C语言高级编程技术

Pascal使用者的第二步

中国科学院希望高级电脑技术公司

C语言高级编程技术

Pascal使用者的第二步

中国科学院希望高级电脑技术公司

前　言

C语言最初由Dennis Ritchie在AT &T贝尔实验室设计出来。它是在DEC PDP-11机上实现的。现在，C语言的编译程序几乎在包括大型机、小型机和微型机的所有计算机上都可用。C语言的普及在过去几年里得到巨大发展；实际上，它是当今最常用的语言之一，具有广泛的应用，如许多编辑程序、编译程序、数据库和操作系统（尤其是Unix），都使用这种语言编写。

这本书是为已有用Pascal编程经验的读者学习C语言程序设计而编写的。不懂得Pascal仍有可能学习这本书，但必须对一些高级程序设计语言方面的知识有良好的了解。

这本书适用于学生、职业程序员和计算机爱好者自学C语言。尤其适合那些要求学习C语言或者将要学习需要C语言知识的高级计算机课程，如操作系统或编译程序等的学生。

我们这本书的主要目的是研究如何用C语言编写可靠的程序以及如何把程序从Pascal语言转换的C语言。

编写这本书的第二个（但不容忽视的）目的是为读者用C语言设计程序提供对不同问题的简易的参考。

另一个目的是使用可移植的C语言，实际上意指我们使用Kernighan和Ritchie标准。已经出现了该标准的许多扩展，如在ANSI C中描述的虚过程、枚举类型等。我们阐述了多数常见的扩展。

学习C语言的第一步是懂得它的语法。我们通过把C语言结构与Pascal相应结构进行比较来帮助读者完成该步。

第二步是完全懂得语言的语义。我们这里的方法是给出几个C语言程序，解释它们的语义并提醒什么是有代表性的易犯错误。

最后一步是探讨程序设计风格。为此我们给出了用C语言程序员的常用风格编写的较大的程序。这种风格在多数地方是简洁的，但我们回避使用“巧妙”的代码，尽管这样做会导致低较代码。

这本书有两个基本部分。第一部分（第2章——第11章）是对C语言的介绍，它完成上面谈到的前两个步骤的目的。由于C语言可能对读者比较陌生，故这些章注意集中于语言本身。算法都相当简单，有时简直不值一提，因此读者不必花费时间去弄懂所用的数据结构和算法。

第二部分（第12章——第14章）假定对在第一部分介绍的C语言基础概念有基本的了解。我们继续讨论C语言程序设计技术并通过一些较大程序的例子介绍C语言的其它特性。这部分介绍的问题以及所用的数据结构和算法较为高深些。

这本书的重要特色包括：

·对C程序设计语言和程序设计技术的完全详细的阐述，包括C语言和Pascal结构的比较以及高深范围内的课题，如悬挂关联问题、堆压缩和其它一些问题。

·对指针的较早引入和对指针与数组、函数和字符串关系的完整详细的阐述（见第7、8、10章）。

·下面数据结构和算法在C语言中的实现：链表（基于指针的和基于数组的两种），栈的数组实现、字典的开杂凑表的实现，图的邻接表表示，集合的位向量实现，管理包括结构化数据的文件的技术，带堆压缩的存储器管理、内部和外部检索技术，可适用于由任何类型元素组成的数组的“虚排序”，带有文件压缩的外部二叉搜索树。

·在Unix、MS-DOS和Macintosh不同环境下C语言的不同实现。说明如何在这些环境里使用指定的编译程序（第14章）。然后通过程序设计例子，我们讨论下面问题：I／O重定向、差错处理、Unix下的低级文件I／O；ANSI C标准、MS-DOS服务、目标、MS-DOS下的文件特性，Macintosh上的窗口和菜单管理。

教法辅导

介绍性质的前十章每章开始都复习Pascal结构，它阐述诸如与标准Pascal有关的系统栈和堆的概念。跟在这个复习后面的是一个词汇表，在学习该章其余部分应懂得这种词汇。在某些情况下，读者可以词汇表开始一章的学习，仅在需要时查阅这个Pascal复习。

