于
!=	不等于

这些关系运算的结果与日常概念是一致的。

关系运算的结果是整型。如果两个操作数满足关系运算符所要求的比较关系,则结果为 1(表示布尔值 true),否则为 0(表示布尔值 false)。例如, $13 > 27$ 的求值结果为 0,而 $13 \leq 21$ 的求值结果为 1。

2. 关系表达式

两个表达式通过小于(<)、小于或等于(<=)、等于(==)、大于或等于(>=)、大于(>)、不等于(!=)等关系运算符可组成一个新的表达式,称为关系表达式。参加关系表达式的操作数可以是字符型、整型、浮点型、双精度型等,但关系表达式的求值结果只能为 1(表示真)或 0(表示假)。例如,以下表达式均是关系表达式。

$x \leq (y - z)$

$a != b + 1$

2.4.4 逻辑运算符及逻辑表达式

逻辑运算符用于逻辑运算,用逻辑运算符把表达式连接起来就构成了逻辑表达式。

1. 逻辑运算符

通常程序设计语言都提供布尔数的与、或、非运算。逻辑运算也就是布尔运算,主要包括的逻辑运算符有 &&(与)、||(或)、!(非)等。其中,! 是一元运算符,而 && 和 || 是二元运算符。

在 C 语言中没有将布尔类型作为基本数据类型,表示逻辑真与逻辑假的方法是对于任何基本类型的数据来说的,只要其值不为 0 则当作真。

这些逻辑运算的含义与日常概念是一致的。在 $x \&\& y$ 中,只要 x 和 y 有一个为 0,运算结果则为 0,否则结果为 1;如果 $x || y$ 中 x 和 y 至少有一个不为 0,则运算结果为 1,否则结果为 0;如果 $! x$ 中 x 为 0,运算结果为 1, x 不为 0 时运算结果为 0。

2. 逻辑表达式

通过逻辑运算符连接组成的表达式称为逻辑表达式。参加逻辑运算的操作数可以是字符型、整型,但逻辑运算的求值结果只能为 1(表示真)或 0(表示假)。例如,以下表达式均为逻辑表达式。

$x \&\& (y || z)$

$! (x \&\& 1)$

例如,判断某年是否为闰年的逻辑表达式可表示为:

$(year \% 4 == 0 \&\& year \% 100 != 0) || year \% 400 == 0$

如果上面的表达式值为 1,则说明 $year$ 是闰年,否则不是。

2.4.5 赋值表达式

在 C 语言中,赋值语句本身也可作为表达式,称为赋值表达式。整个赋值表达式的类型是赋值语句左边的类型,整个赋值表达式的求值结果也是赋值后左边的值,如下所示。

“ $x = y + 1$ ”是一个赋值表达式。由于赋值表达式本身求值还可作为一个值使用,所以“ $z = x = y + 1$ ”也是一个赋值表达式,而“ $z + (x = y + 1)$ ”则是一个算术表达式。但是不提倡

这种简化的连续写法,在许多情况下它使程序变得难以理解。

2.4.6 逗号表达式

由逗号分开的表达式序列仍是表达式,称为逗号表达式。其求值次序是从左到右依次求值,整个表达式的值是最后一个表达式的值。例如:

```
balance+=amount,balance>max_balance
```

先将 amount 与 balance 的值相加并将结果存放在变量 balance 中,然后比较 balance 与 max_balance 值的大小,整个逗号表达式的值是比较结果,将是 0 或 1。

【例 2.5】 逗号表达式用法举例。

```
main()
{
    int balance,amount,max_balance;
    balance=10;
    amount=20;
    max_balance=30;
    balance=(balance+=amount,balance>max_balance);
    printf("%d\n",balance); //打印的结果为 0
}
```

2.4.7 表达式的运算顺序

表达式求值的一个重要问题是求值的次序。对于有多个运算符的表达式先进行哪一个运算? C 语言与许多程序设计语言一样,约定先计算括号内的表达式,再计算括号外的表达式,如表达式“a+((b+c)*d)”指明了求值次序是先求“b+c”,其结果再与 d 相乘,最后再加上 a。在每种情况下都由程序员用括号显式指明每一步求值过程是相当麻烦的,因此,程序设计语言引入运算符优先级与结合性质的概念来简化这一问题,不同程序设计语言规定的优先级与结合性质会有出入,但一般满足先乘除、后加减且从左到右进行计算。例如,按 C 语言规定,表达式“a+(b+c)*d”与表达式“a+((b+c)*d)”的求值结果是相同的。后面还会讨论运算符的优先级与结合性质。

表达式除了求值之外,还可能产生副作用(side effects)。所谓副作用是指表达式求值之外的任何其他动作。例如,赋值表达式会修改一个变量的值。为了减少对表达式理解的二义性,应该尽量避免在同一表达式中对一个变量的值修改两次或两次以上。

