目　　录

第 0 章 概述 ..... 错误！未定义书签。
0.1 计算机的由来及组成 ..... 错误！未定义书签。
0.2 计算机程序 ..... 错误！未定义书签。
0.3 C 语言发展史 ..... 错误！未定义书签。
0.4 C 程序基本结构 ..... 错误！未定义书签。
0.5 C 程序开发步骤 ..... 错误！未定义书签。
0.6 集成开发环境 ..... 错误！未定义书签。
习题 ..... 错误！未定义书签。
第 1 篇 感知篇 ..... 错误！未定义书签。
第 1 章 数据的基本操作 ..... 错误！未定义书签。
1.1 数据的存储与输出 ..... 错误！未定义书签。
1.2 数据的输入与运算 ..... 错误！未定义书签。
1.3 数据的比较与判断 ..... 错误！未定义书签。
第 2 章 结构化程序设计初探 ..... 错误！未定义书签。
2.1 重复与循环语句 ..... 错误！未定义书签。
2.2 基本结构的组合 ..... 错误！未定义书签。
2.3 模块化编程 ..... 错误！未定义书签。
第 3 章 数据结构 ..... 错误！未定义书签。
3.1 数组 ..... 错误！未定义书签。
3.2 结构体 ..... 错误！未定义书签。
3.3 动态数组 ..... 错误！未定义书签。
3.4 文件 ..... 错误！未定义书签。
第 4 章 算法描述和编码规范 ..... 错误！未定义书签。
4.1 程序设计与算法描述 ..... 错误！未定义书签。
4.1.1 程序设计与算法 ..... 错误！未定义书签。
4.1.2 FC 流程图 ..... 错误！未定义书签。
4.1.3 NS 盒图 ..... 错误！未定义书签。
4.2 C 语言编码规范 ..... 错误！未定义书签。
习题 ..... 错误！未定义书签。
第 2 篇 详解篇 6
第 5 章 数据类型与输入输出 7
5.1 C 语言要素 7
5.1.1 字符集 7
5.1.2 标识符与关键字 7
5.1.3 可执行语句 8
5.2 数据类型 9
5.2.1 理解数据类型 9
5.2.2 变量 10
5.2.3 常量 11
5.2.4 整型数据 12
5.2.5 浮点型数据 14
5.2.6 字符型数据 16
5.3 输入与输出操作 18

5.3.1 输入与输出的概念 18
5.3.2 格式化输出函数 18
5.3.3 格式化输入函数 21
5.3.4 字符的输入与输出 22
5.4 编程错误 23
5.4.1 语法错误和警告 24
5.4.2 运行错误 25
5.4.3 逻辑错误 25
习题 26
第 6 章 运算符与表达式 27
6.1 概述 27
6.2 算术运算 28
6.3 赋值运算 30
6.4 表达式中的类型转换 32
6.4.1 隐式类型转换 32
6.4.2 显式类型转换 33
6.5 自增与自减运算 34
6.6 关系与逻辑表运算 36
6.7 其他运算符 38
6.8 运算符的优先级与结合性 39
6.9 案例分析 40
习题 44
第 7 章 选择结构 ..... 错误！未定义书签。
7.1 理解选择结构 ..... 错误！未定义书签。
7.2 简单分支语句 ..... 错误！未定义书签。
7.2.1 单分支 if 语句 ..... 错误！未定义书签。
7.2.2 双分支 if－else 语句 ..... 错误！未定义书签。
7.3 多分支语句 ..... 错误！未定义书签。
7.3.1 嵌套 if 语句 ..... 错误！未定义书签。
7.3.2 多分支 else if 语句 ..... 错误！未定义书签。
7.3.3 switch 语句 ..... 错误！未定义书签。
7.4 案例分析 ..... 错误！未定义书签。
习题 ..... 错误！未定义书签。
第 8 章 循环结构 ..... 错误！未定义书签。
8.1 理解循环结构 ..... 错误！未定义书签。
8.2 循环语句 ..... 错误！未定义书签。
8.2.1 while 语句 ..... 错误！未定义书签。
8.2.2 do 语句 ..... 错误！未定义书签。
8.2.3 for 语句 ..... 错误！未定义书签。
8.2.4 几种循环语句的比较 ..... 错误！未定义书签。
8.3 循环条件 ..... 错误！未定义书签。
8.3.1 计数器控制循环 ..... 错误！未定义书签。
8.3.2 标记控制循环 ..... 错误！未定义书签。
8.4 循环嵌套 ..... 错误！未定义书签。

8.4.1 循环嵌套结构 ..... 错误！未定义书签。
8.4.2 循环中的选择结构 ..... 错误！未定义书签。
8.5 循环中的跳转 ..... 错误！未定义书签。
8.5.1 break 语句 ..... 错误！未定义书签。
8.5.2 continue 语句 ..... 错误！未定义书签。
8.5.3 goto 语句 ..... 错误！未定义书签。
8.6 案例分析 ..... 错误！未定义书签。
习题 ..... 错误！未定义书签。
第 9 章 数组 ..... 错误！未定义书签。
9.1 理解数组 ..... 错误！未定义书签。
9.2 一维数组 ..... 错误！未定义书签。
9.2.1 一维数组定义 ..... 错误！未定义书签。
9.2.2 一维数组引用 ..... 错误！未定义书签。
9.2.3 一维数组初始化 ..... 错误！未定义书签。
9.2.4 一维数组案例分析 ..... 错误！未定义书签。
9.3 二维数组 ..... 错误！未定义书签。
9.3.1 二维数组定义 ..... 错误！未定义书签。
9.3.2 二维数组引用 ..... 错误！未定义书签。
9.3.3 二维数组初始化 ..... 错误！未定义书签。
9.3.4 二维数组案例分析 ..... 错误！未定义书签。
习题 ..... 错误！未定义书签。
第 10 章 函数 ..... 错误！未定义书签。
10.1 理解函数 ..... 错误！未定义书签。
10.2 函数定义和分类 ..... 错误！未定义书签。
10.2.1 函数定义 ..... 错误！未定义书签。
10.2.2 函数分类 ..... 错误！未定义书签。
10.3 函数调用和声明 ..... 错误！未定义书签。
10.3.1 函数调用 ..... 错误！未定义书签。
10.3.2 函数声明 ..... 错误！未定义书签。
10.4 函数参数和函数值 ..... 错误！未定义书签。
10.4.1 形式参数与实际参数 ..... 错误！未定义书签。
10.4.2 函数返回值 ..... 错误！未定义书签。
10.4.3 数组作函数参数 ..... 错误！未定义书签。
10.5 函数递归调用 ..... 错误！未定义书签。
10.6 变量作用域与生存期 ..... 错误！未定义书签。
10.6.1 变量作用域 ..... 错误！未定义书签。
10.6.2 变量存储类别与生存期 ..... 错误！未定义书签。
10.7 内部函数和外部函数 ..... 错误！未定义书签。
习题 ..... 错误！未定义书签。
第 11 章 指针 ..... 错误！未定义书签。
11.1 理解指针 ..... 错误！未定义书签。
11.2 指向变量的指针 ..... 错误！未定义书签。
11.2.1 指针变量定义 ..... 错误！未定义书签。
11.2.2 指针变量引用 ..... 错误！未定义书签。

11.3 数组与指针 ..... 错误！未定义书签。
11.3.1 一维数组与指针 ..... 错误！未定义书签。
11.3.2 二维数组与指针 ..... 错误！未定义书签。
11.3.3 指针数组 ..... 错误！未定义书签。
11.3.4 指向指针的指针 ..... 错误！未定义书签。
11.4 函数与指针 ..... 错误！未定义书签。
11.4.1 指针作函数参数 ..... 错误！未定义书签。
11.4.2 数组名作函数参数 ..... 错误！未定义书签。
11.4.3 返回指针值的函数 ..... 错误！未定义书签。
11.4.4 指向函数的指针 ..... 错误！未定义书签。
11.5 字符串 ..... 错误！未定义书签。
11.5.1 字符数组与字符串 ..... 错误！未定义书签。
11.5.2 字符串与指针 ..... 错误！未定义书签。
11.5.3 字符串函数 ..... 错误！未定义书签。
11.5.4 字符串程序举例 ..... 错误！未定义书签。
11.5.5 main 函数参数 ..... 错误！未定义书签。
11.6 动态空间管理 ..... 错误！未定义书签。
习题 ..... 错误！未定义书签。
第 12 章 自定义数据类型 ..... 错误！未定义书签。
12.1 结构体 ..... 错误！未定义书签。
12.1.1 结构体声明 ..... 错误！未定义书签。
12.1.2 结构体变量定义 ..... 错误！未定义书签。
12.1.3 结构体变量引用 ..... 错误！未定义书签。
12.1.4 结构体数组 ..... 错误！未定义书签。
12.1.5 结构体与指针 ..... 错误！未定义书签。
12.2 链表 ..... 错误！未定义书签。
12.3 枚举类型 ..... 错误！未定义书签。
习题 ..... 错误！未定义书签。
第 13 章 文件 ..... 错误！未定义书签。
13.1 文件概述 ..... 错误！未定义书签。
13.2 文件的打开与关闭 ..... 错误！未定义书签。
13.3 文件读写 ..... 错误！未定义书签。
13.3.1 字符读写函数 ..... 错误！未定义书签。
13.3.2 字符串读写函数 ..... 错误！未定义书签。
13.3.3 数据块读写函数 ..... 错误！未定义书签。
13.3.4 格式化读写函数 ..... 错误！未定义书签。
13.3.5 文本文件与二进制文件 ..... 错误！未定义书签。
13.4 文件的随机读写 ..... 错误！未定义书签。
13.5 文件检测函数 ..... 错误！未定义书签。
习题 ..... 错误！未定义书签。
第 3 篇 进阶篇 ..... 错误！未定义书签。
第 14 章 函数进阶 ..... 错误！未定义书签。
14.1 分解与抽象 ..... 错误！未定义书签。
案例 日期运算 ..... 错误！未定义书签。

同步练习 ..... 错误！未定义书签。
14.2 递归 ..... 错误！未定义书签。
案例 汉诺塔 ..... 错误！未定义书签。
同步练习 ..... 错误！未定义书签。
第 15 章 数组进阶 ..... 错误！未定义书签。
15.1 数据模型 ..... 错误！未定义书签。
案例 贪吃蛇游戏 ..... 错误！未定义书签。
同步练习 ..... 错误！未定义书签。
15.2 查找与排序 ..... 错误！未定义书签。
15.2.1 简单查找算法 ..... 错误！未定义书签。
15.2.1 简单排序算法 ..... 错误！未定义书签。
同步练习 ..... 错误！未定义书签。
第 16 章 数据管理 ..... 错误！未定义书签。
16.1 简单链表 ..... 错误！未定义书签。
案例 通讯录管理 ..... 错误！未定义书签。
同步练习 ..... 错误！未定义书签。
16.2 数据文件 ..... 错误！未定义书签。
案例 通讯录的存储 ..... 错误！未定义书签。
同步练习 ..... 错误！未定义书签。
附录 1 ASCII 表 45

第 2 篇 详解篇

在第一篇初步了解 C 语言基础上，本篇对 C 语言的各个要素进行深入、详细解读。对每个知识点按照理解、语法规则、使用方式的次序进行组织。

C 语言比较晦涩难懂，本篇对每个知识点力争做到深入浅出，针对深奥的内容，用浅显的事物做类比，进而使得深奥的问题变得简单易懂。

本篇对数据类型、输入输出、运算符、表达式、分支语句、循环语句、数组、函数、结构体、文件各个知识点进行了翔实的介绍。每部分针对问题的难易程度，配以适当比例的例题和小案例，来帮助读者理解。数组、函数与指针部分并未做一些较为复杂的案例，因为复杂一点的案例需要涉及到多章节的知识点，在学习完所有知识点后，第三篇会继续探讨。

通过本篇的学习，对计算机语言的一般语法规则和程序结构会有一个普遍性的认识，为学习其他计算机语言打下坚实的基础。

第 5 章 数据类型与输入输出

5.1 C 语言要素

程序设计的主要任务是处理各种数据，数据处理是通过执行一系列程序指令来完成的，而这些指令由特定字符按照严格的规则组成。掌握一门外语要按照以下步骤学习：首先要了解字母和音标，再学习用字母构成的单词，然后按照语法规则构成句子，最后才能用外语进行交流和读写文章。与其他语言一样，程序设计语言也有自己的词法规则和语法规则。本章介绍构成 C 程序的字符集、标识符和关键字，以及基本的输入输出方法。

5.1.1 字符集

字符用于组成标识符、字符串和表达式，C 语言能够识别的字符包括以下几类：

1. 字母

包括 26 个大写字母 A～Z 和 26 个小写字母 a~z，注意 C 程序区别大小写字母。

2. 数字

包括 10 个十进制数字 0～9。

3. 特殊字符

包括 29 个图形字符，见表 5.1。

表 5.1 C 语言字符集

+	-	*	/	%	<	=	>	!	&	|	∧	～	_	.
（	）	[	]	{	}	?	:	;	,	”	‘	#	\	

4. 空白符

包括不可打印的 5 个空白符号：空格符、回车符、换页符、横向制表符和纵向制表符。空白符只在字符常量和字符串常量中起作用。在其它地方出现时，只起间隔作用，编译程序时对它们忽略不计。在程序中适当的地方使用空白符将增加程序的可读性。

有些非英语键盘不一定支持上述所有的字母，ANSI C 引入了三元字符，为某些键盘上没有的字符提供输入方法。每个三元字符由 2 个问号和 1 个其他字符构成，例如，用“??=”代替数字符’#’，用“??！”表示竖线’|’等。

5.1.2 标识符与关键字

C 语言中使用的词分为六类：标识符，关键字，运算符，分隔符，常量和注释符。

1. 标识符

标识符是一系列由字母、数字和下划线组成的字符序列，它是对实体标识的一种定义符，用来标记用户定义的常量、变量、函数和数组等。定义标识符的规则如下：

（1） 只能由字母、数字和下划线构成；

（2） 第一个字符必须是字母或者下划线；

（3） 长度只有 31 个字符有效；

（4） 不能包含空格；

（5） 不能使用关键字。

以下标识符是合法的：

i, x3, name, my_car, sum5, _max

以下标识符是非法的：

3x， s*T， -3x， bowy-1， my car，main, include

在使用标识符时还必须注意以下几点：

（1） 标识符虽然可由程序员随意定义，但标识符是代表某个实体的符号，最好能够“见名知意”，便于阅读理解。

（2） 在标识符中，大小写是有区别的。例如，Max 和 max 是两个不同的标识符。

（3） 标准 C 不限制标识符的长度，但是只有前 31 个字符是有效的。 此外，标识符长度受 C 语言编译系统的限制。例如，在某版本 C 编译器规定标识符前八位有效，当两个标识符前八位字符相同时，则被认为是同一个标识符。