这本书通过例子教授C语言。我们提供了从简单的找出两个整数最大值到外部二叉检索树实现的100多个完整的程序。很多例子（在有关实际系统的第14章中的例子除外）都可以移植的风格编写。每章以应记住的要点和新手C语言程序员常犯的错误两个表结束。我们还提供了100多个习题，并给出了奇号习题的答案。

第1章给出了本书通常用到的一些基本概念，如语言的运行时系统。它还简要地比较了Pascal和C语言。第2章是关于基本数据类型和基本的终端I／O。第3章阐述控制结构。第4章介绍文件I／O（在第12和14章进一步讨论它们）。第5章是关于预处理的。第6章阐述函数和作用域规则。第7章是关于指针的。第8章和第9章定义数据、结构、联合和枚举类型。第10章阐述C语言字符串操作。第11章阐述字位操作。第12章是第4章的继续，它讨论正文和二进制文件I／O的附加细节。第13章介绍一些应用程序：

·计算器程序，它说明链表和开杂凑表的实现。

·应用Dijkstra算法找出图的最短路径的实现，它说明集合和图的实现。

·数据库程序，它说明外部二叉搜索树的实现。

第14章通过例子阐述一些实际系统——Unix、MS-DOS下C语言的使用。

附录包括C语言关键词表和C语言语法的完整描述（附录A），优先级和结合性表（附录B），格式化I／O的详细描述（附录C）和ASCII表（附录D）。附录E包括对奇号程序设计习题的解答。附录G包括对标准库函数的总结。

目　录

第一章 简 介

本章介绍一些你懂得以后章节所需了解的基本知识。首先，我们简要评述一下C语言，然后把Pascal和C语言进行比较。其次，我们讨论整数和字符数据类型在存储器里的表示。最后，介绍运行时系统的概念，包括C语言的文件、条件编译和分别编译。

1.1 C语言

C语言经常被认为是一种低级的高级语言，因为它提供许多允许低级操作执行的工具。因此，C语言也获得了没有结构化和难以阅读的名声。以某些方面来说，这或许能被接受，但要说它是C语言的独有特色是不公平的。可读性差的程序可以使用任何程序设计语言毫无困难地编写。进一步地，作为一种真正的高级语言，C语言提供许多高级的特性。若遵循某些规则，则用C语言设计的程序可以和Pascal语言设计的程序一样有结构性和可读性。C语言的另一重要特色是它的很多实现是很有效率的。

C语言提供典型的基本数据类型：字符型、整数型和实数型。它也提供结构数组类型：数组、记录和联合。它提供一组完整的控制结构，包括条件语句、选择语句以及循环语句。函数可以递归，虽然在文本中它们不能嵌套。程序可以分作可分别编译的模块，一组灵活的作用域规则的存在帮助了这种模块化。C语言最有力的一个方面是它指针的使用，许多语言结构（例如关联调用）都是使用指针实现的。运行时程序库提供标准I／O，这个程序库使得编译程序到其它机器的移植变得比较容易。C语言没有提供如Ada里那样的高级数据类型，也不提供支持并行程序设计的工具。

很多年来，Kernighan和Ritchie给出的C语言的定义是所接受的标准定义。现在C语言正被ANSI标准化。所幸的是，ANSI C定义中的大多数建议是Kernighan和Ritchie标准的扩展，现在所存在的大多数实现也遵循Keringhan和Ritchie，只有一些较新的编译程序适用于ANSI C的各方面。因此，我们有可能比较容易编写出“几乎”可移植的程序。把程序从一个机器移植到另一个机器上，即使有变化，也是比较小的。

1.2 Pascal和C语言的简单比较

本节我们简要比较Pascal和C语言的结构。我们的目的不是给你特定结构的详尽信息，而是让你大致了解C程序设计语言是什么样的。

标识符和注解

两种语言标识符的语法几乎是一样的，只是C语言中允许下划线“_”，并且它们是格感知的（case sensitive）。注解也相同

Pascal	C
(*comment*)	/*comment*/

程序模块和作用域

C语言的程序是一组函数序列。必须有一个main函数，程序以这个函数起开始执行。虽然C语言的作用域规则和Pascal类似，但C语言函数不能在文本中嵌套，可是它们可以包含程序块（由说明以及语句组成的复合句）。和Pascal不同的是C语言中的子例程都定

