# 第一章 导言

c语言中一个通用的实例：在允许使用某种类型变量值的任何场合，都可以使用该类型的更复杂的表达式。

标准库提供的输入输出模型非常简单。无论文本从何处输入、输出到何处，其输入输出都是按照字符流的方式处理。文本流是由多行字符构成的字符序列，而每行字符则由0个或者多个字符组成，行末是一个换行符。标准库负责使每个输入输出流都能遵守这一模型。

字符在键盘、屏幕或者其他任何地方无论以什么形式表现，它在机器内部都是以为位模式存储的。

EOF定义在头文件<stdio.h>中，是一个整形数。它与任何char类型的值都不相等。

赋值操作是一个表达式，并且具有一个值，就是赋值后左边变量保存的值。

出现在main函数之前的函数声明称为函数原型。函数原型中的参数名是可选的。

printf函数中的格式规范%s规定，对应的参数必须是以'\0'结尾形式的字符串。

定义表示创建变量会分配存储单元，而声明指的是说明变量的性质，但并不分配存储单元。

# 类型、运算符与表达式

## 数据类型及长度

C语言只有char，int，float，double四种基本数据类型。

Int通常代表特定机器中整数的自然长度。Short类型通常为16位，long类型通常为32位，int类型可以为16位或者32位。各编译器可以根据硬件特性自主选择合适的类型长度，但要遵循限制：short和int类型至少为16位，long类型至少为32位，short类型不得长于int类型，int类型不得长于long类型。

## 常量

字符常量’\0’表示值为0的字符，也就是空字符（null）。

常量表达式是仅仅只包含常量的表达式。这种表达式在编译时求值，而不在运行时求值。

编译时可以将多个字符串常量直接连接起来，如”hello” “ world”等价于“hello world”。

C语言对字符串的长度没有限制，但是必须扫描完整个字符串后才能确定字符串的长度。

枚举是一个常量整形值的列表。

在没有显示说明的情况下，enum类型的第一个枚举名的值为0，第二个为1，以此类推。如果只指定了部分枚举名的值，那么未指定值的枚举名将依着最后一个指定值向后递增。

## 声明

const限定符指定变量的值不能被修改。

## 算数运算符

在有负操作数的情况下，整数除法截取的方向以及取模运算结果的符号取决于具体机器的实现，这和处理上溢或下溢的情况是一样的。

## 类型转换

C语言的定义保证了机器的标准打印字符集中的字符不会是负值，因此，在表达式中这些字符总是正值。但是，存储在字符变量中的位模式在某些机器中可能是负的。为了保证程序的可移植性，如果要在char类型的变量中存储非字符数据，最好指定signed或unsigned限定符。

使用float类型主要是为了在使用较大的数组时节省存储空间，有时也为了节省机器执行时间（双精度运算很耗时）。

当表达式中包含unsigned类型的操作数时，转换规则要复杂一些。主要原因在于，带符号值与无符号值之间的比较运算是与机器相关的，因为它们取决于机器中不同整数类型的大小。比如-1L>1UL，因为-1L将被提升为unsigned long类型（符号位的1），成为更大的正数。

无论是否进行符号扩展，字符型变量都将被转换为整形变量。

由于函数调用的参数是表达式，所以在把参数传递给函数时也可能进行类型转换。

强制类型转换只是生成一个指定类型的值，没有修改原来的变量值。

## 赋值运算符与表达式

赋值表达式的类型是左操作数的类型，值是赋值操作完成之后的值。

## 运算符优先级与求值次序

同大多数语言一样，C语言没有指定同一运算符中多个操作数的计算顺序（&&,||,?:和，运算符除外）。

同样，C语言也没有指定函数各参数的求值顺序。

# 控制流

## Goto语句与标号

C语言提供了可随意滥用的goto语句以及标记跳转位置的标号。

Goto语句最常见的用法是终止程序在某些深度嵌套的结构中的处理过程，例如一次跳出两层或者多层循环。

标号的命名同变量命名的形式相同，标号的后面要紧跟着一个冒号。标号可以位于对应的goto语句所在函数的任意语句的前面。标号的作用域是整个函数。

建议尽量少的使用goto语句。

# 函数与数据结构

## 函数的基本知识

如果函数定义中省略了返回值类型，则默认为int类型。

## 返回非整形值的函数

如果先前没有声明过的一个名字出现在某个表达式中，并且其后紧跟一个左圆括号，那么上下文就会认为该名字是一个函数名字，该函数的返回值将被假定为int类型，但是上下文并不会对其参数作任何假设。