（4） 用户子定义的标识符最好不使用系统定义的标识符，如系统提供的库函数 printf、main、sqrt 等，以及预处理指令中涉及 include 和 define 等。虽然允许将这些标识符定义新的意义，但会使其失去原来的作用，从而产生歧义。

2. 关键字

C 语言的关键字是具有特定意义的字符串，也称为保留字。标准 C 包含 32 个关键字，见表 5.2，可分为以下几类：

（1） 数据类型说明符（12 个）

用于表示变量、函数、数组或等实体的类型。如 int、double、struct 等。

（2） 流程控制说明符（12 个）

用于控制程序流程的跳转和改变。如 if else 是条件语句的定义符，do whle 和 for 循环语句的定义符，函数中 return 实现返回。

（3） 存储类型说明符（4 个）

常用于说明变量的存储类型，包括 auto、 extern、 register 和 static。

剩余 4 个为其他类型的关键字，包括 const，sizeof，typedef，volatile。C99 增添了 5 个关键字，包括_Bool 、_Complex、_Imaginary、inline 和 restrict。

所有关键字都有固定的含义，不能改变其含义，而且必须是小写。注意，用户定义的标识符不能与关键字相同。

表 5.2 标准 C 中的关键字

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				

5.1.3 可执行语句

C 语言中有效字符构成标识符或关键字，这些字按一定规则连接成语句，语句是构成程序的基本模块。C 程序中的语句必须以分号结束，一条语句可以写在一行或者多行。常用的语句分为以下几类：

1. 声明语句

声明语句用来规定一组标识符的解释或属性，如声明变量、数组或函数。

2. 表达式语句

C 语言提供丰富而强大的运算符，由运算符和操作数可组成表达式语句。

3．程序控制语句

程序中可以一条条语句依次执行，也可用选择语句、循环语句及跳转语句改变程序流程，这类语句称为程序控制语句。

4．复合语句

有时将相关语句组成一个语句块，用大括号{ }将其括起来，这样的语句称为复合语句。

5．函数调用语句

调用系统定义的库函数或者用户自定义的语句。

初学者经常忘记语句后的分号，必须引起注意。

例 5.1 输入某人的年龄，计算出生年并判断是否成年。实现该程序并辨析代码中包含的各种语句。

```
#include <stdio.h>          /*预处理指令*/
int main()                  /*主函数*/
{
   int age, year;           /*变量声明语句*/
   scanf("%d",&age);   /*  函数调用语句，输入年龄*/
   year = 2010 - age;       /*表达式语句*/
   printf("出生年：%d\n ", year);   /*  函数调用语句*/
   if ( age<18)             /*  程序控制语句*/
   {
      printf( "未成年!\n");
      printf( "还有%d 才成年!\n", 18-age);
   } /*  复合语句*/
   return 0;          /*  程序控制语句*/
}
```

5.2 数据类型

5.2.1 理解数据类型

当时可品尝餐桌上一道道美味佳肴时，可能并没有想过制作这些食品复杂的过程。厨师将各种烹饪原料通过不同的烹调技法、按照特定的步骤进行加工，为了传承和发扬苦心研究出来的菜肴制作方法，将这些过程以菜谱的形式记录下来。其实，计算机执行程序和厨师按菜谱做饭的的过程是相似的。程序按照某种特定的方法和步骤处理各种各样的数据，完成某个任务或得出某个结果。程序中的数据就如菜谱中的烹饪原料，程序中的算法就如菜谱中的烧菜方法。选择和搭配食材是一道菜肴是否可口的前提，厨师针对这些原料实施不同的加工的法。同样，程序设计者首先要确定表示和存储数据的方法，在此基础上安排处理数据的步骤，使程序完成特定功能。

C 语言中常用变量和常量存储和表示数据。常量是在程序运行过程不改变的数据，如计算圆周用的 3.14，重力加速度取 9.8。程序运行过程中某些数据的值可能发生改变，此类数据称为变量。

俗语道“人以类聚，物与群分”，具有相似特性的事物一般归为同类。就烹饪原料而言，鱼、虾和贝都为海鲜类，芸豆、白菜和青椒都为蔬菜类，不同种类的食材有相似的保存和加工方法，厨师根据该类食材的特性烹制菜肴。在程序中可能要涉及很多数据，它们的值可为小数、整数和字符等，通过数据类型将其进行归类，同一类型的数据有相似的表示形式和存储方式，程序员根据数据的类型安排操作的步骤和方法。常量和变量是有类型的数据。

在C语言中数据类型可分为：基本数据类型，构造数据类型，指针类型，空类型四大类。常用数据类型见图 5.1。

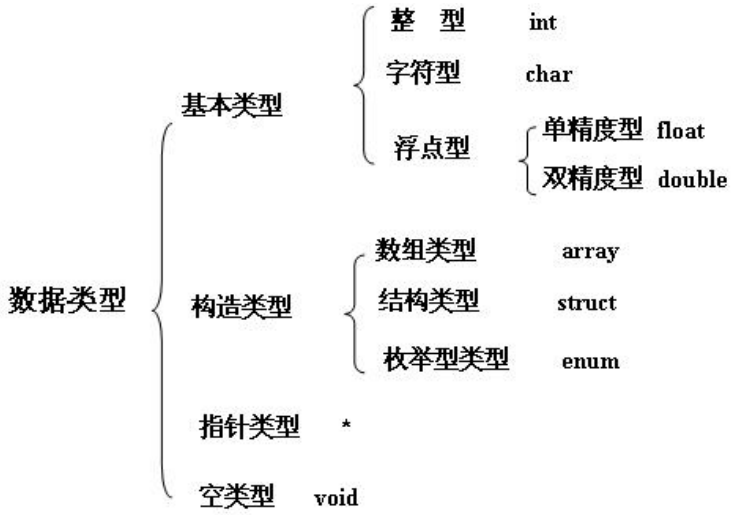

图 5.1 C 语言常用数据类型

很多编译器还支持 long int 和 long double 等扩展类型。C99 增添了一些数据类型：表示逻辑值的数据类型_Bool、用于复数运算的_Complex 和_Imaginary 类型以及用于整数操作的 long long 类型。本章讨论基本数据类型整型(int)、字符型(char)和浮点型（float、double），其他的类型在后面章节中逐一介绍。

5.2.2 变量

1．变量的定义

变量用于存储程序中数据和运算结果，变量三个要素为：类型、名字和值。每个变量有名字和类型，编译器会在内存中开辟相应的空间存放变量的值。使用变量前要先进行定义，变量定义的语法形式为：

类型 变量名列表； /*注释*/

例如，

```
int age;     /*年龄变量*/
float height , weight ; /*  身高和体重*/
```

定义变量应注意以下几点：

（1）变量命名规范化：每个变量由变量名惟一标识，变量名应符合标识符的命名规则，程序中命名风格应保持一致。

（2）变量类型合理化：变量的类型决定变量值的存储和操作方式，应正确选择其类型。

（3）同一语句中不能混合定义不同类型的变量。当定义多个变量时，若它们类型相同，可以在同一语句中定义，并用逗号隔。若类型不同，则必须在不同的语句中分别定义。例如，错误定义形式

```
int age, float height ;
```

（4） 应该对变量进行合理的注释，增加代码的可读性。

2．变量值的存储

定义变量后可对其进行赋值，变量的值也可在定义时获得，这称为变量的初始化，例如

```
int counter = 0;              /*计数器变量*/
double pi = 3.1415926 ,   /*  圆周率*/
```

g = 9.80 ;　　/* 重力加速度*/

这些数据存放在变量所对应的内存空间里，内部存储器（简称内存）是存放数据和代码的硬件，其作用好比存放烹饪原料的冰箱。为了提高存储空间的利用率，最好将冰箱分成小格子和大格子，厨师取放食物时候要记住排放在哪个格子中，以便下次再取放。将存储大量数据的内存分为若干单元，按照数据的类型分配不同大小的存储单元。

计算机中存储的数据都为二进制形式，例如用 1010 代表整数 10。二进制数据最小的单位为位，每一个二进制数据 1 或 0 的存储单位为 1 比特(bit)。将 8 个比特为组合起来构成字节(Byte)，例如 01111111 可以存储在 1 个字节中，表示 10 进制数值 127。若将多个字节组合起来，可以构成更大的单位字(word)，对于 16 位的编译器而言，1 个字包括 2 个字节；而对于目前常用的 32 位编译器而言，1 个字包括 4 个字节；那么如果你使用 64 位的计算机，一个字又该是多少字节呢？

变量代表内存中具有特定属性的一个存储单元，定义变量时编译器为其分配在内存的位置，并根据变量的类型决定变其值占用几个字节。通常用变量名字访问该变量，读取或者更改变量的值。编译器会识别变量对应的内存位置，而不必劳烦程序员记住到底将数据存储在那个字节中。例如有变量 int a; double b;它们在内存中的情况见图 5.2。

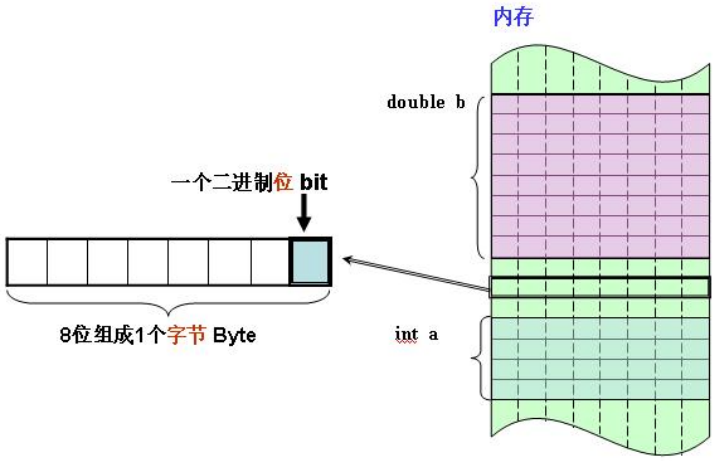

图 5.2 变量在内存中的存储

5.2.3 常量

在程序执行过程中，常量的值始终不改变。按照不同的表示形式，将常量分为直接常量和符号常量。

直接常量也称字面常量，如整数 1998、0、-3；小数 3.14、-1.23；字符 ‘a’、‘+’和‘0’等。通常这些常量出现在表达式中和变量一起参与运算。和变量一样直接常量分为不同的类型，但是它们不存储在内存中。

在 C 语言中，可以用一个标识符来表示一个常量，称之为符号常量。符号常量在使用之前必须先定义，其一般形式为：

#define 标识符 常量

其中#define 也是一条预处理命令（预处理命令都以"#"开头），称为宏定义命令，功能是将该标识符定义为其后的常量值。编译器将程序中所有出现该标识符的地方均用该常量值代替。习惯上符号常量的标识符用大写字母，变量标识符用小写字母，以示两者的区别。

例 5.2　定义符号常量

```
#include<stdio.h>
```

```
#define BYTE 8                  /*定义每字节比特数*/
int main()
{
   int bytesPerWord = 4,       /*每字包含字节数*/
       totalBit ;              /*总比特数*/
   totalBit = bytesPerWord* BYTE;       /*计算一个字包含比特数*/
   printf("%d bits per word\n",totalBit);   /*  用%d 的格式输出整数*/
   return 0;
}
```

使用符号常量的方法和变量类似，区别又两点：符号常量没有类型而言，其值不能改变（即不能再被赋值）。下面为使用符号常量的错误语句

```
#define float PI 3.14;   /*  不能指定类型，没有分号  */
PI = 3.1415926;          /*  不能给符号常量赋*/
```

定义符号常量有不仅可以增加程序的可读性，而且便于代码的维护。用符号常量代替直接常量，可以使代码的语意明确，例如在 5.2 中 BYTE 代表每字节包含 8 个 bit。假设定义符号常量圆周率 PI 为 3.14，并程序中多次引用该符号，若程序中的 PI 值修改为 3.1415926，只需更改预处理指令即可，减少逐个修改直接常量的繁琐劳动，并避免错误修改。

5.2.4 整型数据

1.整型常量

整型常量是由数字构成的常整数。在C语言常用八进制、十进制和十六进制三种整数，它们基数分别为 8、10 和 16。

（1）十进制整常数

十进制由数字 0～9 组成，前面可以加＋或者—区分符号。以下为合法的十进制整常数：

123、-321、0、65535、+1627；

十进制常数是没有前缀，并且不能出现数字以外的字符，以下各数不是合法形式：

023 (不能有前缀 0)、23D (含有非十进制数码)。

（2）八进制整常数

八进制常整数必须以前缀 0 开头，数字取值为 0～7。八进制数通常是无符号数。以下各数是合法的八进制数：

015(十进制为 13)、0102(十进制为 66)、0177777(十进制为 65535)

将八进制常量 0102 转换为 10 进制值，计算过程为 $0102=1\times8^2+0\times8^1+2\times8^0=66$。

以下各数不是合法的八进制数：

256(无前缀 0)、03A2(包含了非八进制数码)、-0127(出现了负号)。

（3）十六进制整常数

十六进制整常数的前缀为 0X 或 0x，其数码包括 16 个：数字为 0~9，以及字母 A~F 或 a~f，其中 A～F（a~f）分别表示 10～15。以下各数是合法的十六进制整常数：

0X2A(十进制为 42)、0XA0 (十进制为 160)、0XFFFF (十进制为 65535)；

以下各数不是合法的十六进制整常数：

5A (无前缀 0X)、0X3H (含有非十六进制数码)。

在程序中根据前缀来区分各种进制数，在书写常数时不要把前缀弄错造成结果不正确。对于整型常数可以添加后缀（u、U 、l 、L），表示无符号和长整数。

2．整型变量

整数变量一般用关键字 int 说明，其数值的取值范围取决于特定的计算机。通常整数占

用一个字的存储空间，由于不同的计算机字长和编译器的位数不同（一般为 16 位或 32 位），因此所能保存的最大和最小整数与计算机相关。如果是 16 位的字长，整数的取值范围为-32768～32767（-2^{15}~（2^{15}-1）），有符号数用 1 位代表符号，用 15 位表示数据。而对于 32 位的字长，整数的取值范围大约为-2^{31}~（2^{31}-1）。

为了控制整数的范围和存储空间，C 语言定义了三种整数类型：short int、int 、long int。

（1） 基本类型：一般整数默认为 int 类型 ，整型变量占 4 个字节（在 32 位机中）。

（2） 短整量：类型说明符为 short int 或 short，整型变量占 2 个字节（在 32 位机中）。

（3） 长整型：类型说明符为 long int 或 long，一般在内存中占 4 个字节。

为了扩大数据的表示范围，也可以将整数声明为无符号型，类型说明符为 unsigned。各种无符号类型量所占的内存空间字节数与相应的有符号类型量相同。但由于省去了符号位，不能表示负数，因此将正数的范围扩大了一倍。如 unsigned int 整型变量值的范围为 0～2^{16}-1，unsigned int 整型变量值的范围为 0～2^{32}-1。表 5.3 和 5.4 分别列出了 Turbo C 和 Visual C++6.0 中各类整型量所分配的内存字节数及数的表示范围。