义为函数，虽然函数的返回值不被考虑时就叫作过程。和在Pascal中一样，C语言中的函数可以递归。下面是一个简单的例子：

```
PROGRAM one;

PROCEDURE proc;
BEGIN
    WRITE('Hello');
END;

FUNCTION func : INTEGER;
BEGIN
    func := 3;
END;


VAR i : INTEGER;
BEGIN
    proc;
    i := func;
    WRITE(i);
END.
```

```
void proc()
{
    printf("Hello");
}

int func()
{
    return(3);
}


int i;
main() {
    proc();
    i = func();
    printf("%d", i);
}
```

上面的C语言程序定义了三个函数：main，proc和func。

不同于标准Pascal，C语言支持函数的分别编译；关键词extern说明一个在另一文件中定义的对象。

数据类型和说明，参数

Pascal支持基本数据类型整数、实数、布尔量和字符。C语言支持类型int，float（它和在Pascal中的real一样）和char。C语言没有布尔类型，虽然任何非零整数等价于真，零等价于假。此外，C语言提供一个专门的限定词long定义双精度整数，它有两倍之多的有意义的数字。相似地，C语言提供类型double定义双精度实数。

C语言类型说明的语法和Pascal相反，例如

```
VAR i：INTEGER；        int i;
VAR c：CHAR；           char c;
```

C语言中，数据可在定义中初始化，例如：

```
int i=3；
```

Pascal支持结构数据类型数组、记录和文件，C语言支持同样的类型，只是文件不是在语言中预定义，而是在系统库里定义。

例

```
VAR arr : ARRAY [0..3] OF INTEGER

rec : RECORD
        i : INTEGER;
        r : REAL;
      END
```

```
int arr[4];

struct {
    int i;
    float r;
} rec;
```

函数定义相似，虽然形式参数说明所在的位置不同，例如：

```
PROCEDURE proc(i:INTEGER);
VAR j:INTEGER;
BEGIN
    j := i + 3;
END
```

```
void proc(i)
int i;
{
    int j;
    j = i + 3;
}
```

Pascal支持两种参数传递方式——值调用和关联调用。C语言只支持值调用，关联调用用指针模拟。

表达式和语句

C语言的表达式和Pascal类似，只是前者的类型由于自动类型转换可随意混合使用：

Equality:	=	==
Inequality:	<>	!=
Logical AND:	AND	&&
OR:	OR	||
NOT:	NOT	!

Assignment:	`x := 3*(y+6);`	`x = 3*(y+6);`
Conditional statement:	`IF a>5 THEN a:=a-b` `ELSE b:=a;`	`if (a>5) a=a-b;` `else b=a;`
Iteration statements:	`WHILE x>0 DO x:=x-1;`	`while (x>0) x=x-1;`
	`REPEAT x:=x+2` `UNTIL x>100;`	`do x=x+2;` `while (!(x>10`
	`FOR i:=1 TO 10 DO` `  x[i] := 0;`	`for (i=1; i<=10; i++)` `  x[i] = 0;`

Selection statement:	`CASE i OF`
	`1:  j := 3;`
	`2:  j := j-1;`
	`END;`
		`switch (i) {`
		`case 1: j = 3;`
		`  break;`
		`case 2: j--;`
		`  break;`
		`}`

Compound statement:	BEGIN *s1*; . . . ; *sk* END	{ *s1*; . . . ; *sk* }
Return from a function:	`func := exp;` `end (* function *)`	`return(exp);`
Pointer declaration:	`VAR p : ^INTEGER;`	`int *p;`
Pointer dereferencing:	`p^`	`*p`
Access to record fields:	`rec.field`	`rec.field`
Record pointer access:	`recpnter^.field`	`recpnter->field`