## 外部变量

外部变量和函数具有下列性质：通过同一个名字对外部变量的所有引用（即使这种引用来自于单独编译的不同函数）实际上都是引用同一个对象（标准中把这一性质称为外部链接）。

程序中经常会出现这样的情况：程序不能确定它已经读入的输入是否足够，除非超前多读取一些输入。但是每当程序多读入一个字符时，就把它压回到输入中，对代码其余部分而言就好像没有读入该字符一样。

标准库中提供了函数ungetc，它将一个字符压回到栈中。

## 作用域规则

如果要在外部变量的定义之前使用该变量，或者外部变量的定义与变量的使用不在同一个源文件中，必须在相应的变量声明中强制性的使用关键字extern。

将外部变量的声明与定义严格区分开很重要。变量声明用于说明变量的属性（主要是变量的类型），而变量定义除此之外还将引起存储器的分配。

外部变量的定义中必须指定数组的长度，但extern声明则不一定要指定数组的长度。

外部变量的初始化只能出现在其定义中。

## 静态变量

用static声明限定外部变量和函数，可以将其后声明的对象的作用域限定为被编译源文件的剩余部分。通过static限定外部对象，可以达到隐藏外部对象的目的。

Static类型的函数除了对该函数声明所在的文件可见外，其他文件都无法访问。

Static类型的内部变量是一种只能在某个特定类型函数中使用但一直占据内存空间的变量。

## 寄存器变量

Register声明告诉编译器，它所声明的变量在程序中使用频率较高。其思想是将register变量放在机器的寄存器中。但编译器可以忽略此选项。

无论寄存器变量实际上是不是存放在寄存器中，它的地址都是不能访问的。

## 初始化

在不进行显式初始化的情况下，外部变量和静态变量都将被初始化为0，而自动变量和寄存器变量的初值则没有定义（即初值为无用的信息）。

对于外部变量和静态变量来说，初始化表达式必须是常量表达式，且只初始化一次（概念上将是在程序开始执行前初始化）。

如果初始化表达式的个数比数组元素少，则对外部变量、静态变量和自动变量来说，没有被初始化的元素将被初始化为0，如果初始化表达式的个数比数组元素多，则是错误的。

## 递归

递归并不节省存储器的开销，因为递归调用过程中必须在某个地方维护一个存储处理值的栈。递归的执行速度并不快，但递归代码比较紧凑，并且比相应的非递归代码更易于编写与理解。

## C预处理器

从概念上讲，预处理器是编译过程中单独执行的第一个步骤。

文件包含指令（即#include指令）的行将被替换为由文件名指定的文件的内容。如果文件名用引号引起来，则在源文件所在的位置查找该文件；如果在该位置没有找到文件，或者如果文件名是用尖括号<>括起来的，则将根据相应的规则查找该文件，这个规则同具体的实现有关。

宏定义（即#define指令）是一种最简单的宏替换：后续所有出现名字记号的地方都将被替换为替换文本。

宏替换只对记号进行，对括在引号中的字符串不起作用。

宏定义也可以带参数，这样可以对不同的宏调用使用不同的替换文本。

<stdio.h>头文件中有一个很实用的例子：getchar与putchar函数在实际中常常被定义为宏，这样可以避免处理字符时调用函数所需的运行时开销。

形式参数不能用带引号的字符串替换。但是如果在替换文本汇总，参数名以#作为前缀则结果将被扩展为由实际参数替换该参数的带引号的字符串。

还可以使用条件语句对预处理本身进行控制，这种条件语句的值是在预处理执行的过程中进行计算。

# 指针与数组

## 指针与地址

通常情况下，机器的一个字节可以存放一个char类型的数据，两个相邻的字节存储单元可存储一个short类型的数据，而4个相邻的字节存储单元可存储一个long类型的数据。指针是能够存放一个地址的一组存储单元（通常是2个或4个字节）。