2.4.8 条件运算

条件运算符“?:”比较特殊,是 C 语言中仅有的一个三元运算符,它将一个表达式分为 3 个部分,如下所示。

```
e1? e2:e3;
```

计算这一表达式时,首先对表达式 e1 进行求值,如果结果不为 0,则对表达式 e2 进行求值,并将 e2 的求值结果作为整个表达式的值(不对 e3 求值);如是 e1 结果为 0,则对表达式

e3 进行求值,并将 e3 的求值结果作为整个表达式的值(不对 e2 求值)。

例如:

```
(amount<=balance)? amount;0;
```

如果 amount 的值不超过 balance 的值,则整个表达式的值为 amount 的值,否则整个表达式的值为 0。

条件运算还可嵌套使用,如下所示。

```
(amount==0)? 0:((amount<0? -1:1);
```

如果 amount 为 0,则整个表达式的值为 0,如果 amount 为负数,则整个表达式的值为 -1,否则整个表达式的值为 1。但条件运算嵌套层次太深时,表达式比较难以阅读和理解。

条件运算完全可以用 C 语言的条件语句 if 来取代,但在许多情况下,使用条件运算更加简洁、方便。具体的用法将在分支结构中再进行详细介绍。

2.4.9 sizeof 运算

sizeof 运算用于计算操作数所占存储空间的大小,计算结果以字节为单位,计算结果的类型是 unsigned。

sizeof 运算有以下两种形式。

- 一种是 sizeof 后面接一个表达式,此时 sizeof 运算只计算表达式所属类型的存储空间大小,而与表达式的值无关。
- 另一种是 sizeof 后面接一对由圆括号括住的类型名字,表示计算该类型的存储空间大小。sizeof 运算主要用于管理程序中各种类型数据所占的存储空间。

【例 2.6】 sizeof 运算符的用法举例。

```
//c program_6
#include <math.h>
main()
{
    int a=0xb7;
    printf(" %d\n",sizeof 88.9L);
    printf(" %d\n",sizeof 3.14159);
    printf(" %d\n",sizeof(2.3f));
    printf(" %d\n",sizeof(int));
    printf(" %d\n",sizeof(float));
    printf(" %d\n",sizeof(char));
    printf(" %d\n",a);
    a=a<<2;
    printf(" %d\n",sizeof(a));
    printf(" %x\n",a);
}
```

2.4.10 运算符优先级与结合性质

表达式的求值次序取决于表达式中各种运算符的优先级与结合性质。优先级高的运算先做,优先级低的运算后做;同一优先级的运算按结合性质从左到右(称为左结合,如通常使用的四则运算 $x+y+z$ 、 $a * b/c$ 等),或从右到左(称为右结合,如求幂运算 5^4 或赋值运算 $x=y=z$ 等)运算。在绝大多数程序设计语言中,括号内的运算总是优先于括号外的运算先行求值。

C 语言规定的部分运算符优先级与结合性质如表 2-5 所示,表中排在上面的运算符有较高的优先级,同一行中的运算符优先级相同。

表 2-5 C 语言的运算符优先级与结合性质

优 先 级	运 算 符	结合性质
1	() [] -> .	左结合
2	! ~ ++ -- + - sizeof * & (类型)	右结合
3	* / %	左结合
4	+ -	左结合
5	<< >>	左结合
6	<<=>>=	左结合
7	== !=	左结合
8	&.	左结合
9	^	左结合
10		左结合
11	&&.	左结合
12		左结合
13	?:	右结合
14	= += -= *= /= %= < <=>>= &.= ^= =	右结合
15	,	左结合

按照表 2-5 中的规定,下列表达式有唯一的求值次序。

表达式	求值次序
(1) $a+b-c$	$(a+b)-c$
(2) $a+b * c$	$a+(b * c)$
(3) $a=b c$	$a=(b c)$
(4) $a+b < c \& \& . d == e$	$((a+b) < c \& \& . (d == e))$

表达式(1)中的运算符+和-的优先级是相同的(位于表中同一行),而其结合性质为左结合,所以从左到右计算,即先算出 $(a+b)$ 的值之后再与 c 的值相减。表达式(2)中运算符 $*$ 比 $+$ 的优先级更高,所以不用考虑结合性质,先求 $(b * c)$ 的值之后再与 a 的值相加。表达式(3)也是因为逻辑运算符 $||$ 比赋值运算符优先级高,所以先对 $b || c$ 求值,然后将结果赋给

变量 a。表达式(4)稍复杂,4 个运算符中+优先级最高、<次之、&& 最低,所以求值次序是先求出式(a+b)的值,然后将这个值与 c 的值作比较,再求 d==e 的值,最后才进行 && 运算。

细心的读者可能已经发现,这种表达式的写法难以理解并容易产生错误,特别是对同一时期使用多种语言的程序员,更难记住不同语言中这么多运算符的优先级。既然看到别人写出的程序是这样难读,那么在写程序时就应该养成一种良好的习惯,除非是非常简单直观的运算,否则应该尽量使用括号消除表达式中求值的二义性。上述表达式(1)、表达式(2)和表达式(3)是可取的,表达式(4)应避免。

也不主张写太长的表达式,过长的表达式容易引起对求值次序的误解。如果表达式太长,应考虑是否可引入合适的中间变量与赋值运算简化它们。

2.5 数据类型转换

从以上关于基本数据类型的讨论知道,C 语言中字符类型与整数类型之间的关系是相当密切的,可以将字符理解为一个整数,也可将一个整数理解为一个字符。在许多情况下,类型之间的转换是非常有意义的。