C语言的一维数组被认为是指向存储器里该数组存储区位置的指针。C语言允许对指针的算术运算，例如，若X是一个一维数组，则X+1指向该数组的第二个元素。一个不带括

号的函数名也是一个指针，这时指向这个函数的代码。

C语言的其它结构

C语言支持带参数的宏。这些宏在程序编译以前扩展。C语言预处理程序处理文件包含和条件编译。

1.3 计算机存储器

一个计算机存储器包括一些字，每个字包括一些字节，每个字节由一些位组成（通常是8位，虽然不尽如此）。例如，16位的字包括两个8位字节，64位的字包括八个8位字节。

大多数情况下，我们假设每个存储器位置是一个字节大小并且单独一个字符存放在一个字节里。这就是我们通常所指的面向字节的存储器结构，这是现在所用的比较一般的结构，如VAX－11和IBM－360以及它们的后继者都使用这种结构。支持所谓面向字结构的机器比较少见。在这些结构中，最小可编址的单元是一个字而不是一个字节。例如，在DEC－20上，每个存储器位置是36位大小；在Cyber180上，每个位置是60位大小。

从程序语言设计者的观点看，面向字节的结构和面向字结构的主要区别正如我们上面谈到的那样，在面向字节的机器上，单独一个存储器位置仅能保存一个字符的值，而在面向字的机器上，单独一个存储器位置能保存几个字符的值。例如，在DEC－20上，每个存储器位置能保存五个7位字符，剩一位没用到。对语言设计者来说，如果考虑可移植性，这会引起一些困难。本书的许多例子能够在任何机器上运行而不必考虑它们的存储器结构。然而一些例子是为通常面向字节的机器设计的，想让它们在面向字的结构上也能正确运行就需要做些修改。

所有的数据对象——如整数或记录——在程序执行期间保存在由几个连续字节组成的存储区里。数据对象的大小就是它所占据的字节数

对于整数和字符数据的简要讨论会对我们了解C语言的某些方面有所帮助。C语言中有两种类型的整数：带符号的和无符号的，无符号整数的代码用纯二进制数翻译，因此一个数的所有位都被用来确定它的值。

类似标准的十进制系统，纯二进制数（也称作无符号的）是一种位置表示。代码的每个位置有一个固定的权，在计算代码值时用做乘数因子。代码的值是每个数字被权所乘后求得的总和。

在十进制中，权是10的幂（从右向左是1，10，100，1000等），而在二进制中，权是2的幂（从右向左是1，2，4，8，16等）。例如要决定二进制数1011的十进制值，执行下面的计算：

```
1101 = 1*8 + 1*4 + 0*2 + 1*1 = 13 decimal
       |||└── weight 1
       ||└─── weight 2
       |└──── weight 4
       └───── weight 8
```

如果给数编码用了几位，则可能取得的最大值是2^{n-1}。

带符号数据比较复杂。要表示带符号数据，其中一位（通常是高位）用来指示数的符号。其余位用来确定它的值。如何确切地确定它的值依赖于所用的表示方法；已设计出的几

种技术有带符号数值、单补码和双补码，但常用的是双补码。

使用双补码所能表示的值的范围依赖于它所用的位的多少。例如，若使用8位，二进制码有256种可能，从00000000到11111111。使用纯二进制数，表示的十进制数范围是0到255；使用双补码，能够表示非零值0到127（00000000到01111111）和负值－128到－1（10000000到11111111）。若使用16位，可能值的范围是从－32768到32768。一般地，如果使用n位表示一个带符号数，则负的最大值是-2^{n-1}，正的极限是$2^{n-1}-1$。

C语言的字符数据类型有时是带符号的，有时是不带符号的。最新C语言的实现支持带符号的和无符号的字符。这意味着把值－1赋给一个字符会被转换成255（假设用双补码）。换句话说，如果字符赋值给整数，计算机并非一定用符号扩展该字符。符号扩展可由一个例子很好地说明。如果一个字符变量有10111110二进制码，它赋值给一个整型变量，则结果可能是00000000 10111110（没进行符号扩展），或者可能是11111111 10111110（进行了符号扩展）。