表 5.3 整型数据的取值范围 (16 位字长)

类型说明符	数的范围		字节数
short int	-32768~32767	即-2^{15}~（2^{15}-1）	2
unsigned short int	0~65535	即 0~（2^{16}-1）	2
int	-32768~32767	即-2^{15}~（2^{15}-1）	2
unsigned int	0~65535	即 0~（2^{16}-1）	2
long int	-2147483648~2147483647	即-2^{31}~（2^{31}-1）	4
unsigned long	0~4294967295	即 0~（2^{32}-1）	4

如果厨师想盛放某种食物，要根据食物的形态和大小选择容器。例如，用小盒子装 1 打鸡蛋放不进去，选择大盒子放 1 个鸡蛋太浪费；又如，瓶子可以盛放油盐酱醋，但不适合盛放鸡蛋。同理，程序员应根据数据值的范围选择合适的数据类型。

表 5.4 整型数据的取值范围 (32 位字长)

类型说明符	数的范围		字节数
short int	-32768~32767	即-2^{15}~（2^{15}-1）	2
unsigned short int	0~65535	即 0~（2^{16}-1）	2
int	-2147483648~2147483647	即-2^{31}~（2^{31}-1）	4
unsigned int	0~4294967295	即 0~（2^{32}-1）	4
long int	-2147483648~2147483647	即-2^{31}~（2^{31}-1）	4
unsigned long	0~4294967295	即 0~（2^{32}-1）	4

例 5.3 输入一个整数值，计算并输出其平方值，验证整型数据的溢出情况

程序 5.3 的功能是计算平方数，当选择基本类型的变量 square_int 存储 1200 的平方数时，可以输出正确的结果，而选择短整型变量 square_short 存储平方数时，发现得到不同的结果。请思考，为什么会产生这种奇怪的现象？

```
#include<stdio.h>
#include<math.h>
int main()
{
```

```
    int num =1200, square_int = num *num;
    short int square_short = square_int;
    printf("square_int = %d\n",square_int);
    printf("square_short = %d\n",square_short);
    scanf("%d",&num);           /*  输入整数*/
    square_int = num *num;    /*  计算平方*/
    printf("square = %d\n",square_int );
    return 0;
}
```

运行结果：

```
square_int = 1440000
square_short = -1792
120000
square = 1515098112
```

仔细分析程序，整型变量 num 的平方数超过了短整型的数据表示范围，即 square_short 存储不下这么大的数据，产生了数据的溢出，为了避免溢出可选用 int 或 long 类型的变量存储。当对变量 num 输入 120000 时，将存储 num 的平方数存储在 square_int 中也会溢出。请读者实验，当输入 num 的值满足什么范围时，能输出正确的结果。今后当程序输出了匪夷所思的结果时，检查是否选择了合适的数据类型，想想是否会发生溢出。

5.2.5 浮点型数据

1. 浮点数的类型

浮点类型也称实型，是用来描述小数的数据类型。包括单精度（float 型）、双精度（double 型）和扩展的双精度（long double 型）三类。

标准 C 并未规定每种类型数据的长度、精度和数值范围。在一般 C 编译系统中，单精度型占 4 个字节（32 位）内存空间，双精度型占 8 个字节（64 位）内存空间。对于扩展的双精度型（long double），不同系统有所差别，有的分配 8 个字节，有的 16 个字节，也有分配 80 位的。由于占用空间的差异，三种实型表示数据的能力是依次递增的，即单精度的数据范围和精度和都低于双精度型。浮点数表示的取值范围见表 5.5。

表 5.5　浮点数表示的取值范围

类型说明符	字节数	精度	数的范围
float	4	6~7	1.2×10^{-38}~3.4×10^{38}
double	8	15~16	2.2×10^{-308}~1.8×10^{308}
long double	16	18~19	3.3×10^{-4932}~1.2×10^{4932}

浮点型数据与整数的存储方式不同，比整数具有更大的数据范围。要注意其精度有限，float 型只能提供 6 位有效数字，double 型只能保证 15 位有效数字，因此小数有时不能被精确表示。

2.浮点型常量

浮点型常量也称为实数或浮点数。在 C 语言中，实数用十进制表示，它有二种形式：

（1）　小数形式：

由十进制数码 0~ 9、小数点和正负号组成。例如：

0.0、25.0、5.789、0.13、5.、.300、-267.8230、- .1

均为合法的浮点数。注意，若小数点前面或者后面的数为 0，可以省略一个 0，但不能呢同时省略，而且必须有小数点。

（2） 指数形式：

由十进制数 0~ 9、阶码标志“e”或“E”以及阶码组成。其一般形式为：

a E ^{n}（a 为十进制实数，n 为十进制整数）

其中，a 为小数形式，E 表示以 10 为底数，n 为指数（只能为整数，可以带符号）。该形式所表示的值为 a*10^{n}。例如：

2.1E5 (等于 2.1*10^{5})、3.7E-2 (等于 3.7*10^{-2})、-2.8E-2 (等于-2.8*10^{-2})

浮点数必须包含一个小数点或者指数，或者两者都包含。以下不是合法的浮点数：

345 (无小数点)、E7 (阶码标志 E 之前无数字)、-5 (无阶码标志)、2.7E (无阶码)

指数形式相当于科学计数法的表示形式，但要注意底数 E 为 10 而不是数学中的常量 e。

标准 C 允许浮点数使用后缀（f、F、l、L）。后缀为“f”或“F”即表示该数为 float 型浮点数，后缀为“l”或“L”即表示该数为 long double 型浮点数，没有后缀的实数默认为 double 型浮点数。例如，下面为不同类型的浮点数：

3.14159、3.14159F、3.1415926L

3. 浮点型变量

浮点型变量与整型的定义规则相同，但需要用 float、double 或 long double 声明其类型，例如：

```
double a, b, c = 8.9;
float f1 = 0.123456f,
      f2 = -789.012F,
      f3 = -123.456E-2F;
```

浮点型变量的初始化和引用形式与整型变量也相似。但要注意，两者在一些操作中存在不同：调用库函数 printf 时用“%f”格式输出浮点数，调用库函数 scanf 时用“%f”和“%lf”分别对 foat 型和 double 型数据进行输入。例如，对于上面定义的 f1，f2 和 f3 按下面的方式操作：

```
printf("f1 = %f, f2 = %f，f3 = %f\n", f1, f2,f3);
printf("f1 = %d, f2 = %d, f3 = %d\n", f1, f2,f3);
```

输出结果为：

```
f1 = 0.123456,   f2 = -789.012024,   f3 = -1.234560
f1 = 0,   f2 = 1069521616,   f3 = -1610612736
```

可见，如果依然按“%d”的格式对 float 型的数据进行解释，会产生错误输出结果。同样，对 double 型变量 a, b, c 按如下形式调用 sancf：

```
scanf("%lf ",&a);
scanf("%f",&b);
scanf("%d",&c);
```

分别输入 1.23 、4.56 和 7.89 后执行 printf("a=%f\nb=%f\nc=%f\n",a,b,c);

输出更为奇怪的结果：

```
a=1.230000
b=-92559604486586627000000000000000000000000000000000000000000000000000000000.000000
c=-92559592117432074000000000000000000000000000000000000000000000000000000000.000000
```

注意，double 型的数据必须用“%lf”格式输入，才能得到正确的结果。企图将单精度型的数据存储存储在双精度的空间中，初学者经常犯这样的错误。

细心的读者应该会发现前面定义 float f2 = -789.012F，按“%f”输出 f2 = -789.012024。这一结果和 f2 的初值不完全相同。前面提到过，float 型只能保证 6 位数据有效，而%f 默认输

出小数点后 6 位数字，后面的数据是没有意义的。

由于浮点型变量是由有限的存储单元组成的，因此能提供的有效数字总是有限的，浮点数有时会产生舍入误差。若 f2 = -789.0126f，则 f2 实际获得的值为-789.013。

例 5.4 验证浮点数的舍入误差

```
#include<stdio.h>
int main()
{
    float f;
    double d;
    f = 0.123456789f;                    /* f 实际值为 0.123457*/
    d = 123456789.123456789;
    printf("%f=f ,d= %f\n", f, d);
     return 0;
}
```

运行结果：

```
f=0.123457, d= 123456789.123457
```

5.2.6 字符型数据

1.字符常量

字符常量是用单引号括起来的一个字符。如'a'、'b'、'='、'+'、'?'都是合法字符常量。在 C 语言中，字符常量有以下特点：

（1）字符常量只能用单引号括起来，不能用双引号或其它括号；

（2）字符常量只能是单个字符，不能是字符串；

（3）字符常量可以是字符集中任意字符，还包括转义字符。

转义字符是一种特殊的字符常量，以反斜线"\"开头，后跟一个或几个字符。不同于字符原有的意义，转义字符具有特定的含义，故称“转义”字符。例如，在前面各例题 printf 函数的格式串中用到的“\n”就是一个转义字符，其意义是“回车换行”。转义字符主要用来表示那些用一般字符不便表示的控制代码，常用的转义字符见表 5.6。各种转义符的含义请读者自己通过实验理解。

表 5.6 常用的转义字符及其含义

转义字符	转义字符的意义	ASCII 代码
\n	回车换行	10
\t	横向跳到下一制表位置	9
\b	退格	8
\r	回车	13
\f	走纸换页	12
\\	反斜线符"\"	92
\'	单引号符	39
\”	双引号符	34
\a	鸣铃	7

计算机通常使用某种编码形式来表示字符，ASCII 码(美国国家信息交换标准字符码）为常用的西文字符编码。每个字符在 ASCII 码表（见附录 A）中对应一个码值，例如，‘n’的 ASCII 值为十进制数 110，转义符‘\n’ 的 ASCII 值为十进制数 10（八进制数 12）。

广义地讲，C 语言字符集中的任何一个字符均可用转义字符来表示。用\ddd 的形式表示反斜杠后为 1～3 位八进制数所代表的字符数，用\xhh 的形式表示反斜杠后为 1～2 位十六进制数所代表的字符数，ddd 和 hh 分别为八进制和十六进制的 ASCII 代码。如\101 表示字母'A' ，\134 表示反斜线，\XOA 表示换行等。

2. 字符变量

字符变量用来存储字符常量，类型说明符是 char，定义形式为：

char c1, c2 = 'A' ;

每个字符变量被分配一个字节的内存空间，因此只能存放一个字符。字符值是以 ASCII 码的形式存放在变量的内存单元之中的。如字符'A'的 ASCII 码为 65，字符'a'的 ASCII 码为 97。若'x'的十进制 ASCII 码是 120，'y'的十进制 ASCII 码是 121。对字符变量 c1，c2 赋'值：

c1 = 'x'; c2='y';

实际上 c1 和 c2 两个单元内存放的为 120 和 121 的二进制代码：

C1：

0	1	1	1	1	0	0	0

C2：

0	1	1	1	1	0	0	1

因此也可以把字符变量看成是整型量。C 语言允许对整型变量赋以字符值，也允许对字符变量赋以整型值。在输出时，允许把字符变量按整型量输出，也允许把整型量按字符量输出。

例 5.5　定义并输出字符变量和常量

```
#include<stdio.h>
int main()
{
   char c1, c2;
     c1 = 'x';
     c2 = 121;
     printf("%c %c\n", c1,c2);          /*以字符形式输出：x y*/
     printf("%d %d\n",c1,c2);          /*为整数形式输出：120 121 */
     printf("%c %c\n",c1-32, c2-32); /*  输出大写字符：  X Y */
     printf("%d %d\n",c1-32, c2-32); /*  输出大写字符：  88 89 */
     return 0;
}
```

本程序中定义 c1 和 c2 为字符型，可以直接用字符常量赋值，也可以赋以字符 ASCII 码对应的整型值。从结果看，c1 和 c2 的输出形式取决于 printf 函数格式串中的格式符，当格式符为"%c"时，对应输出的变量值为字符，当格式符为"%d"时，对应输出的变量值为整数。

此外，C 语言允许字符变量参与数值运算，即用字符的 ASCII 码参与运算。注意'3'和 3 的意义和值都不同，'3'+3 的值为字符'6'对应的 ASCII 码值。由于大小写字母的 ASCII 码相差 32，因此可以通过运算'A'+32 把大写字母'A'换成小写字母'a'。

虽然整型数据和字符数据可以一起参与运算，但两者不能完全等同。有些系统将字符定义为 unsigned char 型，字符变量的数值范围为 0～255。Visual C++中将字符型定义成有符号型，char 型数据的数据范围一般为-128～127，即字符变量中存放一个 ASCII 码值为 0～127 的字符。如果超出此范围，不能用%c 输出正常的字符。

3. 字符串常量

字符串是和字符常量不同的数据类型，在用双引号引起的多个字符为字符串常量，例如

"Hello World" ，"你好！"

对于’A’和”A”，存储它们的机制不同，前者可以用字符变量存储，在内存中只占 1 个字节，而存储后者则需要多个字节。在 printf 中表示格式的参数为字符串，如"c = %c ,i = %d\n"，在屏幕上输出一个字符串。注意字符串中除了包含可见的字符外，还自动在串尾添加一个特殊的字符’\0’，作为字符串的终止标志，因此字符串”A”中实际包含两个字符。

5.3 输入与输出操作

5.3.1 输入与输出的概念

程序对数据进行处理一般分为三个步骤：获取数据、计算和输出结果。计算机从外部获取数据为输入操作，即将数据从键盘、磁盘文件及网络等设备读取到内存中。程序对这些数据按二进制的方式进行表示和处理，并将结果以人类能直观理解的方式呈现在显示器上，或者存储在磁盘、优盘等外部设备中，数据从内存流向外部设备为输出操作。可见，所谓输入输出是相对计算机内存而言的。

在 C 语言中，数据输入输出操作通过调用库函数实现。前面的程序中常使用输入函数 scanf 和输出函数 printf，程序员无需深入了解数据怎样流动与转化，就可以轻松进行输入输出操作。程序中要用预编译命令#include 将头文件“stdio.h”包括到源文件中，stdio 为 standard input &outupt 的缩写，编译后头文件中的内容就成为源代码的一部分了。由于 printf 和 scanf 函数使用频繁，系统允许在使用这两个函数时可不加

#include< stdio.h > 或 #include ”stdio.h”

此外，标准库函数还提供了其他 I/O 函数，按格式化的方法读写数据。本章将讨论常用的标准输入输出函数的基本用法，复杂格式的读写操作请参阅其他资料，了解文件读写操作请阅读第 12 章。