2.5.1 隐式类型转换

当表达式中存在不同类型的操作数时,为对这个表达式进行求值,编译程序需要对其中的部分操作数自动进行类型转换,以保证二元运算符两边操作数的类型是一致的,这种类型转换称为隐式类型转换。例如,age 是整型变量,计算工龄工资的表达式如下。

```
age * 1.50
```

其中,运算符“*”左、右两边操作数的类型不一致,左边是 int 型,右边是 double 型,所以必须首先将 age 转换为 double 型,然后再求表达式值,其结果也为 double 型。这种类型转换并不改变 age 本身的类型,即下次再使用 age 时,它仍是 int 型。

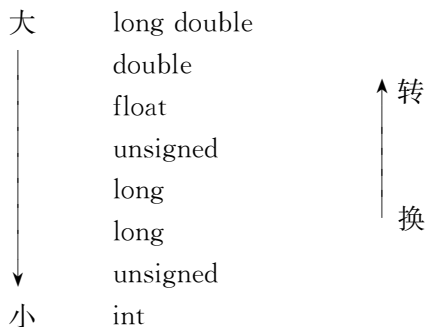
对于参加算术运算、逻辑运算和关系运算的操作数,C 语言规定了如下隐式类型转换规则。

(1)除 unsigned short 外,所有 char、short 型的数据无条件转换为 int 型后才参加运算,这种转换过程称为类型规范化。类型规范化也适合于位运算。例如:

```
char a='\xB7';
```

虽然声明 a 为字符型(仅占一个字节),但参加位运算时必须转换为整型,因而 a<<2 的结果是整型,左移两位不会产生溢出。

(2)在其他情况下,如果一个二元运算符两边操作数的类型不一致,则总是将取值范围的最大值较小的类型转换为取值范围的最大值较大的类型,取值结果也是这种取值范围的最大值较大的类型。取值范围从大到小如下所示。



这种转换规则的目的是在转换过程中尽量不丢失信息。

注意,对于赋值运算,C语言规定了不一样的类型转换规则:当赋值表达式左、右两边操作数的类型不一致时,右操作数首先转换为左操作数的类型,然后将转换结果赋值给左操作数,整个赋值表达式的结果类型是左操作数的类型。显然,这种转换过程常常会丢失部分数据。

由于隐式类型转换不反映在程序文本中,丢失部分数据后又不会报告错误,所以要特别小心不同类型的数据在一起的混合运算。一个好的建议是尽量避免隐式类型转换,如果类型转换过程会导致信息丢失,应该使用显式的强制类型转换。

2.5.2 强制类型转换

在程序中,也可以显式地控制类型转换,类型转换规则不必按前面的规定,这种转换过程称为强制类型转换。强制类型转换具有最高优先级。强制类型转换的形式如下。

(类型名)表达式

例如,已知矩形面积为 20,长为 8,求其宽度。

```
printf("Width is: %d\n",20/8);
```

输出结果是 2,误差很大。主要原因是 20 与 8 均为整数,因而“/”作为整除运算,结果将转换为整数。改进上述语句的方法是使用强制类型转换,让除法运算结果为浮点类型。以下两条语句均可达到目的,输出结果是 2.5。

```
printf("Width is: %f\n",(float)20/8);
```

```
printf("Width is: %f\n",20/(float)8);
```

但以下语句在运算以后才作强制类型转换,输出还是错误的结果 2.000000。

```
printf("Width is: %f\n",(float)(20/8));
```

2.6 位 运 算

位运算主要用于较低层的程序设计。位操作是对操作数以二进制位(bit)为单位进行数据加工的。使用位运算必须正确理解数据的内部表示。例如,处理某一状态标志字时,希望仅取出其中的某几个二进制位进行处理。参加位运算的操作数类型主要是 char 型与 int 型及其修饰符 long 和 unsigned。将位运算的操作数看做二进制数或十六进制数将使位运算

更容易理解。C 语言一共提供了以下 6 种位运算。

~	位求反
&	位与
	位或
^	位异或
<<	左移
>>	右移

其中,除~是一元运算符外,其余均为二元运算符。

(1)“~”是对操作数字节中的二进制数逐位求反,即值为 0 的位变成 1,值为 1 的位变成 0。显然对于任意数 x ,都有 $\sim(\sim x) = x$ 。例如,0x4A 对应的二进制字节为 01001010,求反结果为 10110101,即 0xB5。

(2)“&”是将两个操作数的对应位作逻辑与运算。两个操作数对应的位均为 1 时,运算结果对应的位为 1,否则为 0。例如 0x5A&0xB7(即 01011010 与 10110111 逐位求逻辑与)结果为 0x12(即 00010010)。“位与”的典型用法是用来取出标志字中某一位的值。例如,一个标志字节 flag 的每一位都对应设备的某种状态,如果需要取出左起第三位的状态,只要用 00100000(即 0x20)与 flag 进行位与运算,然后判断运算结果是否为 0 即可。如果该状态为 0,则关系运算“(flag&0x20)==0”的值为 1,否则值为 0。位与的另一种典型用法是用来设置标志字中的某一位为 0,例如想要设置 flag 左起字节的第三位的值为 0,只要用 11011111(即 0xDF,相当于 0x20 位求反)与 flag 进行位与运算,运算结果会将 flag 的第三位设置为 0(不管该位原来是什么值),而不影响其他位的原值。