地址运算符&只能应用于内存中的对象，即变量和数组元素。

指针只能指向某种特定类型的对象，也就是说，每个指针都必须指向某种特定的数据类型。（一个例外情况是指向void类型的指针可以存放指向任何类型的指针，但它不能间接引用其自身）

指针也是变量，在程序中可以直接使用。

## 指针与数组

一般来说，用指针编写的程序比用数组下标编写的程序执行速度快，但理解起来稍微困难一些。

如果指针pa指向a[0],那么\*（pa+1）引用的是数组元素a[1]的内容，pa+i是数组元素a[i]的地址，\*（pa+i）引用的是数组元素a[i]的内容。无论数组中元素的类型或数组长度是什么，结论都成立。

根据定义，数组类型的变量或表达式的值是该数组第0个元素的地址。对数组元素a[i]的引用也可以写成\*（a+i）。

如果pa是一个指针，那么在表达式中也可以在它的后面加下标。pa[i]与\*(pa+i)是等价的。简而言之，一个通过数组和下标实现的表达式可等价的通过指针和偏移量实现。

数组名和指针之间有一个不同之处。指针是一个变量，但数组名不是变量。

在函数定义中，形式参数char s[] 和char \*s是等价的。

## 地址算数运算

C语言中的地址算数运算方法是一致且有规律的，将指针、数组和地址的算数运算集成在一起的该语言的一大优点。

和其他类型变量一样，指针也可以初始化。对指针有意义的初始化值只能是0或者是表示地址的表达式，对后者来说，表达式所代表的地址必须是在此前已定义的具有适当类型的数据的地址。

C语言保证，0永远不是有效的数据地址。

指针与整数之间不能相互转换，但0是唯一的例外：常量0可以赋值给指针，指针也可以和常量0进行比较。程序中常用符号常量NULL来代替常量0，这样便于更清晰的说明常量0是指针的一个特殊值。符号常量NULL定义在标准头文件<stddef.h>中。

在某些情况下对指针可已经进行比较运算。比如指针p，q指向同一个数组的成员，那么它们之间就可以进行关系比较运算，如果p指向的数组元素的位置在q指向的数组元素位置之前，那么p<q为真。任何指针与0进行相等或不相等的比较运算都有意义。但是，指向不同数组的元素的指针之间的算数或比较运算没有意义。（有一个特例：指针的算数运算中可以使用数组的最后一个元素的下一个元素的地址）

指针可以和整数进行相加或者相减运算。p+n 表示指针p当前指向的对象之后第n个对象的地址。无论指针p指向的对象是何种类型，该结论都成立。计算p+n时，n将根据p指向的对象的长度按比例缩放，而p指向的对象的长度取决于p的声明。

指针的减法运算也是有意义的。如果p和q指向相同数组中的元素，且p<q，那么q-p+1就是位于p和q指向的元素之间的元素的数目。

有效的指针运算包括相同类型指针之间的赋值运算；指针同整数之间的加法或减法运算；指向相同数组中元素的两个指针间的减法或比较运算；将指针赋值为0或指针与0之间的比较运算。其他所有形式的指针运算都是非法的。

## 字符指针与函数

例如printf(“hello world”); 这样的语句中，实际上是通过字符指针访问该字符串的。Printf语句接收的是一个指向字符数组第一个字符的指针。也就是说，字符串常量可以通过一个指向其第一个元素的指针访问。

C语言中没有提供将整个字符串作为一个整体进行处理的运算符。

字符数组中的单个字符可以修改，字符数组始终指向同一个存储位置。如果一个字符指针初值指向一个字符串常量，这个指针可以被修改指向其他地址，但如果尝试修改字符串的内容，是没有意义的。