5.3.2 格式化输出函数

printf 函数称为格式输出函数，其关键字最末一个字母 f 即为“格式”(format)之意。其功能是按用户指定的格式，将相关数据输出到显示器屏幕上。

1.printf 的一般形式

前面的例题中已了解该函数的基本用法，调用的一般形式为：

printf(“格式控制字符串”， arg1,arg2,…, argN);

其中圆括号的参数中包括两部分：格式控制字符串和输出列表 arg1,arg2,…, argN。控制字符串用于指定参数列表中各个表达式的输出格式，由格式字符串和非格式字符串组成。格式字符串由两种类型项构成：

（1） 格式说明符

以%开头的字符串，用于定义每个数据的输出格式。在%后面跟有各种格式字符，以说明数据的类型和形式。如：“%d”表示按十进制整型输出，“%f”表示按实型输出，“%c”表示按字符型输出等。

（2） 普通字符

用于显示在屏幕上的非格式字符串，。这些字符在显示中起提示作用，以字符的形式输出，并包括一些转义字符。例如，

```
int i = 1 ;
double d = 3.14 ;
printf(“i = %f, d = %f\n” ， i, d);
```

输出结果为

i = 1 , d = 3.14(回车换行)

若参数列表中若包含多项参数，则用逗号分隔。其中参数可以为变量、常量、函数调用以及其他表达式。必须注意，格式字符串和各输出项在数量、类型和顺序上必须一一对应。

例 5.6 使用标准输出函数

```
int main()
{
   int hour = 23, minute = 45;                    /* 定义变量，小时和分钟*/
   double time = hour + minute / 60.0 ;       /* 转化成小数时间*/

   printf("%d hours %d minutes\t", hour, minute);
   printf(":%f\n" , time);
   /*等效于 printf(" : %f\n" , hour + minute / 60.0); */
   return 0;
}
```

运行结果：

23 hours 45 minutes　　: 23.750000(回车换行)

本例中首先输出了整数小时和分钟，将变量 hour 和 minute 的值按十进制整数形式输出。然后将其转化成小数的时间存储在实型变量 time 中，按%f 的格式输出该变量和直接输出表达式 hour + minute / 60.0 的结果相同，若执行语句 printf(" : %f\n" , time);则输出错误的结果 0。

除了这些基本的输出操作，printf 语句提供某些格式字符串，能有效的控制数据显示的对齐方式、长度以及小数位等。具体说明见表 5.7。下面分别讨论整数和实数的格式化输出形式。

表 5.7 输出格式字符及其含义

格式字符	意　　义
d	以十进制形式输出带符号整数(正数不输出符号)
o	以八进制形式输出无符号整数(不输出前缀 0)
x,X	以十六进制形式输出无符号整数(不输出前缀 Ox)
u	以十进制形式输出无符号整数
f	以小数形式输出单、双精度实数
e,E	以指数形式输出单、双精度实数
g,G	按数据精度以%f 或%e 中较短的输出宽度输出单、双精度实数
c	输出单个字符
s	输出字符串

2. 整数的格式化输出

用以显示整数的格式说明符为：

%[对齐方式][输出最小宽度]整数类型

[]为可选项，各项具体含义如下：

（1） 输出最小宽度：%nd

用以显示该小数的最小位数，若数据实际位数多于定义的宽度 n，则按实际位数输出；若实际位数少于定义的宽度 n 则补以空格或 0。

（2） 对齐方式：%d 或%-d

一般默认数据右对齐，若输出宽度前加负号，则设置为左对齐。

（3） 整数类型：%d 、%o、 %x%X 或%ld

整数可以表示成不同进制的数据。内存中的数据都存储为二进制的形式，可以将其转化为十进制形式输出，用%d 格式表示；可以用%o 的格式按照八进制输出无符号的整数，还可以用%x 或者%X 输出十六进制形式的无符号整数。l 表示长型量输出，如%ld 输出长整型数。

例 5.7 格式化输出整数

```
int main()
{
    int   hour = 1234, minute = 56;
    printf("%d,%d\n" , hour,minute);
    printf("%8d,%4d\n" , hour,minute);       /*右对齐，不足宽度补空格*/
    printf("%08d,%04d\n" , hour,minute);    /*右对齐，不足宽度补 0*/
    printf("%-8d,%-4d\n" , hour,minute);     /*左对齐，不足宽度补空格*/
    printf("%-8d,%-4d\n" , -hour,-minute);   /*负数符号占 1 位*/
    printf("%o,%o\n" , hour,minute);          /*输出八进制*/
    printf("%4x,%3X\n" , hour,minute);       /*输出八进制，右对齐*/
   return 0;
}
```

运行结果为

```
1234,56
    1234,   56
00001234,0056
1234      ,56
-1234     ,-56
2322,70
 4d2, 38
```

3.实数的格式化输出

用以显示实数的格式说明符为

%[输出最小宽度] [.精度]实数类型

其中方括号[]中的项为可选项，各项具体意义如下：

（1） 输出最小宽度：%nf

用以显示该小数的最小位数 n，包括整数、小数点及小数部分的总位数。一般默认数据右对齐，若输出宽度前加负号，则设置为左对齐。

（2） 精度：%n.mf

用于显示小数点后的整数 m，精度格式符以“.”开头后跟十进制整数。若实际位数大于所定义的精度数，则按四舍五入截去超过的部分。一般实数默认输出为 6 位小数。

（3） 实数类型

一般用格式 f 将 float 型和 double 型表达式以十进制小数形式输出，lf 表示按 long double 类型输出。用格式字符 e 或 E 将实型数据按科学计数法的形式输出。

例 5.8 格式化输出实数

```
int main()
{
   float   f = 3.1416;
   double d = 3.1415926;
```

```
    printf("%f    %10.2f    %-10.2f\n", f, f, f);        /*输出小数形式*/
    printf("%f    %10.7f    %-10.7f\n", d, d, d);

    f=.00000001f;
    d = 1234.56789;
    printf("%f    %e\n",f, f );                          /*输出指数形式*/
    printf("%e    %8.2e\n", d, d);
    return 0;
}
```

运行结果：

```
3.141600              3.14    3.14
3.141593     3.1415926    3.1415926
0.000000    1.000000e-008
1.234568e+003    1.23e+003
```

4. 提高输出的可读性

计算机的输出经常用于分析数据之间的某些联系，以供检验结果的正确性，或为决策提供依据，因此输出的正确性和清晰性尤为重要。以下一些方法可以增强输出的可读性，使其便于理解：

（1） 在数据之间插入相应的分隔符增加数据的间距，如空格和逗号；

（2） 在输出的部分数据之间输出回车换行\n，分隔多个数据；

（3） 设置数据的长度和精度，使多组数据对齐；

（4） 在输出中给出变量名，明确数据的含义。

（5） 在输出数据前输出字符串，用以显示提示信息。

总之，为了将结果正确清晰地输出显示，程序员应仔细考虑程序产生输出结果的形式。

5.3.3 格式化输入函数

scanf 函数称为格式输入函数，即按用户指定的格式从键盘上输入数据，将其值存储到相应的变量之中。scanf 函数的一般形式为：

```
scanf(“格式控制字符串”,&arg1, &arg2…,&argN );
```

其中，格式控制字符串的作用与 printf 函数相同，指定输入数据的个数、类型和样式。地址表列中给出各变量的地址，若有多个地址则用逗号将其分隔。

变量的地址表示该变量在内存中的位置，即变量值所存储的字节编号。变量的地址是在定义时由编译器自动分配的，一般表示为十六进制的整数，可用地址运算符“&”后跟变量名表示该变量的地址。注意，变量的地址和变量的值是不同的概念。例如

```
int i ;
scanf(“i =%d”, &i) ;                    /*输入 i =1*/
printf("i = %d, &i = %x", i, &i);   /*分别输出变量 i 的值和地址  i = 1, &i = 13ff7c*/
```

注意输入数据的格式和顺序必须和格式控制字符串严格对应，否则变量不能获得正确的值。

在运行输入函数程序时，将退出编译窗口进入运行窗口，屏幕上光标闪动等待用户输入，输入数据后程序继续执行后面的代码。若需要连续输入多个数值，一般用空格、回车符或制表符分隔数据。例如

```
double lentgh, width;
scanf( “%lf%lf” ,&lentgh, &width);
```

可以按如下格式输入数据：

1.2 3.4

或者

1.2

3.4

则将 1.2 存储到变量 lentgh 中，而 width 获得值 3.4。函数从键盘中读取数据时，将忽略分隔符。注意，不能其他字符分隔，除非在格式控制字符串中指定的特殊分隔符。例如

scanf(“%lf,%lf” , &lentgh, &width);

必须按如下格式输入数据：

1.2,3.4

变量 lentgh 和 width 才能获得正确的值。

如果错误地指定输入数据的格式，或输入数据的形式与控制字符串不匹配，则无法将从键盘输入的数值传递给变量。例如

```
int i; float f; double d;
scanf (“%d %f”, &i,&f,&d );        /*格式控制字符串与地址数目不匹配*/
scanf (“%d %f”, &f,&d );           /*格式控制字符串与变量类型不匹配*/
scanf (“d = %f”, &d);              /*格式控制符与变量类型不匹配*/
```

在程序运行中，若输入的格式和格式控制字符串不完全相同，有时会造成输入错误。例如

```
int i; float f; double d;
scanf (“%d %f”, &i,&f);       /*输入 3.14 5.6，  则 i=3，f=0.14 */
scanf (“%lf”,   &d);          /*输入 3，则 d = 3 .0*/
scanf (“%i = d”,&i);          /*输入 3，则 i 不能获得数值 3，应输入 i = 3*/
```

初学者使用 scanf 函数经常犯错，必须注意以下几点：

（1）scanf 中要求给出变量地址，若在格式控制字符串后直接给出变量名，则会在运行时出错。如 scanf("%d",a);是非法的。

（2）float 类型的数据可以用%f 进行输入或输入操作。对于 double 型的实数，可以用%f 的格式输出，但是必须用%lf 的格式输入数据。

（3）编译器在遇到分隔符或非法数据时即认为输入操作结束。例如

```
scanf(“%d %f %lf”, &i,&f,&d);
```

当错误输入 12A 6.7 时，A 即为非法数据，d 获得数值 12，而后面的变量将无法获得数据。

(4) 格式字符可以用空格分隔，但最好不用回车作为分隔符，除非后面还有数据要输入。例如

```
scanf(" %d %f ", &i, &f);      /*输入 3 4.5  或则 3（回车）4.5 均可*/
scanf(" %d\n%f ", &i, &f);     /*输入 3 4.5  或则 3（回车）4.5 均可*/
scanf("%lf\n",&d);             /*输入 3.14 ,光标闪动，等待继续输入*/
```

5.3.4 字符的输入与输出

1. 写字符

写入字符操作是从标准输入设备（通常指键盘）输入一个字符，将该值存储到字符变量中，完成该操作常使用以下两种方式：

（1） 调用格式化输入函数 scanf，使用格式字符“%c”表示输入数据的类型为字符。

（2） 调用非格式化输入函数 getchar，其功能是读字符。一般调用形式为：

```
char c = getchar( );
```

注意两种形式都可以接收任意字符，包括空格、制表符和回车。这意味着输入数值数据时若使用的以上分隔符，都将作为有效字符存储到字符变量中。例如，

```
int i   ;   float   f   ;     char c;
scanf("%d %f %c ", &i, &f, &c);
printf("%c %d", c ,c );
```

当输入 1(空格) 3.14（回车）后，直接输出回车和 10。原因在于 3.14 后面的回车不是分隔符，而是有效字符（ASCII 码值为 10）。当输入数值型数据后，想继续输入字符型数据，必须先接收前面遗留的分隔符，然后才能成功输入新的有效字符。例如

```
scanf("%d %f ", &i, &f);   /*输入 1 3.14（回车）*/
c = getchar();                  /*接收空格, c = '\n' */
c =getchar();                    /*输入字符 A*/
printf("%i   %f   %c", i, f,c);   /*输出 1   3.14   A*/
```

接收空格并输入字符还可以写成如下形式：

```
getchar();   c   = getchar();
```

或者

```
scanf("\n%c",&c);
```

因此，要特别注意 C 语言中混合数据的输入函数的用法。

2. 读字符

读取字符是将内存中某个字符变量的值传送到标准输出设备（通常为显示器），常通过调用以下两个函数完成输出操作：

（1） 格式化输出函数 printf，使用格式字符"%c"表示输出的数据为字符类型。

（2） 非格式化输出函数 putchar 函数，其功能是在显示器上输出单个字符。其一般形式为：

```
putchar(字符数据);
```

例如：

```
putchar('A');      /*输出大写字母 A*/
putchar(x);        /*输出字符变量 x 的值*/
putchar('\101'); /*也是输出字符 A*/
putchar('\n');    /*换行*/
```

使用字符输入函数和字符输出函数还应注意几个问题：

（1） getchar 函数只能接收单个字符，输入多于一个字符时，只接收第一个字符；

（2） 使用字符输入输出函数前必须包含文件"stdio.h"；

（3） 可以将两种函数配合使用，将输入输出函数的调用整合到同一语句中。例如，

```
char c = getchar(); putchar (c);
```

等效于

```
putchar(getchar());      /*输入字符后显示*/
```

5.4 编程错误

无论对于 C 语言的初学者，还是经验丰富的程序员，在设计和实现程序时经常会出现错误，程序很少第一次运行就正确。即使改正错误后运行成功，也并不能证明这是没有漏洞的完美程序。Murph 法则"可能出错的事情终将会出错"，非常适于计算机编程。实际上，程序中的错误是常见的，他们拥有一个专用的名字（bug），而纠正错误的过程成为调试（debug）。传说这个名字和计算机先驱 Grace Murray Hopper 诊断出的第一个硬件错误有关，而这个错误正是由计算机组件中一个昆虫引起的。

在编写程序时，经常会出现各种各样的错误，调式程序是编写一个完整程序的必要步骤。程序员在实现的程序功能的基础上，应尽量查找出所有的错误，并将其改正，保证程序正常

运行。一般将程序中的错误分为三类：语法错误、运行错误和逻辑错误。

例 5.9 阅读下面的代码，指出并改正其中的错误

编写程序，将各种容积单位进行转换，如将公升转换成加仑。

英国：1 加仑等于 4.5459711330833 公升

美国：1 加仑等于 3.78542686366028 公升

```
#include<stdio.h>
#define GALE 4.5459711       /*英制：1 加仑等于 4.5459711 公升 */
#define GALA 3.7854268       /*美制：1 加仑等于 3.7854268  公升*/
int main()
{
    int liter,          /*升*/
    float galE,       /*  英式加仑*/
          galA;       /*美式加仑*/
    printf("输入升");
    scanf("%d", liter)
    galE = liter * GALE;
    galA = liter * GALA；
    print(" %d 升＝%d 英制加仑\n",liter,galE);
    print(" %f 升＝%f 美制加仑\n",liter,galA);
    return 0;
}
```