(3)“|”是将两个操作数的对应位作逻辑或运算。两个操作数对应的位均为 0 时,运算结果对应的位为 0,否则为 1。例如,0x55|0xB4(即 01010101 与 10110100 逐位求逻辑或)结果为 0xF5(即 11110101)。“位或”的典型用法是用来设置标志字中的某一位为 1。例如,要设置 flag 左起第三位的状态为 1,只要用 00100000(即 0x20)与 flag 进行位或运算,运算结果会将 flag 的第三位设置为 1(不管该位原来是什么值),而不影响其他位的原值。

(4)“^”是将两个操作数的对应位作异或运算。两个操作数对应的位相同(即均为 0 或均为 1)时,运算结果对应的位为 0,否则为 1。例如,0x55^0xB4(即 01010101 与 10110100 逐位求逻辑异或)结果为 0xE1(即 11100001)。“位异或”有一个重要性质,对于任意 y 都有 $(y \wedge x) \wedge x = y$,即对任意数据连续用 x 作两次位异或运算的结果是其本身。由于这一性质,“位异或”经常用于对数据进行简单的加密与解密, y 是被加密的数据, x 是密码,数据以 $y \wedge x$ 的加密形式存放或传输,使用时再用 x 作一次位异或即得到解密数据。

(5)“<<”是将左操作数的位向左移,移动次数由左操作数决定。每次移动过程中,右端出现的空位补 0,移出左端的位舍弃。例如,0xB4<<2(即 10110100 向左连移两次)结果为 11010000(即 0xD0)。

(6)“>>”是将左操作数的位向右移,移动次数由右操作数决定。每次移动过程中,左端出现的空位按原来的最左位的值补入,移出右端的位舍弃。例如,0xB4>>2(即 10110100 向右连移两次)结果为 11101101(即 0xED)。

由于左、右移位运算会丢失部分信息,所以左、右移位是不可逆的,即 $(x \ll n) \gg n$ 或 $(x \gg n) \ll n$ 不一定等于 x 。

与加、减、乘、除运算类似,位运算也提供了更简便的运算符 $\ll$ 、 $\gg$ 、 $\&$ 、 $|$ 及 $\wedge$ 。例如, $a=a\&b$ 可以写为 $a\&=b$, $x=x\ll y$ 可以写为 $x\ll=y$ 。

【例 2.7】 位求反运算举例。

```
main()
{
    unsigned char a,b;
    a=0x9a;
    b=~a;
    printf("a: %x\tb: %x\n",a,b);
}
```

运算结果如图 2-3 所示。

【例 2.8】 位逻辑运算举例。

```
main()
{
    unsigned char a,b;
    a=0xb9;
    b=0x83;
    printf("a AND b=0x %x\n",a&b);
    printf("a OR  b=0x %x\n",a|b);
    printf("a NOR b=0x %x\n",a^b);
}
```

运算结果如下。

a AND b=0x81

a OR b=0xbb

a NOR b=0x3a

【例 2.9】 左移位操作举例。

```
main()
{
    unsigned char a,b;
    a=0x5b;
    b=a<<2;
    printf("a:0x %x\n",a);
    printf("b:0x %x\n",b);
}
```

运算结果如图 2-4 所示。

a:0x5b

b:0x6c

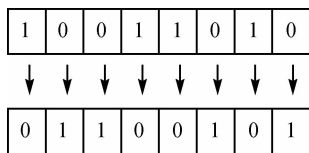


图 2-3 位求反运算结果

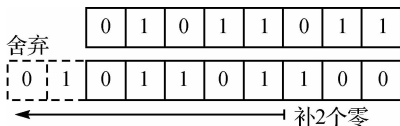


图 2-4 左移位操作示意图

【例 2.10】 右移位操作举例。

```
main()
{
    unsigned char b;
    char a=-8;
    b=248;
    a>>=2;
    b>>=2;
    printf("a=0x%x= %d\n",a,a);
    printf("b=0x%x= %d\n",b,b);
}
```

运算结果如下。

a=0xfffe=-2

b=0x3e=62

请读者画出本例右移位的操作示意图。

2.7 数组和指针

为了帮助读者能较轻松地学习和理解 C 语言的数组和指针的概念,在本节将简单地介绍一些有关数组和指针的基本概念和用法,详细的介绍将放在第 6 章。

2.7.1 数组

变量和常量只能用来表示少量相互之间没有内在联系的数据,大量的成批数据要使用数组(array)来表示。数组是具有一定顺序关系的若干变量的集合体。组成数组的变量称为数组的元素,简称元素。C 语言中数组的元素变量用数组名后面跟有方括号“[]”的下标表示,如下所示。

a[2],b[2][3],name[100],y[2][3][4]

其中,a、b、name、y 是用户命名的数组名。只有一个下标的数组称为一维数组,带有 2 个、3 个下标的分别是二维数组、三维数组。

C 语言数组的下标是从 0 开始的,下标必须是整型常量或变量。

数组在内存中存储时,是按下标递增的顺序连续存储各数组元素的值。数组名代表了数组所占有的这一片连续存储区域的首地址。数组的首地址也就是数组第一个元素的地址。例如,上面的数组名为 a 的数组首地址是 a 或 &a[0],数组名是一个常量地址,不能对它赋值或进行 & 运算。