## 多维数组

数组元素按行存储。

如果将二维数组作为参数传递给函数，那么在函数的参数声明中必须指明数组的列数。

二维数组作为函数参数可以写作：f(int a[2][2]),f(int a[][2]), f(int (\*a)[2])。第三种声明形式表示参数是一个指针，它指向具有2个整形元素的一维数组。

## 指针与多维度数组

指针数组的一个重要优点在于，数组的每一行长度可以不同。

## 命令行参数

在支持C语言的环境中，可以在程序开始执行时将命令行参数传递给程序。调用主函数main时，它带有两个参数。第一个参数（习惯上成为argc，用于参数计数）的值表示运行程序时命令行中参数的个数；第二个参数（成为argv，用于参数向量）是一个指向字符串数组的指针，其中每个字符串对应一个参数。

按照C语言的约定，argv[0]的值是启动该程序的程序名，因此argc的值至少为1。

ANSI标准要求argv[argc]的值必须为一个空指针。

UNIX系统中的C语言程序有一个公共的约定：以负号开头的参数表示是一个可选标志或参数。

## 指向函数的指针

在C语言中，函数本身不是变量，但是可以定义指向函数的指针。

任何类型的指针都可以转换为void \* 类型，并且在将它转换回原来的类型时不会丢失信息。

Int (\*comp)(void \*, void \*)；这样的声明表明comp是一个指向函数的指针，\*comp代表一个函数。

# 结构

## 结构的基本知识

结构声明时，struct后面的名字是可选的，称为结构标记。结构标记用于为结构命名。

Struct {...} x,y,z;这种方式的声明与一般声明 int x,y,z; 从语法角度上具有相似的意义。

## 结构与函数

结构的合法操作只有几种：作为一个整体复制和赋值，通过&运算符取地址，访问其成员。

结构之间不可以进行比较。

结构类型的参数和其他类型的参数一样，都是通过值传递的。

结构指针的使用频率非常高，为了使用方便，C语言提供了另一种简写方式（相比于(\*p).x的方式）。假定p是一个指向结构的指针，可以用 *p->结构成员* 这种形式引用相应的结构成员。

## 结构数组

C语言提供了一个编译时一元运算符sizeof，可用来计算任一对象的长度。它的值等于指定对象或类型占用的存储空间字节数。（严格的说，sizeof的返回值是无符号整形值，类型为size\_t，该类型在头文件<stddef.h>中定义）

条件编译语句#if中不能使用sizeof，因为预处理器不对类型名进行分析。但预处理器并不计算#define语句中的表达式，因此，在#define中使用sizeof是合法的。

## 指向结构的指针

千万不要以为结构的长度等于各成员长度的和。因为不同的对象类型有不同的对齐要求，所以，结构中可能会出现未命名的空穴(hole)。

## 类型定义（typedef）

C语言提供了一个称为typedef的功能，它用来建立新的数据类型名，例如声明

typedef int Length;

将Length定义为与int具有同等意义的名字。类型Length可用于类型声明、类型转换等，和类型int完全相同。

从任何意义上讲，typedef声明并没有创建一个新类型，它只是为某个已存在的类型增加了一个新的名称而已。

实际上，typedef类似于#define语句，但由于typedef是由编译器解释的，因此它的文本替换功能要超过预处理器的能力。

除了表达方式更简洁之外，使用typedef还有另外两个重要原因。首先，它可以使程序参数化，以提高程序的可移植性。第二个作用是为程序提供更好的说明性。

## 联合

联合是可以（在不同时刻）保存不同类型和长度的对象的变量，编译器负责跟踪对象的长度和对齐要求。联合提供了一种方式，以在单块存储区中管理不同类型的数据，而不需要在程序中嵌入任何同及其有关的信息。

联合变量读取的类型必须是最近一次存入的类型。

访问联合中成员的方式和访问结构的方式相同。

实际上，联合就是一个结构，它的所有成员相对于基地址的偏移量都为0，此结构空间要大到能够容纳最“宽”的成员，并且，其对齐方式要适合于联合中所有类型的成员。

联合只能用其第一个成员类型的值进行初始化。

## 位字段

C语言提供了一种直接定义和访问一个字中位字段的能力，而不需要通过按位逻辑运算符。位字段（bit-field），或简称字段，是“字”中相邻位的集合。“字”（word）是单个的存储单元，它同具体的实现有关。