5.4.1 语法错误和警告

由于程序某些代码不符合语法将产生语法错误。当代码违反了一条或者多条语法规则，在试图编译该程序时编译器能够自动识别出此类错误（error），程序员可以根据编译器提示的错误信息进行更改。此类错误必须予以改正，才能运行程序。

有时编译器还会给出警告信息（waring），提示程序中某行可能会出现问题，但程序可以运行。对于此类存在隐患的语句，最好将其改正。

例题 5.9 中存在很多语法错误，下面分别讨论：

错误 1：变量混合定义，在同一语句中定义不同类型的数据，出错代码段为：

```
int liter,       /*升*/
float galE,   /*  英式加仑*/
galA;       /*美式加仑*/
```

编译器提示语法错误信息：

```
error C2059: syntax error : 'type'（）
error C2146: syntax error : missing ';' before identifier 'galE'
error C2065: 'galE' : undeclared identifier
```

将变量 liter 后的逗号改成分号，在两个语句中分别定义 int 和 float 类型变量，消除三个错误。

错误 2：语句后丢失分号，语句不完整，出错代码为：

```
scanf("%d", liter)
galE = liter * GALE;
```

编译器提示语法错误信息：

```
error C2146: syntax error : missing ';' before identifier 'galE'
```

改正此类错时，若在被指出有错的一行中未发现错误，就需要查看上一行是否漏掉了分号。

错误 3：程序中包含非法字符，语句以中文分号“；”结束，出错语句为：

galE = liter * GALE；

编译器提示语法错误信息：

error C2018:unknown character '0xa3'

error C2018: unknown character '0xbb'

由于在编辑程序时键入非法字符，会导致许多不易察觉的错误。必须注意，除了注释中文字和双引号中字符，程序代码中不能包含中文字符。

错误 4：标识符书写错误，将 printf 写成 print，将 return 写成 Return，出错代码段为：

print(" %d 升＝%d 美制加仑\n",liter,galA);

Return 0;

编译器提示语法错误信息：

warning C4013: 'print' undefined; assuming extern returning int

error C2065: 'Return' : undeclared identifier

关键字一定不能拼写错，例如，不能将 main 写成 mian，编译器会给出 error 提示；对于标准库函数 printf，函数名拼写错误导致不能调用该函数，编译器提示 warning。虽然不将其改正也能够通过编译，但链接时提示以下错误：

5_9.obj : error LNK2001: unresolved external symbol _print

Debug/5_9.exe : fatal error LNK1120: 1 unresolved externals

因此对警告信息不能忽视，应分析原因消除 warning。

错误 5：数据类型不一致，可能会出现数据丢失，有问题的代码段为：

galE = liter * GALE;

galA = liter * GALA;

编译器提示警告信息：

warning C4244: '=' : conversion from 'double ' to 'float ', possible loss of data

将 GALE 替换成 4.5459711，参与运算的是 double 类型数据，因此表达式 liter * GALE 的值为 double 型数据。由于赋值符号的左边的变量 galE 为 float 类型，当左右边的表达式类型不同，编译器提示将高精度的数据赋给低精度的变量，可能会产生精度的丢失。尽管这类错误不影响程序的编译和执行，但最好消除此种警告，将变量的类型设置为 double。

当编译器检测错误时，会提示某处已出错并且给出可能的出错原因。但不幸的是，错误信息通常较难理解，有时还会误导用户，因此程序员需要通过不断实践获得更多经验，借助提示信息排除错误。

5.4.2 运行错误

运行错误是在程序运行时由计算机检测并显示的。当程序企图执行一个非法操作（如除数为 0）时会发生运行错误。当运行错误发生时，计算机会停止执行程序，显示诊断信息用于指示出错行。改正所有的语法错误后，运行例题 5.9 会出现的提示框。

错误 5：调用输入函数错误，第二个参数必须是变量的地址，出错代码段为：

scanf("%d", liter);

编译时不会产生语法错误的提示，但会造成程序无法运行。使用地址和指针操作时，经常会出现此类错误。

5.4.3 逻辑错误

当程序执行不正确的算法时，会产生错误的结果。程序没有按照设计者的意图执行，这类问题导致无法得到预期的结果，从而产生逻辑错误。此类错误通常不会引起运行错误，编

译器也不会提示出错信息，因此它是最难检测与调试的一类错误。

错误 6：调用输出函数时错误，格式字符串中数据的类型与变量不一致，出错代码段为：

```
print(" %d 升＝%d 英制加仑\n",liter,galE);
print(" %f 升＝%f 美制加仑\n",liter,galA);
```

按%f 的形式输出整型的公升数，或则用%d 的格式输出浮点数 galA，程序都将显示错误的结果。对于逻辑错误，计算机不会有任何提示信息，程序员需要彻底测试程序，可以借助调式工具和自己的编程经验，并将运行结果和正确结果比对，耐心的逐步排查错误。

对于初学者，编程中难免会出现各种各样的问题和错误，要静下心来仔细分析，并注意总结教训。建议读者在学习的过程中记录出现的各种错误，积累调试程序的经验，避免重复犯错。只有经历过无数次失败，程序才会最终成功运行。学习其实就是不断试错的过程，不要害怕和回避错误。通向程序高手的路遍布荆棘，真正的勇士敢于直面满是 bug 的程序并挑战所有难题。

习题

1．编写程序求三个整型数的和、积和平均值，输入三个数的值，输出结果。

2．编写程序求圆柱体底面周长、圆柱体的表面积和体体积。要求输入圆柱的底面直径和高，输出计算结果，取小数点后 2 位数字，输入输出时要求有文字说明。

3．编写程序实现华氏温度和摄氏温度的转换。输入一个华氏温度 F，要求输出摄氏温度 C。输出结果要有文字说明，取小数点后 4 位数字。转换公式为：

$$c=5\times(F-32)/9$$

4．输入一个小写字母，输出对应的大写字母。

5．将字符串译成密码，密码规律是用原来的字母后面的第 4 个字母代替原来的字母，例如，字母”A“后面 4 个字母是”E“，用”E“代替”A“。因此”china“应译为”Glmre“。编写一个程序，变量 C1、C2、C3、C4 和 C 的初始值分别为‘C’，‘h’，‘i’，‘n’，‘a’，输出经过加密运算后的密码。

6．设计一道习题，完成 unsigned int 与 int 的相加运算。

7．设计一道习题，验证数据溢出现象。例如，有 short s=-32768，输出 s=s-2 后 s 的结果;直接输出 s-2 的结果。

8．设计一道习题，验证浮点数有效位。

9．如果 5*7=23，按照这样的进制计算，请问 4*6 的值为什么？

10．有一个数值 152，它与十六进制数 6A 相等，那么该数值是（）进制数。

11． 对于 short 类型的十进制整数 23，内存中什么样？写出十进制整数 23 转换成的 8 进制数和 16 进制数。

12．分析‘1’、1 和”1”的区别，它们在内存中分别怎样存储？

第 6 章 运算符与表达式

6.1 概述

丰富的运算符和表达式使C程序简洁且功能完善，这也是C语言的主要特点之一，可以说C语言是一种表达式语言。

运算符是表示某种操作的符号，C语言使用这些运算符，可以对各种基本类型的数据进行操作。表达式由运算符和操作数构成，它是规定一些操作的式子。表达式和数学公式的形式类似，因此即使不懂的编程的人，也能看懂C程序中的大部分表达式。表达式中用运算符取代函数调用，这种符号化的语言使程序的可读性大大提高。

C语言的运算符按功能分为算术运算符、关系运算符、逻辑运算符等几类，详见表6.1。也可按运算符连接操作数的个数分为三类：

（1） 单目运算符

也称一元算符，只有一个操作数的运算符。包括负号（-）、正号（+）、自增(++)、自减(--)、非(!)、sizeof、指针运算符和部分位操作运算符。

（2） 双目运算

也称二元算符，连接两个操作数，大部分运算符属于此类。

（3） 三目运算符

连接三个操作数，C中唯一的三元运算符为条件运算符（？：）。

表达式按照一定的规则进行数值计算，每类运算符具有特定的优先级和结合性。操作数参与运算的先后顺序不仅要遵守运算符优先级别的规定，还受运算符结合性的制约，以便确定是自左向右进行运算还是自右向左进行运算。这种结合性是其它高级语言的运算符所没有的，因此也增加了C语言的复杂性。

表达式是由常量、变量、函数和运算符组合起来的式子。每个表达式有一个值及其类型，即计算结果的值和类型。单个的常量、变量、函数调用形式可以看作是表达式的特例，称为初等表达式。一般将位于运算符左边的操作数称为左操作数，而右边的为右操作数。

表 6.1 C 运算符及其分类

运算符种类	运算符用途	运算符
算术运算符	用于各类数值运算	加(+)、减(-)、乘(*)、除(/)、 求余(%)、自增(++)、自减(--)
关系运算符	用于比较运算	大于(>)、小于(<)、等于(==)、不等于(!=) 大于等于(>=)、小于等于(<=)
逻辑运算符	用于逻辑运算	与(&&)、或(||)、非(!)
位操作运算符	参与运算的量， 按二进制进行运算	位与(&)、位或(|)、位非(~)、位异或(^)、 左移(<<)、右移(>>)
赋值运算符	用于赋值运算	简单赋值(=)、复合算术赋值(+=、-=、*=、/=、%=)、 复合位运算赋值(&=,|=,^=,>>=,<<=)
条件运算符	用于比较运算	三目运算符（？:）
逗号运算符	用于分隔数据	逗号，
指针运算符	用于指针操作	取地址(&)、取内容(*)
求字节数运算符	计算数据所占内存 的字节数	字节数(sizeof)
其他运算符	特殊用途	有括号()，下标[]，成员(→，.)

6.2 算术运算

用算术运算符和括号将操作数连接起来的构成算术表达式，本节讨论的基本算术运算符包括：二元运算符（+、-、*、/和%）和一元运算符（+、-），它们用于实现各种基本类型数据的四则运算。比较复杂的算术运算符（++和--）将在 6.4 节中单独介绍。

1．整数运算

双目运算符（+、-、*、/）实现加减乘除运算。作为单目运算符的负号（-），其功能是将整数的符号取反。求余运算符（%）用于进行整数的求模操作，例如，表达式 11%4 的结果为 11/4 的余数 3。用于整数计算的运算符用法同数学表达式中的用法很相似，需要注意以下几点：

（1） 整数运算表达式的值为整型

整数运算总数得到整数结果。例如，表达式 11/4 的结果不是 2.75 而是 2，整数除法返回商的整数部分，结果舍去小数部分，不同编译器规定了不同的舍入方法。又如 1/3 的结果为 0，1/5*5 的结果不为 1，这种操作常无法得到精确的结果。

（2） 除法和求余操作的特殊性

除法运算中除数不能为零，若除法和求余计算符的右操作数为零，C 标准未定义其行为，一般不能进行除零的运算。此外，要求余数运算的操作数必须为整型。

（3） 运算符的优先级和结合性

在涉及几个不同的运算符的表达式中，运算符的执行顺序由优先级决定。按照“先乘除，后加减”的运算法则，双目运算符“+”和“- ”的优先级低于“*” 、“/” 和“%”, 但是单目运算符负号的优先级却高于乘除运算符。按照这样的规则，表达式-4*5+7.5/3.6-8%4 的求值顺序为（(-4)*5）+（7.5/3.6）-（8%4）。

当运算符具有相同的优先级，其执行顺序由此类运算符的相关性决定。具有“左结合”相关性的运算符，会首先和左操作数绑定。例如，四则运算符为左结合，表达式 4*5/3%4 的求值顺序为（（4*5）/3）%4。对于大部分运算符都是左结合，表达式是都从左向右依次计算的，只有几类为特殊的右结合。例如，负号为右结合运算符，表达式-5+x 中“-”和右操作数 5 关联。

表达式的计算顺序与运算符的优先级和结合性相关，有时可以用括号运算符“（）”来标明或者改变计算顺序，括号中的表达式总是先执行。例如，表达式-b/2*a 与-b/(2*a)具有不同的值。但是，标准 C 并没有规定表达式的求值顺序，计算表达式 4*5+7/3 时，先计算 4*5 还是先计算 7/3，这取决于编译器。尽管求值顺序对于这个表达式的结果没有影响，但是在某些情况下，不同求值顺序会造成不同的结果，稍后会看到这种情况。

2． 实数运算

对于小数运算，除了求余运算符，其他四则运算符（+、-、*、/）都可以用来参与运算，表达式结果为实型。例如，表达式 3.6/1.2 的结果为 2.0 而不是 2，1.0f/3.0f 的结果为 0.333333f。

3．字符运算

字符数据通常用某种方式进行编码，如单字节英文字符一般使用 ASCII 码，在许多大型机上的字符用 EBCDIC（扩展的二进制编码的十进制交换码）。无论使用哪种方式编码，每个字符有一个确定的整数值，因此可以认为字符类型的数据象整型数据一样，可以进行相关运算。如表达式‘a’+ 1 的值为字符‘b’的 ASCII 码值 98；又如表达式’D’-‘A’的值为 3，表示两个字符常量的码值之差为 3。

可以通过运算将一个字符转换成另一个字符，但是要保证表达式的结果在有效范围内，才能表示相应的字符。有符号的 char 型数据的字面值在-128 到 127 之间，观察 ASCII 码的

编码规律，一般 0～32 和 127 表示控制功能字符，可打印数据的码值在 31～126 之间。例如，

```
char letter = 65;       /*  存储 ASCII 码中的 A*/
letter = letter + 32;   /*  转换为 ASCII 码中的 a*/
printf( "%d %c\n", letter, letter);   /*输出码值 97 和代表的字符  a*/
printf( "%d %c\n", letter + 100, letter + 100);   /*输出 197   ?*/
printf( "%d %c\n", letter - 100, letter - 100);     /*输出  -3   ?*/
```

4. 数值函数

表达式中常出现函数调用的形式。标准库头文件<math.c>中定义了许多和数学运算相关的函数，如求绝对值函数 abs()、求平方根函数 sqrt()和三角函数 sin()、cos()和 atan()等，常见数值函数见表 6.2。

表 6.2　<math.h>中常用的数值函数

函数	结果说明
floor(x)	返回不大于 x 的最大整数
fabs(x)	返回 x 的绝对值
exp(x)	返回 e 的 x 次幂 e^x
log(x)	返回 x 的自然对数（以 e 为底）
log10(x)	返回 x 的对数（以 10 为底）
pwe(x,y)	返回 x 的 y 次幂 x^y
sqrt(x)	返回 x 的平方根
sin (x)	弧度 x 的正弦值
cos (x)	弧度 x 的余弦值
tan（x）	弧度 x 的正切值