由于数组是元素的集合体,是由若干带下标的变量所组成的,所以数组也具有与其他变量相同的数据类型和存储类别,因此,数组在使用前必须对数组的类型、数组的存储方式进行声明。声明的方式如下。

[存储类别] 数据类型 数组名[元素个数];

若数组的存储类别采用系统的默认值(自动存储类别),则可省去存储类别的说明。编译系统在处理数组声明语句时,将为该数组在内存中分配与类型相对应的一定数量的存储空间。例如,有如下数组声明。

```
int a[100], num[3][4];
```

```
static char name[200]; //static 为静态存储类别
```

在第一条声明语句中,定义了两个整型数组,其中一维数组 a 共有 100 个元素,即 $a[0] \sim a[99]$ 。

二维 num 数组有 3 行 4 列,共 $3 \times 4 = 12$ 个元素。

存储类别采用隐含类别。num 数组的结构如图 2-5

所示。第二条语句定义了字符型的 name 数组,有

200 个元素,并定义采用静态存储类别。

在定义数组的同时,可以对数组进行初始化。如下所示。

```
int b[5] = {1, 3, 2, 4, 6};
```

这时在定义整型数组 b 有 5 个元素时,同时给它们赋了初始值,即 $b[0] = 1$ 、 $b[1] = 3$ 、 $b[2] = 2$ 、 $b[3] = 4$ 、 $b[4] = 6$ 。

若对数组进行初始化的数据的个数与数组元素的个数相同,则定义数组时的元素个数可以省略,即下面的定义与上面的定义是等价的。

```
int b[] = {1, 3, 2, 4, 6};
```

【例 2.11】 数组初始化举例。

```
main()
```

```
{
```

```
    int a[5] = {1, 3, 5, 7, 9}, b[2][3];
```

```
    b[0][2] = 10;
```

```
    b[1][2] = 100;
```

```
    printf("a array: %d %d %d %d %d\n", a[0], a[1], a[2], a[3], a[4]);
```

```
    printf("b[0][2] = %d\n", b[0][2]);
```

```
    printf("b[1][2] = %d\n", b[1][2]);
```

```
}
```

对二维数组的处理将在第 6 章复合数据类型中进行详细介绍,这里先不讨论。

2.7.2 指针

指针是 C 语言中最具有代表性的特征之一。C 语言程序设计中指针的概念是必不可少的内容。利用指针可以直接对内存中各种不同数据结构的数据进行快速处理。并且利用指针可以很方便地在各函数之间进行数据传递。

1. 指针的声明

指针也是一种变量,它与其他变量一样也具有 3 个要素:指针的名字、指针的类型、指针的值。它也需要占用存储空间。但它与其他普通变量有着不同之处,指针的存储空间中存放的是另一个变量的地址。而普通变量的存储空间里存放的该变量自己的数值。指针的类

num[0][0]	num[0][1]	num[0][2]	num[0][3]
num[1][0]	num[1][1]	num[1][2]	num[1][3]
num[2][0]	num[2][1]	num[2][2]	num[2][3]

图 2-5 num 数组的结构

型不是它自己的类型,而是它指向的对象的数据类型。

在 C 语言中,根据指针所指对象的不同,指针有不同的声明方式和使用形式。通常有指向一般变量的指针、指向数组的指针、指向结构体的指针、指向函数的指针以及指向指针的指针等。在第 6 章之前只学习最简单最基本的指向变量和指向一维数组的指针。

假设变量 x 的存储地址是 2001H, x 的值是 35, 指针变量 p 中存放了变量 x 的地址, 如图 2-6 所示。从图 2-6 上可以看到以下几方面的表示。

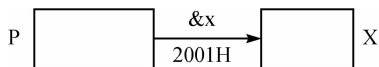


图 2-6 指针变量 p 指向了变量 x

- (1) p 指针的值, p 的值是 2001H。
- (2) x 的值, x 的值是 35。
- (3) x 的地址, x 的地址是 2001H, 可以表示为 $\&x$ 。
- (4) p 指针与 x 的联系, 指针变量 p 指向了变量 x。

可以看出, 当一个指针指向某一个对象(变量)时, 它与这个对象就有了密切的关系。

如同使用变量一样, 在使用一个指针之前也必须遵循“先声明, 后使用”的原则来声明指针。如何把一个变量声明成指针变量呢? 如何让一个指针指向一个对象(变量)呢? 下面将讨论这些问题。

指针的声明, 即在变量名前加上“*”, 这就使得指针变量不同于其他变量; 还要说明它将要指向的对象的数据类型(请注意, 指针本身是不具有类型的, 指针的类型是由系统定义的)。其定义方式如下。

对象的类型 * 指针变量名

例如:

```
int * px;  
float * py;  
char * pc;
```

在此声明了 3 个指针变量, 一个指向整型变量的 px 指针变量, 一个指向实型变量的 py 指针变量和一个指向字符型变量的 pc 指针变量。

2. 指针指向某个对象

在介绍让指针指向某个对象时, 首先要弄清楚变量的地址、变量的值的表示。

- 变量的地址。变量所占有的存储空间的地址, 地址的表示采用地址运算符“&”。例如, x 的地址表示为 $\&x$ 。
- 变量的值。变量所占有的存储空间的内容, 用指针表示为“*p”。例如, 若 $x=100$, 用指针 p 指向 x, 则 *p 为 100。