1.4 预处理、编译和连接

产生一个可执行程序经过下面三步：

- 编辑
- 编译
- 连接

虽然一个系统和另一个系统的具体细节不尽相同（具体例子第14章给出，该章讨论一些实际的系统），编译程序常产生目标代码程序。目标代码程序还不是可执行的，因为它仅包括用户定义的函数的代码；运行程序时所需要的系统例程没被包含进来。要使这个目标代码成为可执行的，它就必须和该程序所用到的系统例程结合或连接起来；这个任务由称之的连接程序或装入程序的程序完成。连接程序把指令代码和系统程序连接起来，当连接完成后，若有没有定义的符号就报告相应的错误（如果引用了一个没有定义的函数或连接到错误的程序库时会发生这样的错误）。

系统程序库提供运行时支持以及对不同操作系统服务的访问——典型的是I／O，但有时是其它服务，如浮点例程。发出操作系统请求的例程通常不是运行时系统的一部分。

一种语言的运行时系统是管理存储区的专门支持的代码，通常称作栈（为子例程调用而设）和堆（为动态存储请求而设）以及运行时“在幕后”通过请求或自动产生而发生的其它事件。这个运行时环境是自我管理并且通常独立于操作系统本身。通常在语言的系统程序库里提供例程允许用户操纵这个环境。

Pascal和C语言都要求运行时支持以便执行递归的过程和函数以及支持那些包含有指针的操作。

在某些语言中（例如C语言）用户可把一个程序分成几个模块。其中每个被分别编译并且必须连接在一起（以及所要求的系统程序库）才产生可执行代码。使用分别编译模块的好处是如果发现了一个错误，只需改变并重新编译不正确的模块而不是整个程序。这个改正过的模块再和其余已经编译过的模块重新连接。

一些程序设计语言支持所谓预处理的特性。它允许用户指明源代码文本中的某个序列被文本中的其它序列代替。这个任务（常称作宏替代）由一个预处理应用程序完成，并且它通

常在编译段以前发生；然而在许多系统中，预处理和编译段串联完成。很多预处理程序所支持的一个相关特性是文件包含。它说明其它文件的文本在编译中作为源程序的一部分而被包含。我们在下节中介绍文件包含的细节。

1.5 包含文件

程序文本通常存放在一个单独的文件中，为了使程序易于修改，它可以分成两个或更多个文件，例如一个文件包括数据结构而另一个包括主程序代码。其中一个文件被当作主文件，它包括有预处理指令，该指令指示在此处所指出的文件在编译时被包含进来，编译程序编译一个程序碰到包含指令时，它打开所指的文件，就象该文件文本出现在程序中一样编译文件文本。到了包含文件的末尾时，编译程序从主文件的断点继续向下编译。当编译完成后，就产生单独一个目标代码文件而不是每个包含文件一个。这样，从连接的观点看，程序包括有包含指令和程序由单独一个文件组成不存在任何差别。

不要把分别编译和包含文件混淆。程序可以被分为分别编译的模块（每个模块编译完成后产生单独一个目标代码文件）每个模块可再被分为几个包含文件（虽然它仍被编译程序当作连续的文本）。

大多数C语言系统支持宏替代和文件包含。我们将在第5章里评述标准的C语言预处理程序。

词汇表

字节：　计算机中字的一部分，通常8位。

连接：　把几个文件的目标代码结合起来，还可能包括一些程序库

宏：　在编译段预处理期间对一个内部过程的扩展。

预处则：　对源文件某些部分的文本替代。

运行时系统：　管理运行时程序请求的支持代码。

第二章　基本数据类型和终端I／O

本章介绍C语言所支持的基本数据类型：int，char，float，以及double。我们介绍C语言赋值号，然后讨论算术表达式，介绍一些终端I／O操作。我们通过讨论类型转换和定义新的数据类型结束本章。

Pascal结构的复习：

这节我们讨论相应C语言结构将表示的那些Paseal的结构。每章开始都有类似的讨论。我们还给出了本章和以后各章用到的一些术语定义。因此这样的复习也是弄懂该章其余部分所要求掌握的。这节的最后我们给出了一个词汇表，它帮助你学习该章的主要部分。