例 6.1 编程实现角度与弧度的转换，并计算三角函数的值。由用户输入角度，输出该角度值对应的弧度和正余弦值。

```
#include<stdio.h>
#include<math.h>
#define PI 3.14159
int main()
{
  double angle, radian,     /*角度和弧度*/
         sine,cosine;     /*正弦和余弦*/
  printf("Input angle: ");
  scanf("%lf",&angle);
  radian = PI*angle/180;
  sine = sin(radian);
  cosine = sqrt(1 - sine*sine);
/* cosine = cos(radian);*/
  printf("%radian = %.2f\n",radian);
  printf("sin = %f\ncox = %f\n",sine, cosine);
return 0;
 }
```

6.3 赋值运算

1. 赋值运算符

赋值运算符记为“=”，注意它不同于数学中的等于符号“=”。由赋值运算符连接的式子称为赋值表达式。其一般形式为：

变量 = 表达式

表示将其右侧的表达式求出结果，赋给左侧的变量。其作用是把数据值写入变量，用表达式的值覆盖变量原来的值，例如

```
double pi = 3.14, area = 6.28, radium;
radium = sqrt( 2*area / (pi*pi));
```

除了赋值功能，该运算符还可以用于变量的初始化，两者的区别在于：前者是在定义变量时为其赋初始值，而后者是修改变量原来的值。

2.左值与右值

赋值符号右面的操作数为右值，右值表达式可以是常量、变量以及其他各种类型的表达式。赋值符号的形式也可以表示为：

左值表达式 = 右值表达式

大部分表达式都可作为右值，并不是所有表达式都能够出现赋值符号左侧。只有像变量一样有存储空间的表达式，才能作为左值（left value）表达式。算术表达式、常量以及函数调形式不是左值表达式。例如，对于变量 int x = 1, y = 2; double pi ;下面是错误的赋值表达式：

```
x + 1 = y;
3.14 = pi;
sqrt(y) = x;
```

3. 赋值表达式

赋值运算符的优先级很低，低于算术运算符。该运算符具有右结合性，即先计算右值表达式的值，然后将其值传递给左值。赋值表达式为左值表达式，因此允许连续赋值操作。如

```
a=b=c=5
```

可理解为

```
a=(b=(c=5))
```

在其它高级语言中，赋值表达式一般构成一个语句，称为赋值语句。 而在 C 中，把“=”定义为运算符，组成赋值表达式。凡是表达式可以出现的地方均可使用赋值表达式。例如，

```
x = (a =5 ) + (b =8)
```

是合法的。它的意义是把 5 赋予 a，8 赋予 b，再把 a，b 相加，将和赋予 x，故 x 的值为 13。

根据赋值表达式的规则，分析下面赋值语句的合法性：

```
x = y = z + 2 ;        /*  合法，等效于 x = （y = z + 2）*/
( x = y ) = z + 2 ;   /*  合法，赋值表达式作左值*/
z + 2 = x = y ;       /*  不合法，算术表达式不能为左值*/
num=3*4=4*5;   /*  不合法，等效于 num= (3*4=4*5)*/
```

4. 复合赋值运算

复合赋值运算符提供了赋值运算和其他运算结合的简洁形式，在赋值符“=”之前加上其它二目运算符可构成复合赋值符。例如，算术赋值运算符+=、-=、*=、 / =和%=，以及位复合赋值运算符<<=、>>=、&=、^=和|=。 构成复合赋值表达式的一般形式为：

变量　双目运算符= 表达式

它等效于

变量 = 变量 双目运算符 表达式

例如：

a += 1　　等价于 a = a + 1

x *= y + 7　　等价于 x = x * (y + 7)

r % = p　　等价于 r = r % p

初学者可能不习惯使用复合赋值符，但这十分有利于编译处理，能提高编译效率并产生质量较高的目标代码。而且，左值表达式的内容不必在右边重复出现，使表达式的书写更加容易和清晰。

例 6.2 编程求长方形操场的面积，操场的长和宽由用户输入，输出长和高的值以及操场面积。一般认为长度都是大于宽度的，如果不满足该条件，交换数据的值。

程序中需要定义两个变量 length 和 width 分别存储长和宽，当 width>lentgh 时，则需要交换两个变量的值。若用下面的操作，能否实现交换操作？

```
int length = 60, width = 75;
length = width;
width = length;
```

执行两个赋值赋值语句后，length 和 width 的值都为 75，数据 60 因被 75 覆盖而造成丢失。可见数学中常用的表示方式在程序中并不能实现数据的交换。可以借助一个中间变量 temp，进行两个变量值的交换，temp 的类型要与欲交换变量的类型相同。交换过程如图 6.1 所示。首先，用 temp 临时存储 length 原来的数据；然后，用 width 的值取代 length 的旧值；最后，将暂存在 temp 中的数据赋给 width，即 width 获得了 lentgh 的旧值。注意三个语句的执行顺序不能随意改变。

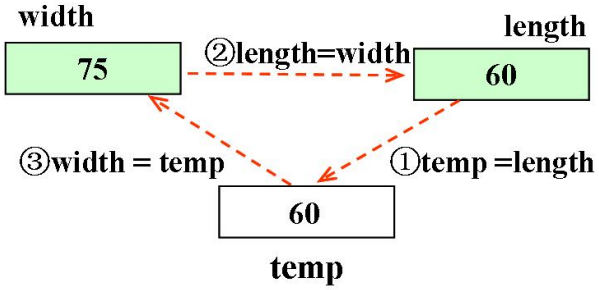

图 6.1 变量的交换

```
/*计算操场面积 */
#include<stdio.h>
int main()
{
    int  length,  width,  area ;
   int  temp;     /*用于交换的中间变量*/
   printf("Input length and width: ");
   scanf("%d %d",&length,&width);
   if( width>length )    /* 如果满足条件，则执行{}中的代码*/
   {  /* 交换变量的值*/
     temp = length ;
     length = width;
```

```
        width = temp;
    }
    /*如果不满足条件 width>length，程序跳过 if 语句后顺序执行*/
    area = length*width;     /*计算面积的赋值表达式*/
    printf("length = %d, width = %d, area = %d\n",    length, width,area);
  return 0;
}
```

运行结果

```
Input length and width: 60 75
length = 75, width = 60, area = 4500
```

该程序中使用了赋值表达式和算术表达式，还有关系表达式 width>length，关系运算符将在 6.4 节详细介绍。除了一般的表达式语句和函数调用语句，程序中使用 if 语句，当满足小括号中的条件时，执行花括号中的语句块，否则跳过语句块，执行 if 语句后面的代码。用关系表达式和 if 语句构成的选择结构，将在第 7 章中讨论。

6.4 表达式中的类型转换

C 语言允许在一个表达式中使用不同类型的数据进行混合运算，编译器根据一定规则自动将数据转换成正确的类型，在计算表达式的值时尽量不破坏数据的准确性。这种隐式转换不受程序员控制，也称为自动转换。程序员也可将某个表达式的值强制转换成特定类型，这种转换称为显式转换。

6.4.1 隐式类型转换

1. 混合运算中的类型转换

进行混合运算时，表达式当中包含多个类型的数据，在求表达式值的过程中完成自动类型转换。当运算符两边的操作数类型不同时，操作数需要通过转换保持类型一致，然后才能进行运算。计算过程中遵守“就高不就低”的原则，即类型级别低的操作数转换成级别高的操作数类型。数据类型的转换级别见图 6.2。

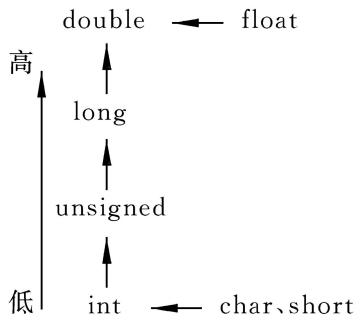

图 6.2 类型转换

级别低的操作数转换为级别高的操作数类型，对于一般算术计算有如下转换规则：

（1） 如果两个操作数有一个为实型，另一个为字符型或者整型，将后者转换成实型；

（2） 如果两个操作数为整型或字符型，类型的级别由低到高为

char → short → int → long int

（3） 如果两个操作数为实型，类型的级别由低到高为

float → double → long double。

2. 赋值运算中的类型转换

当赋值运算的左值表达值与右值表达式类型不同时，将右值转换成左值的类型。当执行此类自动类型转换时，有时会将级别较高的表达值转换为较低级别的类型，数据可能丢失精度，发生以下改变：

（1） 从 float 转换到 int 将导致小数部分被截断，例如 int x =3.14， x 被赋值为 3；

（2） 从 double 转换到 float 将导致数字舍入，例如 float x =3.1415976， x 被赋值为 3.14160；

（3） 当从 int 转换为 char，将 int 的低 8 位存储在 char 型变量中，将会导致高位丢失；

（4） 当从 long int 转换到 int，也将导致高位的丢失。

例 6.3 根据单价、数目、税金和折扣率计算订单的总价

为了计算总价，声明不同类型的变量存储单价、数目、税金和折扣率：

```
double price = 10.2;   /*单价*/
long count = 5L;   /*数目*/
int discount = 15;   /*折扣*/
float tax = 2.5f;   /*税金*/
long double total_cost = ( count*price + tax)* ( (100-discout)/100.0);
```

计算表达式时其类型需要逐一转换：

（1） 计算 count*price，将 count 转换为 double 类型再进行乘法运算，结果为 51.0;

（2） 计算 count*price + tax，将 tax 转换为 double 类型再进行加法运算，结果为 53.5

（3） 计算(100-discout)/100.0，将整型表达式 100-discout 转换为 double 类型，再进行除法运算，结果为 0.85;

（4） 将右值表达式的 double 型的结果转换为 long double 型赋值给 total_cost。

赋值计算可能会造成部分数据信息的丢失，编译器通常会给出警告信息，但是代码仍将被编译，程序可能会得到不正确的结果。当在代码中进行可能导致数据丢数的类型转换时，可以使用显式类型转换，或者选择合适的数据类型存储数据，从根本上消除这种潜在的错误。

6.4.2 显式类型转换

要计算班级人数中男女比例，用如下语句计算能否得到准确的结果？

```
int    male_mumber =26，  female_mumbe = 5;
float ratio = male_mumber / female_mumber;
```

按照自动类型转换规则，整型除整型结果仍然为整数，无法得到 0.52 而只是将 0.0f 赋给 ratio。此时需要按与自动类型不同的方式进行数据类型的转换，可以通过使用显式类型转换符（），其一般形式为：

(类型说明符) (表达式)

其功能是把表达式的运算结果强制转换成()中的类型。计算男女比例的表达式为：

（float）（male_mumber）/ female_mumber

先将整型变量 male_mumber 的值显式转换为 26.0，然后编译器将 female_mumber 转换成同意的类型。若将表达式写为如下形式

（float）（male_mumber / female_mumber）;

则整除完将算术表达式的值 0 转换为 float 型，仍然无法得到正确的结果。

利用强制类型转换运算符，可以将一个表达式的值转换成特定的类型，例如：

```
(double)a          (将 a 转换成 double 类型)
(float)(x+y)       (将 x+y 的值转换成 float 型)
```

(int)x%3　　(将 x 的值转换成整型才能进行求余运算)

注意，强制类型转换只改变表达式值的类型，而表达式中的变量 x、y 和 a 的类型保持不变。例如

```
double y = 3.1415926 ; int x ;
x = ( float) y ;
```

x 被赋值 3，但是 y 仍然为 double 型变量。

例 6.4 在长为 100 米、宽为 75 米的长方形操场中开辟一个环形跑道，剩余的为绿化面积。跑道包括直道和弯道，假设两个弯道都为一个半圆环。输入环的内外半径和直道长度，计算操场中跑道面积和绿化面积，并输出操场的绿化率。

辨析该程序中那些代码进行了类型转换，属于那种转换类型，它们是如何转换的？

```
#include<stdio.h>
#define PI 3.14
int main()
{
    int    length = 100, width = 75;
    float radiumIn, radiumOut,line; /*内外环半径和直道长度*/
    double area = length*width,      /*长方形面积*/
          rangArea;                          /*圆环面积*/

    printf("输入跑道内外半径和直道长度：  ");
    scanf("%f %f %f",&radiumIn, &radiumOut,&line);

    rangArea = PI*(radiumOut*radiumOut-radiumIn*radiumIn);
    rangArea += (radiumOut-radiumIn)*line;      /*跑道面积*/
    area -= rangArea;              /*绿化面积*/

    printf("跑道面积为  %.2f\n",rangArea);
    printf("操场绿化面积为  %.2f\n",area);
    printf("操场绿化率为  %.2f\n", 100*area / (length*width));
  return 0;
}
```

6.5 自增与自减运算

1. 自增和自减运算符

前面介绍了如何使用赋值运算符修改变量的值，以及使用+=或-=等复合赋值运算符，实现变量值的递增或递减操作。C 还提供了对数据进行加 1 或减 1 的简洁操作方式，通过自增运算符(++)以及自减运算符(--)实现。程序中往往需要一个计数器变量，对于变量 int counter = 0， 有以下表达式：

```
counter = counter +1
counter +=1
++counter
```

这三个表达式都能使 counter 的值递增 1，显然第三种形式最简洁的，程序员更喜欢使用自增运算符对计数器操作。

自增运算符和自减运算符为一元运算符，其优先级高于其他算术运算符。例如，若

counter = 5， 则执行下面的语句：

total = ++counter + 4 ;

先使 counter 自增为 6，再进行加法操作，最后 total 被赋值 10。等效于下面语句：

counter = counter +1 ;

total = counter +4;

自增和自减的操作数通常为整型变量，多用在循环结构中。 自增和自减运算符直接修改操作数的值，因此操作数必须为能修改左值。算术表达式和常量不能进行自增或自减操作。例如，以下为错误的表达式：

++5 （x*y）++ --PI（PI 为符号常量）

2. 前缀形式和后缀形式

自增和自减运算符可作为前缀和后缀，其表达式有以下几种形式：

++i 前缀自增： i 自增 1 后再参与其它运算，表达式的值为 i 加 1 后的新值

--i 前缀自减： i 自减 1 后再参与其它运算，表达式的值为 i 减 1 后的新值

i++ 后缀自增： i 参与运算后，i 的值再自增 1，表达式的值为 i 的旧值

i-- 后缀自减： i 参与运算后，i 的值再自减 1 ，表达式的值为 i 的旧值

前两种将运算符放在变量前，称为前缀形式，操作数 i 都是自增后的的值即为表达式的值，该表达式为左值表达式。后两种将运算符放在变量后，称为后缀形式，注意此时表达式的值和变量 i 的值不同。使用后缀自增时，先使用操作数原来的值进行计算，再递增操作数的值，该表达式不是左值表达式。例如，将前面的例子改为：

counter = 5;

total = 4 + counter++ ;

先使用 counter 原来的值 5 进行加法操作，counter 后自增为 6，total 被赋值 9。等效于：

total = counter +4;

counter = counter +1 ;