字段的所有属性几乎都同具体的实现有关。

# 输入与输出

## 标准输入输出

命令行 prog <infile 将使得程序prog从输入文件infile（而不是从键盘）中读取字符。实际上，程序prog本身并不在意输入方式的改变，并且，字符串“<infile”也并不包含在argv 的命令行参数中。

当文件名用一对尖括号<>括起来时，预处理器将在由具体实现定义的有关位置中查找指定的文件。

## 格式化输出--printf函数

格式字符串包含两个类型的对象，普通字符和转换说明。

每个转化说明都有一个百分号字符（即%）开始，并以一个转换字符结束。在字符%和转换字符之间可能包含以下部分：

负号：转换参数以左对齐方式输出。

数：指定最小字段宽度。

小数点：用于将字段宽度和精度分开。

数：用于指定精度，即指定字符串中要打印的最大字符数、浮点数小数点后的位数、整型最少输出的数字数目。

字母h或l：h表示整数作为short类型打印，l表示整数作为long类型打印。

转换字符有d,i,o,x,X,u,c,s,f,e,E,g,G,p,%。

在转换说明中，宽度或精度可以用星号\*表示，这时，宽度或精度的值通过转换下一参数（必须为int）来计算。

Sprintf函数和printf函数一样，但它将输出结果存放到string中。

## 变长参数表

函数printf的正确声明形式为：

int printf(char \*fmt, ...)

其中，省略号表示参数表中参数的数量和类型是可变的。省略号只能出现在参数表的尾部。

标准头文件<stdarg.h>中包含一组宏定义，它们对如何遍历参数表进行了定义，该头文件的实现因不同的机器而不同，但提供的接口是一致的。

va\_list类型用于声明一个变量，该变量将依次引用各参数。

宏va\_start将va\_list类型的变量初始化为指向第一个无名参数的指针。在使用va\_list类型变量之前，该宏必须被调用一次。参数表必须至少包含一个有名参数，va\_start将最后一个有名参数作为起点。

每次调用va\_arg，该函数都将返回一个参数，并将va\_list类型的变量指向下一个参数。va\_arg使用一个类型名来决定返回的对象类型、指针移动的步长。最后，必须在函数返回之前调用va\_end，以完成一些必要的清理工作。

## 格式化输入--scanf函数

scanf函数扫描完其格式串，或者碰到某些输入无法与格式控制说明匹配的情况时，该函数将终止，同时，成功匹配并赋值的输入项的个数将作为函数值返回。如果到达文件结尾，该函数将返回EOF。返回EOF和返回0是不同的，0表示下一个输入字符与格式串中的第一个格式说明不匹配。

还有一个输入函数sscanf，用于从一个字符串（而不是标准输入）中读取字符序列。

Scanf函数忽略格式串中的空格和制表符。此外，在读取输入值时，它将跳过空白符。如果要读取格式不固定的输入，最好每次读入一行，然后再用sscanf将合适的格式分离出来读入。

Scanf函数可以和其他输入函数混合使用。无论调用哪个输入函数，下一个输入函数的调用将从scanf没有读取的第一个字符处开始读取数据。

## 文件访问

在读一个文件之前，必须通过库函数fopen打开该文件。Fopen用文件名与操作系统进行某些必要的连接和通信，并返回一个可以用于文件读写操作的指针。

该指针称为文件指针，指向一个包含文件信息的结构，这些信息包括：缓冲区的位置、缓冲区中当前字符的位置、文件的读或写状态、是否出错或是否已经到达文件结尾等等。<stdio.h>中定义了包含这些信息的结构FILE。