让我们从讨论词法结构开始。Pascal程序是按标准Pascal规则组合在一起的字符序列。这些字符组合构成了不同的符号。一个符号是下列情形之一。

- 运算符，如+
- 送键词，如begin
- 标识符，如temp
- 文字（数字和字符串），如124
- 标号，如LOOP
- 特殊符号，如[
- 注解

Pascal的标识符是由字母和数字组成；它们必须是以字母开头并且不是格感知的。也就是说，标识符temp和TEMP或Temp是一样的。注解以{开始，以}结束或以（*开始以*）结束。在标准Pascal中，注解不许嵌套。

标准Pascal支持四种基本数据类型：integer，char，boolean和real，实数的算术精度依赖于实现，不能由程序员说明。

变量不能在它们说明时初始化。代之的是程序中包含相应的语句完成所希望的初始化。

类型检查是静态的。也就是说，一个表达式的类型通常在程序编译时确定。

类型相容是强制的，也就是说，在一个赋值语句中，左边的类型必须和右边的类型相同，没有自动类型转换。对于输入操作也是一样。唯一例外是整数表达式可以根据需要转换为实数。例如，一个赋值的左边是实数类型，右边的类型是整数，则整数值自动转换为实数值。

一个赋值是一个语句而不是一个表达式，不支持多重赋值（如X：＝y：＝2）

终端I/O由四个复载过程提供，它们是READ，READLN，WRITE和WRITELN，每个都有无限个参数，一个过程称之复载的，如果它接受不同类型的参数，而具体执行的动作由过程根据所说明的参数决定。例如：

READ（X）

是复载的，因为根据X的类型，该过程读入一个整数、一个实数或一个字符值。

下面是终端输入的一般描述。虽然它在某种程度上依赖于系统，但仍然能解释一些重要

的方面。终端输入基本上可按下面的两种之一对待：

·带缓冲的／不带缓冲的

·带回波的／不带回波的

如果终端输入是带缓冲的，则执行Pascal的READ语句时所敲入的信息直到按了回车键后才读进去。这允许你出现错误时编辑你的输入。例如如果要求是：

```
VAR  C, CHAR;
.....
READ  (C)
```

如果你想敲的是b而你却敲了a，你仅需按退格键消除a，输入b。一旦你按了回车键之后，输入就不能再编辑了。

在某些应用里，这种输入方法并不合适。例如，如果你有一个菜单驱动的程序并且要求用户敲入

```
N——eait  new  fill
Q——auit  eaitor
```

象许多基本屏幕的编辑者那样，你可能想不必强迫用户敲了N或Q后再接着输入回车——一个应该做的敲键动作。

是否可能让用这种方式的字符立即读入呢？通常我们用某种方法保存一个内部的输入缓冲区，当Pascal中执行READ（C）时，赋给C的值是从这个缓冲区内得到的。如果缓冲区为空，你可以送入一行新的字符，它放在这个缓冲区里；在你按了回车键以前没有任何东西从缓冲区读出。这种类型的输入称为带缓冲的输入。要让字符立即读入，就需要不带缓冲的或者称为列的输入方式。这种方式不使用缓冲区，每当产生一个输入请求时，字符就直接从所敲的键盘上取得。

交互式输入的另一个方面是所敲字符是否在屏幕上回波显示，Pascal的某些实现提供允许不带回波输入的专门例程。

输入过程（WRITE和WRITELN）可以被用作格式化的或非格式的输出。例如：

```
WRITE(X)          ——非格式化的
WRITE(X, 10)      ——格式化的
```

词汇表

自动类型转换　一个表达式的类型被编译程序改变以满足类型相容要求。

带缓冲I／O　I／O操作通过内部缓冲区完成。

复载过程　指这样的过程，它能接受不同类型的参数，该过程的确切定义由参数的类型决定。

静态类型　当程序编译时就能确定的类型。

C语言结构

2.1　语法结构

C语言的语法结构和Pascal类似，由于C语言是自由格式的，也就是说一个程序被看作是由白空字符分开的符号序列（空格、制表换页和新行字符集中叫做白空字符）。这些标识不必象某些语言那样必须出现在指定的列的范围内或进行其它的格式变换，附录A列出了C