如果单独使用自增和自减运算符，前缀和后缀形式效果相同，例如++i 和 i++ 都使 i 值加 1。但是如果在混合运算中，自增或自减的操作数要参与表达式的其他运算，两种形式表达式的值不同。为了避免这种歧义的产生，最好单独使用++或—运算符。例如，将上面的操作改写成等价的形式：

total = counter+ 4 ;

++counter ;

这样操作就更清晰了。

前面的规则也同样适用于递减运算符，例如，如果 counter 的初始值为 5，则执行语句

total = --counter + 4 ;

total 的值为 8。若语句改写为：

total = 4 + counter-- ;

则 total 的值为 9。等价于操作--counter ;total = 4+counter 。

3. 结合性和副作用

对于表达式 a+++b 应该如何操作？自增和自减运算符的优先级和结合性有些复杂。前缀++和--运算符从右向左结合，后缀++和--运算符则从左向右结合，后缀高于前缀的优先级。例如，

4 +++counter 等效于 4 +（++counter）

coutner-- -4 等效于（counter--）- 4

a+++b 等效于(a ++)+b

a+ ++b 等效于 a +(++b)

为了使表达式的意义更加清晰，请用括号或者空格分隔多个连续的运算符，避免产生歧义。

使用自增和自减运算符可能会带来的副作用，应避免在一个表达式中多次使用自增或自减运算符。例如，对于变量 int counter=1， 有如下语句：

```
total =   ++counter*3   +   ++coutner / 5   +   ++coutner;
counter = ++counter + 1;
```

理解这些语句是相当困难的，而且重要的是，该语句多次修改了同一变量 counter 的值，标准 C 对这样的行为是未定义的，因此不同的编译器会有不同的理解，得出的结果是不确定的。请读者在不同的编译平台下验证此类代码，分析自增和自减运算的副作用。为了保证结果的一致性，计算表达式时最多对每个变量修改一次。

使用自增和自减运算符，经常容易出错，初学者难于理解后缀形式。特别是当它们出在较复杂的表达式或语句中时，容易产生歧义。因此，建议单独自增和自减运算符，尽可能不将其用于混合运算。

6.6 关系与逻辑表运算

1. 关系运算符

除了进行数值运算，计算机还能进行处理各种的逻辑问题。生活中有常需要进行判断和选择，如“红灯停，绿灯行”、“如果明天下雨，就要带雨伞”、“若身高超过 1 米 2，就要买票”等等。若希望程序也能同人一样，做出合理的判断和决择，就需要比较机制，C 语言用关系表达式表示比较的结果。

关系运算符用于比较数据关系，C 语言提供的关系运算符见表 6.3。其用法与数学中的符号类似，形式略有不同，除了“>”和“<”，其他关系运算符都是两个字符构成。但要注意，用于判断两个数值相等时，使用逻辑等号 “==”（两个连续的等号），它不同于赋值符号“=”。

表 6.3 关系运算符

运算符	说明	运算符	说明
>	大于	<	小于
>=	大于等于	<=	小于等于
==	等于	!=	不等于

关系运算符都是二元运算符，其结合性均为左结合。它们的优先级低于算术运算符，高于赋值运算符。在 6 个关系运算符按优先级可分为两类：其中“<”，“<=”，“>”和“>=”的优先级相同，“==”和“!=”的优先级相同，前一类的优先级高于后一类。

用关系运算符将两个表达式接起来，构成关系表达式，例如，以下为合法的关系表达式：

```
a+b > c-d
b*b-4*a*c >= 0
‘a’+1 == ‘b’
-2*a/b !=   0
```

关系运算符可连接各种表达式表达式，如变量、常量、算术表达式、赋值表达式等，也允许出现关系表达式嵌套的情况。例如：

```
a = 1 > b =2
a> ( b>c )
a!= (c==d)
```

关系表达式的值有“真”和“假”两种，用整型的 0 或 1（非 0）表示比较结果不成立或成立。新标准 C99 中增加了 bool 类型表示逻辑值，用 true 以及 false 代表真或假。例如，

对于变量 int x =3 y = 4，z =5，有如下关系表达式

x >= 2　关系成立，值为 1

y < z+x　关系不成立，值为 0

字符型变量和常量可以参与数值运算，也可以用在关系表达式中，比较它们的 ASCII 码值。例如，有变量 char c1= ‘A’， c2 = ‘B’，则

c1 > c2　值为 0

c1 == c2 – 2　值为 0

c1 + 32 >=‘a’　值为 1

c2 – c1 != ‘1’　值为 1

2. 逻辑运算符

程序按照关系表达式的结果，决定下一步进行怎样的操作。有时需要处理更为复杂的问题，比如判断今年是 365 天还是 366 天。凡是符合下面条件二者之一的年份为闰年：

（1） 能被 4 整除，但不能被 100 整除。

（2） 能被 400 整除。

用一个关系表达式很难得出正确的结论，可以将多个关系表达式用逻辑运算符组合起来，解决更多的逻辑问题。

C 语言中提供了三种逻辑运算符，用法见表 6.4。和关系表达式一样，逻辑表达式的值为 1 或 0，表示逻辑“真”或“假”。

表 6.5 逻辑运算符

运算符	表达式	说明
&& 逻辑与	E1&& E2	当表达式 E1 和 E2 都为真时，逻辑表达式的值为 1； 当表达式 E1 和 E2 任意 1 个为假时，逻辑表达式的值为 0；
|| 逻辑或	E1 || E2	当表达式 E1 和 E2 至少有 1 个为真时，逻辑表达式的值为 1； 当表达式 E1 和 E2 都为假时，逻辑表达式的值为 0；
! 逻辑非	!E	当表达式 E 的值为 1 时，逻辑表达式的值为 0 当表达式 E 的值为 0 时，逻辑表达式的值为 1

逻辑运算符中，“!”为单目运算符，具有右结合性，其优先级高于算术运算符和关系运算符。“&&”和“||”均为双目运算符，具有左结合性，其优先级低关于系运算符，高于赋值运算符。按照运算符的优先顺序讨论以下表达式：

a>b && c>d　等价于　(a>b)&&(c>d)

!b==c||d<a　等价于　((!b)==c)||(d<a)

a+b>c&&x+y<b　等价于　((a+b)>c)&&((x+y)<b)

因此，判断闰年的问题对应的逻辑表达式为 ：

year％4==0　&&　year％100!=0　||　year％400==0

等价于

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

3.逻辑运算时应注意的问题

构建正确的关系表达式和逻辑表达式是程序能作出明智的判断的前提。使用关系运算符和逻辑运算符应注意的下几个问题：

（1） 正确表示区间和范围

数学中经常讨论数值的范围，比如 x 在（0，4）区间内，或者 y 属于区间闭[a b]，初学者经常用关系表达式错误的表示为如下形式：

0< x < 4

a<= y <= b

按照关系表达式的运算规则，当 x= -2 时 x 不在（0，4）区间内，但表达式 0< x < 4 的值为 1，表示关系成立；当 y=0.3 时 y 在[0 , 0.5]范围内，但表达式 0<= y <= 0.5 的值为 0，表示关系不成立，两种情况的表达式值和实际条件不符合，导致逻辑错误。

C 语言中使用逻辑表达式表示数据区间范围，用逻辑与将两个关系表达式连接起来，即 E1&&E2，表示当满足关系 E1 且同时满足关系 E2 时，表达式的值为 1。因此，上面的表示 x 和 y 区间的表达式为：

0< x && x< 4

a<= y && y<= b

（2） 逻辑表达式的值参与运算

由于关系表达式和逻辑表达式的有整型值，因此可以参与混合表达式运算。例如，

```
int i = 1 , j = 7 , odd , even ;
odd =   ( i %2 != 0 ) + ( j % 2 != 0 );
even =   ( i %2 == 0 ) +( j % 2 == 0 );
```

odd 和 even 分别为 i 和 j 中奇数的个数和偶数的个数。但是这样的表达式不太容易理解，应尽量避免使用这种意义不清的表达式，如 5 > 2 > 7 > 8 和 a = i + (j % 4 != 0) 。

（3） 区分逻辑等与赋值符号

C 语言用“==”表示数学中的等于“=”，用以判断两个数据的相等关系。初学者经常混淆逻辑等运算符“==”和赋值运算符“=”。若有变量 int i = 1 , j = 7，下面两个表达式的值不同：

```
result =   i = j;     /*   result = 7 */
result =   i==j     /*   retult = 0 */
```

如果误用两个运算符，会造成不易发现的逻辑错误。

此外，使用逻辑等或逻辑不等运算符，可以准确的判断两个整型数据是否相等。但由于计算机不能精确表示所有实数，因此最好不要用“==”和“!=”连接两个实数。例如，

1.2345678901234567897 == 1.2345678901234567898

两个操作数的末尾不同，但是表达式的值为 1，这是由数据有效位数引起的误差。比较实数是否相等时，宜采用求误差值形式：

fabs (x - y) < ε

表示实数 x 与 y 相当接近，ε为设定的精度，一般为很小的数。例如，代数中 x=3.14 描述乘如下表达式

fabs(x – 3.14) < 1e-5

表示| x – 3.14 |≈0，若该关系表达式值为 1，则代表 x 的值约等于 3.14。

6.7 其他运算符

1. 条件算符

条件运算符是 C 语言中唯一的三目运算符，即有三个操作数参与运算。由条件运算符组成条件表达式的一般形式为：

表达式 1 ？ 表达式 2 : 表达式 3

其求值规则为：如果表达式 1 为逻辑真（值为非 0），则将表达式 2 的值作为条件表达式的值，否则表达式 3 的值为整个条件表达式的值。条件表达式通常用于赋值语句之中。例如

```
max = ( a>b ) ? a : b ;
```

语句的功能是求两个数中的最大值，若条件 a>b 成立，则把 a 赋予 max，否则把 b 赋予 max。

条件运算符“? :”是一对运算符，不能分开单独使用。它的优先级低于关系运算符和算术运算符，但高于赋值符。它的结合方向是自右至左。例如

max=a>b?a:b　　等效于 max = (a>b)?a:b

a>b?a:c>d?c:d 等效于 a>b?a:(c>d?c:d)

第二个表达式是条件表达式的嵌套情形，即表达式 3 又是一个条件表达式。

2. 逗号运算符

在 C 语言中逗号“,”也是一种运算符，又称为“顺序求值运算符”，逗号常可作为分隔符使用。用逗号将多个表达式连接起来组成一个表达式，称为逗号表达式，其一般形式为：

表达式 1,表达式 2

逗号其求值过程是分别求两个个表达式的值，并将表达式 2 的值作为整个逗号表达式的值。例如，

```
int a = 2, b = 4, c = 6, sum1, sum2;
sum1 = ( a+b, b+c );          /* 等效于 sum = b + c , sum 的值为 10*/
```

逗号运算符的优先级最低，对于上面的例子，改写表达式后执行下面的语句：

```
sum2= a+b, b+c          /* 等效于 sum = a+ b */
printf("sum1=%d,sum2=%d",sum1,sum2);
```

输出 sum1 = 10，sum2 = 6。整个逗号表达式的值仍为 10，但 sum 的值为 6。

用逗号连接或分隔多个表达式时，逗号表达式可扩展为以下形式：

表达式 1,表达式 2,…表达式 n

整个逗号表达式的值等于表达式 n 的值。程序中使用逗号时，并不一定要求整个逗号表达式的值。例如，在变量说明语句中以及函数参数表中，逗号只是作为间隔符。

3. sizeof 运算符

程序中常需要知道操作数的尺寸，即在内存中存储该类型操作数需要的几个字节。使 sizeof 运算符可以计算变量、数组以及某个表达式的尺寸，结果为一个整型常量。也可以使用该运算符观测当前系统中某种类型的大小，请读者自行运行下面程序，仔细分析运行结果。

例 6.4　　测试不同类型变量的尺寸

```
#include<stdio.h>
int main()
{
   int i = 1;
   char c='A';
   double d = i + c;
   printf("sizeof(int): %d \n", sizeof(i) );              /*输出 int 型变量 i 的尺寸*/
   printf("the size of short and long: %d %d \n",
              sizeof (short),sizeof (long) );          /*输出常整数和短整数的尺寸*/
  printf("sizeof(double): %d \n", sizeof(d +3.14) );   /*输出 double 型表达式的尺寸*/
   printf("sizeof(i+c): %d \n", sizeof(i+c) );          /*输出 int 型表达式数据的尺寸*/
   return 0;
}
```

6.8 运算符的优先级与结合性

在混合计算中，计算顺序取决于运算符的优先级和结合性。前面介绍了常用运算符的用法，下面对其优先级和结合性进行总结。

就运算符的优先级而言，单目运算符高于双目运算符，算术运算符高于关系和逻辑运算符，赋值符号和逗号最低。常用运算符按优先级从高到低排列见表 6.5。

同一优先级的运算符，运算次序由结合方向决定。大部分运算符都为左结合，只有少数

运算符为特殊的右结合，包括部分一元运算符、赋值运算符以及和赋值运算相关的运算符。

为了增强代码可读性，在混合运算中最好用括号标明实际运算顺序。例如

i +1 < j * 4 && ! P || Q

等价于 ： (((i +1) < (j * 4)) && (! P)) || Q

P != i < j || Q && S

等价于 ： (P != (i < j)) || (Q && S)

表 6.5 运算符的优先级和结合性

优先级	含义	运算符	运算数	结合性
1	初等运算	[ ] ( ) -> .	1 个 单目 运算符	左结合
2	自增自减	后置++ 后置--		左结合
3	一元运算	前置++ 前置-- 加号+ 减号- & * ~ ! sizeof()		**右结合**
4	类型转换	（类型）		左结合
5	算术运算	* \ %	2 个 双目 运算符	左结合
6		+ -		左结合
7	位移运算	<< >>		左结合
8	关系运算	< > <= >=		左结合
9		== !=		左结合
10	位与运算	&		左结合
11	位异或运算	^		左结合
12	位或运算	|		左结合
13	逻辑运算	&&		左结合
14		||		左结合
15	条件运算	?=	3 个 三目运算	**右结合**
16	复合赋值 运算	= += -= *= /= %= <<= >>= &= ^= |=	2 个 双目 运算符	**右结合**
17	分隔符	,		左结合

6.9 案例分析

程序设计不是随性而发或者信手拈来的，如果没有规划，一开始就编写代码，很难高效的实现程序。编程必须按照科学的方法分析问题、设计算法并用相应的语言实现。按照软件工程科学的理论，程序设计应划分为如下步骤：需求分析、概要设计与详细设计、代码实现以及测试。首先，需要对问题进行分析，确定需要哪些数据和输出怎样的结果；其次，设计算法，对复杂的算法需要逐步细化，直至安排好每个细节和步骤；然后，按照设计方案用 C 语言编写程序，实现算法；最后，运行并测试程序，修改程序中的各种错误。