FILE像int一样是一个类型名，而不是结构标记。是通过typedef定义的。

如果发生错误，fopen将返回NULL。

文件打开后，有多种方法可以对文件进行读写，其中，getc和putc最简单。Getc从文件中返回下一个字符，参数是文件指针。

getc函数函数返回文件指针指向的输入流中的下一个字符。如果到达文件尾或出现错误，返回EOF。

putc输入函数将第一个参数（字符）写入到第二个参数（文件指针）指向的文件中，并返回写入的字符。如果发生错误则返回EOF。getc和putc是宏而不是函数。

启动一个C语言程序时，操作系统环境负责打开3个文件，并将这3个文件的指针提供给该程序。这3个文件分别是标准输入、标准输出和标准错误，相应的文件指针分别为stdin，stdout和stderr，他们在<stdio.h>中声明。

对于文件的格式化输入或输出，可以使用函数fscanf和fprintf。与scanf和printf的唯一区别是它们的第一个参数是文件指针。

文件指针stdin与stdout都是FILE\*类型的对象，但它们是常量。

## 错误处理--stderr和exit

Fprintf函数产生的诊断信息将会输出到stderr上。

标准库函数exit将终止调用程序的执行。按照惯例，返回值0表示一切正常，非0返回值通常表示出现了异常情况。exit为每个已经打开的输出文件调用fclose函数，以将缓冲区中的所有输出写到相应的文件中。

使用exit有一个优点，它可以从其他函数中调用。

如果流fp（函数ferror的FILE类型的指针参数）中出现错误，则函数ferror返回一个非0值。

函数feof(FILE\*)与ferror类似。如果指定的文件到达文件结尾，它将返回一个非0值。

## 行输入和行输出

标准库函数fgets：

char \*fgets(char \*line, int maxline, FILE \*fp)

从fp指向的文件中读取下一个输入行（包括换行符）,并存放在line中，最多可读取maxline-1个字符。读取的行将以’\n’结尾保存到数组中。通过返回line，如果遇到文件结果或发生错误，返回NULL。

输入函数fputs将一个字符串（不需要包含换行符）写入到一个文件中，如果发生错误，返回EOF，否则返回一个非负值。

库函数gets和puts的功能和fgets和fputs类似，但是它们是对stdin和stdout进行操作。另外，gets函数在读取字符串时将删除结尾的换行符（‘\n’），而puts函数在写入字符串时将在结尾添加一个换行符。

ANSI标准规定，ferror在发生错误时返回非0值，而puts在发生错误时返回EOF，其他情况返回一个非负值。

## 其他函数

### 字符串操作函数

strcat(s, t) 将t指向的字符串连接到s指向的字符串的末尾

strncat(s, t, n) t指向的字符串的n个字符连接到s的末尾

strcmp(s, t) 根据字符串值比较大小

strnncmp(s, t, n) 只在前n个字符中比较

strcpy(s, t) t复制到s

strncpy(s, t, n) t中前n个字符复制到s

strlen(s, t)

strchr(s, c) s指向的字符串中查找c，返回第一此出现的位置的指针或者NULL

strrchr(s, c) 返回最后一次出现的位置的指针或NULL

### 字符类别测试和转换函数

isalpha(c) 是字母返回非0值，否则返回0

Isupper(c)

islower(c)

isdigit(c) 是数字返回非0值，否则返回0

isalnum(c) 是字母或数字返回非0值，否则返回0

isspace(c)

toupper(c)

tolower(c)

### Ungetc函数

int ungetc(int c, FILE \*fp)

将字符c写回到文件中，成功返回c，否则返回EOF。每个文件只能接收一个回写字符。

### 命令执行函数

system(char \*s)函数执行包含在字符串s中的命令，然后继续执行当前程序。S的内容在很大程度上和所用的操作系统有关。

### 存储管理函数

函数malloc和calloc用于动态的分配存储块。

void \*malloc(size\_t n)

分配成功时，malloc函数返回一个指针，指向的空闲空间足以容纳n个指定长度对象的数组，否则返回NULL。存储空间被初始化为0。

free(p)函数释放p指向的存储空间，p是通过调用malloc或calloc函数得到的指针。存储空间的释放顺序没有限制。如果释放的不是通过malloc和calloc得到的指针所指向的存储空间，会是一个严重错误。