指针是用来指向某个对象(变量)的, 也就是让指针与某个变量联系起来。指针指向某个对象有如下两种方式。

- (1) 用赋值语句通过给指针赋值使得指针指向某个变量。

```
int * px, x;  
float * py, y;  
char * pc, c;
```

```
px=&x; }
py=&y; } 赋值语句
pc=&c; }
```

这里指针 px 指向 x, 指针 py 指向 y, 指针 pc 指向 c, 指针与所指对象的关系如图 2-7 所示。

(2) 用初始化的方式让指针指向某个变量。

初始化是在声明变量的同时赋予的初始值, 故称为初始化。由于指针保存的是地址, 因此, 对指针初始化时赋予的初
始值必须是地址量。初始化的方式如下。

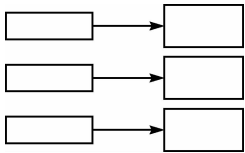


图 2-7 指针与所指对象的关系

对象的类型 * 指针变量名 = 初始地址值

如上面的表示也可改为如下形式。

```
int x, *px=&x; //声明 px 指针的同时, 让 px 指向 x
float y, *py=&y;
char c, *pc=&c;
```

这样既完成了指针变量的定义, 同时也使得指针得到了初始值, 即所指向的对象的地址。

【例 2.12】 指针初始化举例。

```
main()
{
    int x, *px=&x; char c, *pc=&c;
    float y, *py;
    py=&y;
    x=100;
    y=3.5;
    c='$';
    printf("&x=0x%x\ttx= %d\n", px, *px);
    printf("&y=0x%x\tty= %f\n", py, *py);
    printf("&c=0x%x\ttc= %c\n", pc, *pc);
}
```

程序运行的结果是:

```
&x=0x10c4 x=100
&y=0x10ba y=3.500000
&c=0x10c0 c= $
```

2.7.3 指针和数组的简单用法

1. 指向一般变量的指针

这种情形非常简单, 这里只举两个例子。

【例 2.13】 指针用法举例之 1。

```
main()
```

```

{
    int x,y, *p=&x;
    x=100;
    y= *p;
    printf("x= %d\n", *p);
    printf("y= %d\n", y);
}

```

从图 2-8 中可以看到程序的运行结果。

【例 2.14】 指针用法举例之 2。

```

main()
{
    int x,y, *p1=&x, *p2=&y;
    p2=p1; //让 p2 和 p1 同时指向 x
    x=100;
    y=200;
    printf("x= %d\n", *p1);
    printf("y= %d\n", *p2);
}

```

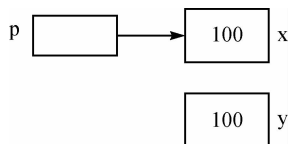


图 2-8 程序运行结果

程序中 p2 指针在初始化时是指向变量 y 的(见图 2-9 的实线箭头所示),随后采用赋值方式将 p2 的指向改为与 p1 同样的指向,即都指向 x,因此,打印 * p2 值实际上是 x 的值。

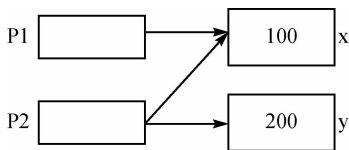


图 2-9 程序运行结果

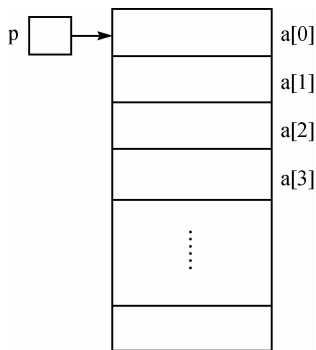


图 2-10 一维数组的指针

2. 指向一维数组的指针

在 C 语言中,指针和数组之间的关系十分密切,它们都可以处理内存中连续存放的一批数据。假设程序中声明了一个整型数组 a,共有 10 个元素。数组名代表了数组的首地址,因此,利用数组名,再通过下标的变化,就可以处理数组的任一元素。

如果在程序中声明一个整型指针,即“int * p”,让它指向 a 数组的方式是“p=a;”或“p=&a[0];”。

这两种方式都是使指针 p 指向数组 a 的首地址,如图 2-10 所示。知道了如何让指针指

向首地址,那么要让指针指向数组的第 i 个元素又如何表示呢? 由于数组名就是数组的首地址,因此,第 i 个元素的地址可以表示为 $a+i$ 或 $\&a[i]$ 。

可以看出, $a+i$ 就是 $\&a[i]$,都是第 i 个元素的地址。于是用指针就可以这样表示,因为 $p=a=\&a[0]$ 是指向数组的首地址,很显然, $p+i$ 就是指向第 i 个元素的地址。表 2-6 给出了用指针和数组分别表示数组的地址以及数组元素值的方法。

表 2-6 指针、数组对数组地址和值的表示

表示项	数组地址的表示		数组元素值的表示	
	数组表示	指针表示	数组表示	指针表示
表示首地址	a 或 $\&a[0]$	p 或 $p+0$	$a[0]$ 或 $*(a+0)$	$*(p+0)$
表示第 i 个元素	$a+i$ 或 $\&a[i]$	$p+i$	$a[i]$ 或 $*(a+i)$	$*(p+i)$