C 语言提供了丰富的运算符实现各种运算，由它们可以构成灵活而简洁的表达式。一个

简单的 C 程序可由表达式语句和输入输出语句组成，一般算法包括以下步骤：

（1） 准备数据：定义变量与常量；

（2） 获取数值：为数据赋值或输入数据值；

（3） 加工数据：表达式计算；

（4） 显示结果：输出计算结果；

下面讨论一个简单程序的设计和实现过程。

问题描述

编写程序求解一元二次方程 $ax^2+bx+c=0$ 的根。

算法分析

解决此类数学问题，首先要了解求解公式及过程。为了保证方程有实根，方程系数 a、b 和 c 满足如下条件：$b^2-4ac\geq 0$ 且 $a\neq 0$，则一元二次方程式的根为 ：

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a} \quad , \quad x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$

可将上面的根分解成一下两项：

$$p=\frac{-b}{2a} \quad , \quad q=\frac{\sqrt{b^2-4ac}}{2a}$$

则两个根可以表示为 $x_1=p+q$， $x_2=p-q$。

算法设计

程序求方程根的方法和数学上的求解过程相似，算法的步骤为：

（1） 定义变量并初始化；

（2） 计算中间变量的值；

（3） 计算方程根 root2 和 root2 的值；

（4） 输出计算结果。

上面的算法中，若某个步骤的操作不够明确，还需进行细化，完善后的算法为：

（1） 定义 float 型变量并初始化：

变量包括根的系数 a、b、c，方程的根 root1、root2，中间变量 disc、p 和 q。

（2） 计算中间变量

求根的判别式 $disc=b^2-4ac$

求中间变量值 $p=\frac{-b}{2a}$ 和 $q=\frac{\sqrt{disc}}{2a}$；

（3） 计算方程根 root2 和 root2 的值：

$root1=p+q$, $root2=p+q$

（4） 输出计算结果，例如

x1= -0.5858, x2= -3.4142

实现时需注意表达式的正确表示方法。

代码实现

```
/* 求解一元二次程的根*/
#include<stdio.h>
#include<math.h>
int main()
{
  /*定义变量*/
```

```
    float a = 1,b = 4,c = 2 ; /*方程系数*/
     float root1,root2,   /*方程的根*/
           disc,p,q;       /*中间变量*/
     /*求方程的根*/
     disc = b*b-4*a*c;
     p = -b /(2*a) ;
     q = sqrt(disc)/(2*a);
     root1 = p + q;
     root2 = p - q
    /*输出结果*/
    printf("x1=%8.4f, x2=%8.4f\n",root1,root2);
     return 0;
}
```

程序测试

为了验证该程序，可通过多组数据来进行测试。在运行程序前，可改变变量 a、b 和 c 的初值，将输出结果和正确结果进行比较，判断程序是否正确。例如:

测试用例 1：a = 1,b = 4,c = 2;

程序输出： x1= -0.5858, x2= -3.4142

测试用例 2：a = 1,b = 2,c =1 ;

程序输出： x1= -1.0000, x2= -1.0000,

测试用例 3：a = 0.5,b = 4.2,c = -1.2;

程序输出： x1= 0.2766, x2= -8.6766

通过观察以上 3 组输出，验证代码的正确性。有时通过结果很难直接定位错误所在，可以借助集成开发环境中的调试工具，逐步排查错误。本节介绍 Vc++6.0 中的基本调试方法。

首先，在工具栏上点击右键，在弹出菜单上选择调试选项“debug”，出现浮动面板中的各种调试工具，如图 6.3 所示。

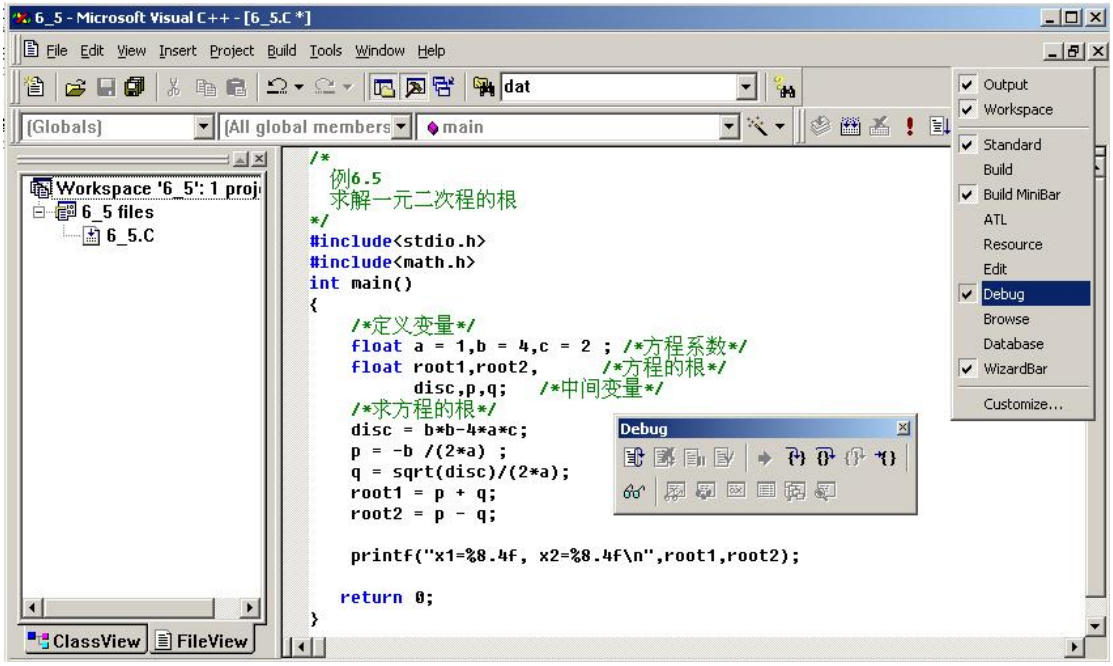

图 6.3 显示调试工具栏

使用基本调式工具，可用鼠标点击调式面板中的按钮，如图 6.4 所示。其中，单步执行（Step over）的快捷键为键盘中的 F10，该方法可使程序逐条语句执行；按钮（Run to Cursor）对于的快捷键为 Ctrl+F10，可使程序执行到光标所在处。

图 6.4 基本调试按钮

点击调试面板中变量（varible 按钮），可观察程序运行过程中各个变量的值。在图 6.5 中左下方的变量窗口中，跟踪变量值的变化。点击观察（watch）按钮出现观察窗口，输入变量的名字，可以在图 6.5 中右下方的窗口中观察特定变量。

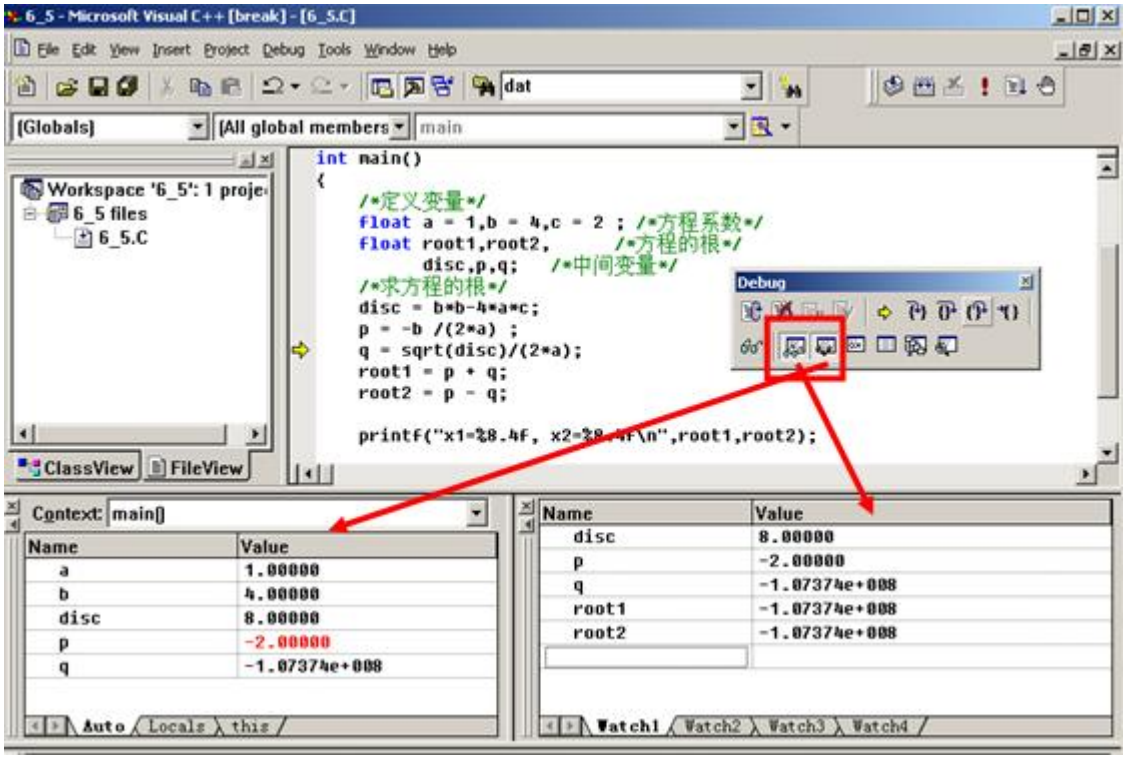

图 6.5 观察变量窗口

使用这些调式方法，执行某个语句后查看各个变量或表达式的结果，判断实际运行结果是否和预期结果一致。合理的利用调试工具，能快速准确定位错误，为调式程序提供捷径，后面的章节中会介绍其他调试工具。

通过测试可以发现，有一些特殊情况不能输出正常的结果。例如，当 a =0 时，会输出结果

x1= -1.#IND, x2= -1.#INF

数学中要求除法运算中分母不能为零，而对于程序中的除 0 操作，标准 C 中未定义其行为。又如，当根的判别式 disc 不满足条件 disc≥0 时，输出和除 0 类似的结果。由于对负数 disc 进行开方操作 sqrt(disc)，方程无实根，导致无法输出正确的结果。因此该程序不够完善，需根据变量 disc 和 a 的值进行判断，采用不同的方法计算根的值，该问题在下章中继续讨论。

习题

1．输入三角形的底和高，计算并输出其面积。

2．输入一个 3 位整数，将每位的数据进行分离，要求逆序输出对应的数，即分别按照字符型输出个位、十位、百位上的整数。例如，输入 123，输出 321。

3．输入三个小数，判断并输出其中的最大值和最小值。

4．自己写一个算术表达式，计算结果，要求用到所有算术运算符。

5．设计习题，练习自增与自减运算符，体会它们的用法和副作用。

6．写出符合下列条件的逻辑表达式

1）一个既能被 2 整除又能被 3 整除的正数；

2）对于平面中的点，坐标表示为 (x,y) ，表示落入图 6.6 中绿色部分 (不压线)的点 ；

3）对于三角形的三条边 a、b 和 c，能构成合法的三角形的表达式；

7．输入三角形的两个边长，计算其中一个角的三角函数，比较该值与直接调用标准库函数所得的结果是否相同。

8．编写程序预测断电后冰箱的温度。断电后经过一段时间，温度保持情况由如下公式决定：

$$T = \frac{4t^2}{t+2} - 20$$

其中，t 为断电后经过的时间（小时），T 为温度（℃）。程序提示用户输入时间，它以整数小时和分钟表示，需要将其转换为浮点型的小时数。输出此时的温度值（小数点后 2 位有效数字）。

9．编程计算汽车的平均速度

汽车在高速公路上匀速行驶。沿途都有距离上路地点的里程标志，已知开始和结束的里程，分别输入上路时间和下路时间（时、分、秒），假设汽车上路和下路的时间在同一天。计算汽车在该段形式的平均速度，并以“公里数/每小时”的形式输出平均速度。程序的运行界面如所示。

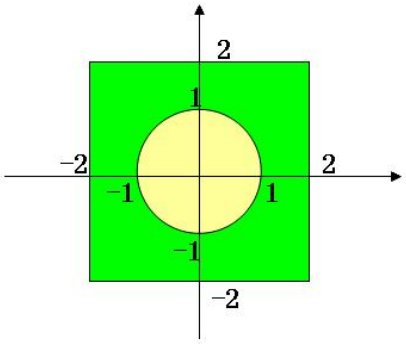

图 6.6　习题 6(2)

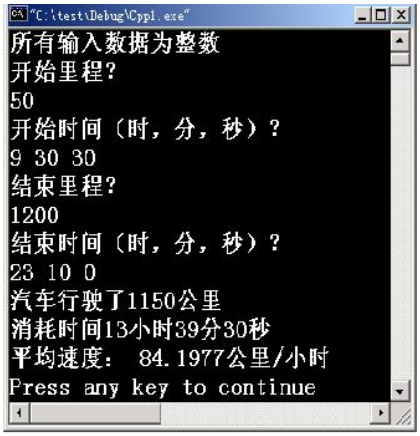

图 6.7 计算汽车的平均速度程序运行界面

10. 编程计算汽车销售人员的工资。销售人员的总工资包括以下几部分： 基本工资、提成 和奖金。其中基本工资为定额，按当月的销售台数提成（即按提成=每台汽车提成×销售台数）。若销售总额高于 100000 元则将总销售量的 2%作为奖金。设每台汽车的单价为 10800 元，每台提成为 1000 元。输入销售人员的基本工资和销售台数，计算他当月的工资。

附录 ASCII 表

码值	字符	码值	字符	码值	字符	码值	字符	码值	字符	码值	字符
0	NUL	22	SYN	44	,	66	B	88	X	110	n
1	SOH	23	ETB	45	-	67	C	89	Y	111	o
2	STX	24	CAN	46	.	68	D	90	Z	112	p
3	ETX	25	EM	47	/	69	E	91	[	113	q
4	EOT	26	SUB	48	0	70	F	92	\	114	r
5	ENQ	27	ESC	49	1	71	G	93	]	115	s
6	ACK	28	FS	50	2	72	H	94	^	116	t
7	BEL	29	GS	51	3	73	I	95	_	117	u
8	BS	30	RS	52	4	74	J	96	`	118	v
9	HT	31	US	53	5	75	K	97	a	119	w
10	LF	32	space	54	6	76	L	98	b	120	x
11	VT	33	!	55	7	77	M	99	c	121	y
12	FF	34	"	56	8	78	N	100	d	122	z
13	CR	35	#	57	9	79	O	101	e	123	{
14	SO	36	$	58	:	80	P	102	f	124	|
15	SI	37	%	59	;	81	Q	103	g	125	}
16	DLE	38	&	60	<	82	R	104	h	126	~
17	DC1	39	'	61	=	83	S	105	i	127	
18	DC2	40	(	62	>	84	T	106	j		
19	DC3	41	)	63	?	85	U	107	k		
20	DC4	42	*	64	@	86	V	108	l		
21	NAK	43	+	65	A	87	W	109	m		