使用已经释放的空间也是错误的。

### 随机数发生器函数

函数rand()生成介于0和RAND\_MAX之间的为随机整数序列。RAND\_MAX是在头文件<stdlib.h>中定义的符号常量。

函数srand()设置rand函数的种子数。

# UNIX系统接口

## 文件描述符

UNIX操作系统中，所有的外围设备都被看做是文件系统中的文件，因此，所有的输入输出都要通过读文件或写文件完成。

通常情况下，在读写文件之前，必须先通知系统，这个过程为*打开文件*。如果一切正常，操作系统将向程序返回一个小的非负整数，称为*文件描述符*。任何时候对文件的输入输出都是通过文件描述符标识文件。系统负责维护已打开文件的所有信息，用户程序只能通过文件描述符引用文件。

因为大多数输入输出都是通过键盘和显示器实现的，所以UNIX做了特别的安排。当shell运行一个程序时，将打开3个文件，文件描述符分别是0,1,2，依次表示标准输入，标准输出和标准错误。

任何情况下，文件赋值的改变都不是由程序完成的，而是由shell完成的。

## 低级IO--read和write

输入与输出是通过read和write系统调用实现的。C语言程序中，可以通过函数read和write访问这两个系统调用。两个函数都有三个参数，第一个是文件描述符，第二个是程序中存放读写的数据的字符数组，第三个是要传输的字节数。

两个调用返回实际传输的字节数。读文件时，函数的返回值可能会小于请求的字节数。返回值0表示到达文件结尾；-1表示发生错误。写文件时返回的是实际写入的字节数，返回值与请求数不等，说明发生错误。

一次调用中，读出或写入的数据的字节数可以为任意大小。最常用的值为1，或是类似于1024或4096这样的与外围设备的物理块大小相对应的值。值大的话效率更高，因为系统调用次数少。

如果要在包含头文件<stdio.h>的情况下编译自定义的getchar函数，就需要用#udef预处理指令取消名字getchar的宏定义，因为在头文件中，getchar是以宏方式实现的。

## open、creat、close和unlink

处理默认的标准输入、标准输出和标准错误文件外，其他文件都必须在读或写之前显式地打开。系统调用open和create用于实现该功能。

open与fopen类似，但是open返回一个文件描述符，仅仅是一个int类型的数值，而fopen返回一个文件指针。发生错误时，open将返回-1。

使用open打开一个不存在的文件将导致错误。可以用create系统调用创建新文件或覆盖已有的旧文件。

如果create成功创建了文件，它将返回一个文件描述符，否则返回-1.如果此文件已存在，creat将把该文件的长度截断为0，从而丢弃原来已有的内容。使用create创建一个已存在的文件不会导致错误。

一个程序同时打开的文件数是有限的（通常为20）。相应的，如果一个程序需要同时处理许多文件，那么它必须重用文件描述符。函数close(int fd)用来断开文件描述符和已打开文件之间的连接，并释放此文件描述符，以供其他文件使用。Close函数与标准库中的fclose函数相对应，但它不需要清洗(flush)缓冲区。如果程序通过exit函数退出或从主程序中返回，所有打开的文件将被关闭。

函数unlick(char \*name)将文件name从文件系统中删除，对应于标准库函数remove。

## 随机访问--lseek

系统调用lseek可以在文件中任意移动位置而不实际读写任何数据：

long lseek(int fd, long offset, int origin)

将文件描述符为fd的文件的当前位置设置为offset，offset是相对于origin指定的位置而言的。随后进行的读写操作将从此位置开始。Origin的值可以是0,1,2，分别指定offset从文件开始、从当前位置和从文件结束出开始算起。

使用lseek系统调用时，可以将文件视为一个大数组，其代价是访问速度会慢一点。

lseek系统调用返回一个long类型的值，此值表示文件的新位置，若发生错误，则返回-1。标准库函数fseek与系统调用lseek类似，但是fseek第一个参数是FILE \*类型，且在错误时返回一个非0值。