【例 2.15】 指针与数组的关系举例之 1。

```
main()
{
    int i,a[10],*p;
    p=a;
    for(i=0;i<10;i++)
        a[i]=i*2;
    for(i=0;i<10;i++)
        printf("a[ %d]= %d\t*(p+ %d)= %d\n",i,*(a+i),i,*(p+i));
}
```

程序的输出结果如下。

```
a[0]=0      *(p+0)=0
a[1]=2      *(p+1)=2
a[2]=4      *(p+2)=4
a[3]=6      *(p+3)=6
a[4]=8      *(p+4)=8
a[5]=10     *(p+5)=10
a[6]=12     *(p+6)=12
a[7]=14     *(p+7)=14
a[8]=16     *(p+8)=16
a[9]=18     *(p+9)=18
```

由此可以看出, $a[i]$ 、 $*(p+i)$ 与 $*(a+i)$ 的表示是完全等价的,都是第 i 个元素的值。

【例 2.16】 指针与数组的关系举例之 2。

```
main()
{
    int i,a[]={10,20,30,40,50},*p;
    p=a;
    for(i=0;i<5;i++)
```

```

    printf("a[ %d]= % -6d",i,p[i]);
    printf("\n");
    for(i=0;i<5;i++)
        printf(" * (p+ %d)= % -4d",i,a[i]);
    printf("\n");
}

```

程序运行结果如下。

```

a[0]=10    a[1]=20    a[2]=30    a[3]=40    a[4]=50
* (p+0)=10  * (p+1)=20  * (p+2)=30  * (p+3)=40  * (p+4)=50

```

可以看出, `p[i]`、`a[i]`、`*(a+i)`和`*(p+i)`也都是完成同样功能的。

需要特别指出的是,指针和数组在访问地址中的数据时,其表示形式是相同的,因为指针和数组名都是地址量,但指针和数组名在本质上是不同的,指针是地址变量,而数组名则是地址常量。这就意味着指针可以接受赋值,如 `p=a` 是合法的,但数组名则不能被赋值,如果有 `a=p` 的赋值语句则是错误的。

2.8 简单应用程序举例

利用本章学习的内容,下面着手编写简单的应用程序。

【例 2.17】 如果今天天气预报的气温是 81.25°F,相应的摄氏温度是多少? 根据常识,将华氏温度 F 转换为相应摄氏温度(°C)C 的公式如下。

$$C = (F - 32) * 5 / 9$$

利用这个公式来解决问题。

程序 1:

```

/* 将一个华氏温度转换为摄氏温度。 */
#include <stdio.h>
main()
{ printf(" %f\n", (81.25 - 32) * 5 / 9); }

```

程序 1 给出了解决本问题的一个方案。该程序虽然解决了今天的问题,但是明天天气有变化时,还必须修改程序,并重新编译、链接和运行。

下面的程序 2 对程序 1 作了改进,让程序每次运行时才确定当天气温。程序运行过程中,由用户输入当天气温的华氏度数。为了使程序可以记住用户输入的数据并用作计算,必须引入一个变量存放用户的输入。为了让程序易读、易理解,为变量起一个有含义的名字 `fahrenheit_temperature`,让变量名字自己说明自己的用途。但这个名字也确实太长了,不易记忆,拼写也容易产生错误。在天气预报时华氏温度习惯缩写为 `Fah.`,所以将变量名字设计为 `fah_temp`,`fah_temp` 选择什么样的类型呢? 整型足够大了,但气温可能还有小数部分,所以选择 `float` 型比较合适。

程序 2:

```

/* 程序:fah2cel.c */

```



```

/* 功能:将用户输入的华氏温度转换为摄氏温度并输出 */
#include <stdio.h>
main()
{
    float fah_temp;                //用户输入的华氏温度
    printf("Enter Fahrenheit temperature please:"); //给出输入提示
    scanf("%f",&fah_temp);        //由用户输入华氏温度
    printf("%f\n",(fah_temp-32)*5/9); //转换为摄氏温度并输出
}

```

【例 2.18】 输入两个数 a 和 b,打印出两个数的和与积(用指针表示)。

```

main()
{
    int a,b,t,y,*p1,*p2;
    a=100;
    b=20;
    p1=&a;
    p2=&b;
    t=*p1+*p2;                //相当于 t=a+b
    y=*p1* *p2;                //相当于 y=a*b
    printf("t=a+b= %d\t y=a*b= %d\n",*p1,*p2);
}

```

【例 2.19】 分析程序的运行结果。

```

main()
{
    int a[10],*p;
    a[2]=10;
    a[5]=20;
    p=a; //指针指向数组的首地址
    printf("a[2]= %d\t * (p+2)= %d\n",a[2],*(p+2));
    printf("*(a+5)= %d\t p[5]= %d\n",*(a+5),p[5]);
}