语言中定义的关键词和各种专门字符。

C语言标识符建立的规则和Pascal一样，但它们是格感知的。例如，Temp和TEMP是两个不同的标识符。这种格感知性对于C语言定义的关键词也适用，也即它们必须以小写字母输入，否则它们不被认为是关键词。另外C语言允许开头、结尾和中间使用下划线。一般地用户应当避免开头是下划线的标识符，因为许多系统程序库为其特殊目的而使用这种名字。

标识符可以是任意长度，但在标准C语言中只有前八个字符有意义。然而有许多实现把多于八个的字符也当作有意义的。从程序的可移植性来讲，应当遵循八个这一限制。

C语言定义四种文字，常量）：

- 整数
- 长整数
- 字符
- 双精度数

我们将在2.4节中详细讨论它们

C语言的注解以／*开始，以*／结束；例如：

```
/*this is a comment */
```

注解可以出现在白空字符可合法出现的任何地方，标准C语言不允许注解的嵌套，例如：

```
/*comment and/*nested comment*/*/
```

是非法的。

2.2 基本数据类型、赋值

变量说明

C语言提供的几个基本数据类型是：char，int，long int，short int，float和double。和Pascal不同的是，C语言不提供内部布尔类型。取而代之，布尔值用整数表示；值为0代表假，非0代表真。

类型标识符和Pascal中的相似：

Pascal	C
INTEGER	int long int short int
CHAR	char
REAL	float double

int可以指定是long或short，short int等价于short，long int等价于long。并且，char，int，short和long可指定是不带符号的。省缺时整数是带符号的，字符是带符号的或者不依赖于实现。

类型说明short表示通常比一般整数的有意义位少的整数，有意义数字的实际个数依赖于具体实现。类型说明long表示通常有更多有意义位的整数，虽然用来存放一个长整数的实际字节数也依赖于具体实现。

double类型变量具有两倍于float类型变量的有意义位数，当然也要求更多的存贮器。它们常称为双精度实数

C语言保证存贮一个对象所需的字节数遵守下面的规则：

char的大小是一个存贮单位

short int的大小<=int的大小<long int的大小

unsigned int的大小=int的大小

float的大小<=double的大小

一个存贮单位的精确意义依赖于实现，传统地，一个存贮单位叫做一个字节，这也是我们在本书中所用的术语。对于分配给short、int和long变量相同字节数的实现来说，前面的定义仍然正确。

说明变量时，C语言的语法和Pascal所用的相反，并且C语言没有使用Pascal关键词VAR

Pascal	C
VAR i : INTEGER; c, d : CHAR; x, y : REAL;	int i; char c, d; float x, y;

赋值

左值是一个可被翻译作一个地址的表达式；例如变量是左值而常量不是。直观地说，左值就是可出现在赋值左边的表达式。

在Pascal中赋值是语句，而这里赋值是表达式。它们的语法是：

Pascal	C
l-value := *expression*	*l-value* = *expression*

C语言中的赋值表达式可以通过跟在后面的分号变成一个语句，例如：

Pascal	C

i := 2;
c := 'a';
x := x + 2 * x;
	i = 2;
	c = 'a';
	x = x + 2 * x;

考虑赋值

X=X+2

这里的X是一个左值。这个赋值左边出现的X表示一个存贮器位置的地址，比如说是L，而在右边出现的X表示的当前值。这个赋值的执行结果是值X+2存放在存贮器位置L。这一描述也适用于Pascal中的赋值语句。

C语言的赋值不用冒号。或许更麻烦的是相等运算符。Pascal中是单等号=，而在C语言中是双等号==。Pascal和C语言的这些微妙差别可能引起许多费解。

由于C语言的表达式不是一个语句而是一个表达式，所以它必须产生一个值。一个赋值所返回的值就是右边的值。赋值表达式的类型是左边的类型；例如：

i=8——整数表达式等于8并且把值8赋给变量i

因为一个赋值是一个表达式，所以有可能多重赋值；例如：

X=y=3

由于赋值运算符=从右到左结合，前面的表达式等价于