```

程序运行结果如下。

```

a[2]=10          * (p+2)=10
*(a+5)=20       p[5]=20

```

程序的运行结果可以帮助我们进一步理解表 2-6 给出的指针和数组分别对数组地址及数组元素值的表示方法。

本章小结

本章介绍了 C 语言程序设计中最基本的内容和知识。通过本章的学习,要求读者重点掌握 C 语言的基本数据类型、数据类型的声明方式、数组和指针的声明及初始化方式以及用数组和指针表示数组地址和值的方式。

学习一种程序语言的目的是要学会用这种语言去编写程序,从而解决实际的应用问题,如何把一个问题用计算机语言描述成计算机可以执行的程序,这就是程序设计的宗旨。程序设计除了按照语言的语法规则进行正确编程之外,还应该注意编程的风格,这是初学者开始学习时应该学会养成好的程序设计习惯。一个好的程序员编写的程序应该使用户操作方便,因此,在程序中应该给出一些相应的提示信息。一个好的应用程序必定有良好的用户界面。用户界面友好是指用户可以根据屏幕上的各种操作提示,用简单、直接的方式输入数据或查询结果。编程过程中还应该养成在程序中增加注释的表示风格,这对增强程序的可读性将有很大的帮助。除此之外,还应该为编写的程序提供一个合适的程序名,并将程序名字写在注释中,当然这并不是 C 语言硬性要求的。程序起什么样的名字好呢?【例 2.17】是一个从华氏温度转换到摄氏温度的程序,为程序起名最好有同样的意义。但如果起名为 `fahrenheit_to_celsius` 则太长了,一般可以用含义相同的缩写表示,这里我们按英文习惯可用 `2` 代表 `to`,因此,这个程序可取名为 `fah2cel`。

最后,这个程序还必须存放在一个文件中,常用程序名作为文件名,文件的后缀名可按习惯定为 C,所以整个程序是存放在文件 FAH2CEL.C 中的。

习 题

1. 总结 C 语言关于标识符的有关规定。
2. 总结表达式和语句的区别。
3. 设 $x=4, y=-8$, 说明下列各题运算后, x, y 和 z 的值分别是多少?
(1) $z=(x++) * (--y)$
(2) $z=(++x) - (y--)$
(3) $z=(++x) * (--y)$
(4) $z=(x++) + (y--)$
4. 判断下列关系表达式或逻辑表达式的运算结果(1 或 0)。
(1) $10 == 9 + 1$
(2) $10 \&\& 8$
(3) $8 || 0$
(4) $!(3 + 2)$
(5) 设 $x=10, y=9, x >= 8 \&\& y <= x$
5. 编写一个程序测试所用的 C 语言编译系统中各基本类型的数据所占的字节数。
6. 编写一个程序测试一下把带小数的 float、double 和 long double 型数据用 printf() 函数输出时, 其小数部分是被截断还是四舍五入? 在第几位进行?
7. 要找比整数 i 大而又是一个整数 j 的倍数的值, 所用公式如下。
$$\text{next_multiple} = i + j - i \% j$$

试编写一个程序,来针对(i,j)数值对(365,7)、(12258,23)、(996,4)进行计算。

8. C语言提供的合法关键字是()。

- A. swith B. cher C. Case D. default

9. 在C语言中,合法的字符常量是()。

- A. '\084' B. '\x43' C. 'ab' D. "\0"

10. 若已定义x和y为double型,则表达式“ $x-1, y=x+3/2$ ”的值是()。

- A. 1 B. 2 C. 2.0 D. 2.5

11. 合法的数组定义是()。

- A. int a[]="string"; B. int a[5]={0,1,2,3,4,5};

- C. char a="string"; D. char a[]={0,1,2,3,4,5};

12. 设a为整型变量,不能正确表达数学关系 $10 < a < 15$ 的C语言表达式是()。

- A. $0 < a < 15$ B. $a = 11 || a = 12 || a = 13 || a = 14$

- C. $a > 10 \& \& a < 15$ D. $!(a <= 10) \& \& !(a >= 15)$

13. 若t为double型,表达式“ $t=1, t+5, t++$ ”的值是()。

- A. 1 B. 6.0 C. 2.0 D. 1.0

14. 若有语句

```
char s[10]; s="abcd"; printf("%s\n", s);
```

则结果是(u代表空格)()。

- A. 输出abcd B. 输出a C. 输出abcduuuuu D. 编译不通过

15. 若有以下语句。

```
int u=010, v=0x10, w=10;
```

```
printf("%d, %d, %d\n", u, v, w);
```

则输出结果是()。

- A. 8,16,10 B. 10,10,10 C. 8,8,10 D. 8,10,10

16. 若有以下语句。

```
char c1='b', c2='e';
```

```
prinif("%d, %c\n", c2-c1, c2-'a'+'A');
```

则输出结果是()。

- A. 2,M

- B. 3,E

- C. 2,E

- D. 输出项与对应的格式控制不一致,输出结果不确定

17. 若有以下语句。

```
int a[10]={1,2,3,4,5,6,7,8,9,10}, *p=a;
```

则不能表示a数组元素的表达式是()。

- A. *p B. a[10] C. *a D. a[p-a]

18. 下面程序的输出结果是()。

```
main()
```

```
{
```

```
unsigned a=32768;  
printf("a= %d\n",a);  
}
```

A. a=32768 B. a=32767 C. a=-32768 D. a=-1

19. 以下运算符优先级最高的是()。

A. <= B. == C. =% D. &&