Pascal

软件工具

邹广然　刘实　翟清武编译

内蒙古大学出版社

Pascal 软 件 工 具

(美) Brian W.Kernighan P.J.Plauger 著

邹广然 刘 实 翟清武 编译

内蒙古大学出版社

1989.1 呼和浩特

内 容 简 介

本书主要包括两部分内容：1.丰富的Pascal语言写的软件工具；2.通过介绍这些工具，介绍了如何用结构化程序设计思想，逐步求精、自顶向下设计方法，编写具有良好可移植性、可扩充性和良好用户界面的软件工具。

前五章给出了一些基本的工具，包括一些过滤程序、文件处理、排序和正文模式识别。后面三章，给出了功能较强的正文编辑、排版程序和宏处理程序。本书可供计算机应用人员、计算机软件人员阅读和参考；也可供高等院校有关专业师生教学使用。

Pascal 软 件 工 具

Pascal RUANJIAN GONGJU

(美) Brian W. Kernighan、P.J.Plauger 著

邹广然 刘 要 编译

内蒙古大学出版社出版发行

内蒙古大学印刷厂印刷

开本 787×1092/16 印张 20.75 字数 400 000

印数 0 001—4 000

1989年1月第1版 1989年1月第1次印刷

ISBN 7-81015-057-X

TP·1 定价：4.90元

编译者序

随着计算机技术的发展，计算机应用日益广泛，特别是 *IBM-PC/XT*、国产长城系列微机的日趋普及，计算机已深入到了各个领域。众多的计算机应用、维护、开发人员，面对着丰富的软件工具和应用软件，可能都想过：这些软件工具是如何开发的？怎样才能编写出具有良好的可移植性、可扩充性和良好用户界面的软件工具？在编写软件工具时，应注意到哪些问题？有哪些理论、技巧和方法？

本书通过大量实用的完整程序实例，介绍了如何用结构化程序设计思想，自顶向下、逐步求精设计方法，编写具有良好的可移植性、可扩充性和良好用户界面的软件工具。书中提供的软件工具有：过滤程序、文件处理、排序、正文模式处理和功能较强的正文编辑、排版程序、宏处理程序。**这本书本身就是采用类似于书中介绍的软件工具，由计算机编排的。**同时，给出了详细的功能说明，介绍了上述软件工具的设计实现方法。它是计算机应用人员、软件人员开发软件工具的很好的参考书，也可供计算机软件专业、计算机应用专业的师生教学使用。

本书根据美国 *Bell* 实验室的 *Brian W. Kernighan* 和 *P. J. Plauger*：《*Software Tools in Pascal*》编译。各章编译者如下：第一、二章 **邹广然**，刘实整理；第三章 **邹广然**，翟清武整理；第四、五章 **翟清武**；第六、七、八章 **刘实**；附录 **翟清武、刘实**。编译英文教材，是我们的第一次尝试，缺点、错误一定很多，恳请有关前辈、专家多提宝贵意见。本书油印本，作为内蒙古大学本科生高年级的教材，经受了实践的检验。

在本书出版之际，我们深切缅怀已故原中国计算机学会理事、内蒙古计算机学会理事长、内蒙古大学计算机科学系副教授、我们敬爱的导师**邹广然**先生。

本书编译过程中，得到了内蒙古大学计算机科学系**谢育先**副教授、**叶新铭**副教授，内蒙古电子计算中心**朱玉芳**副研究员、**王升亮**副研究员、**王侯**副研究员的热情帮助和支持。

本书采用了内蒙古大学计算机科学系的正文编辑软件工具、《星汉排版系统》、《科印排版系统》。当我们坐在计算机前，以计算机做为"笔"和"纸"，用类似于书中介绍的软件工具，编写我们的书稿的时候，我们得到了一种美的享受。我们的同事和校友，先后有 20 多人，帮助我们录入了书稿。内蒙古电子计算中心**张主**工程师、**满都拉**助理工程师帮助我们进行了书稿的计算机排版。

本书能够正式出版，是和内蒙古大学出版社**巴特尔**副社长、**戴其芳**副总编的热情帮助和支持分不开的。是他们鼓励、支持我们，继续采用计算机编排系统，使得本书在最短的时间内和读者见面。在此一并表示我们深深的谢意。

内蒙古大学计算机科学系　刘　实
内蒙古电子计算中心　翟清武

1988.10.15

引　言

在本书中，我们准备讨论两件事——如何书写形成良好工具的程序，以及在此过程中如何很好地进行程序设计。

这里，我们所说的工具是什么意思呢?假如我们有一个 *5000* 行的 *Pascal* 程序，需要找出所有出现变量 *time* 的地方,以便安全地把它由整型变为实型，你打算怎么实现呢?

一种可行的方法是用红笔把所有出现 *time* 的地方都作上标记。在近百页的计算机纸上想看出标记错误来，那简直是不可想象的。这是一种无头脑的做法，是一种既令人生厌又紧张的工作。出错机会也非常多。即使你已找出了所有的 *time*，也仅仅作了一点工作，因为机器是不能阅读红色标记的。

另一种方法是写一个简单程序，让它找出所有含有标识符 *time* 的那些行。这是一种改进。因为用程序作这种事比用手工的办法既快，又准确。缺点是这个程序太专用了，只能由作者使用一次，然后收起来，忘掉了，其它人无法使用。即使某些东西非常相似，也不得不为每种新的应用重新投资。

找出 *Pascal* 程序中的 *time*。仍是一般问题中的特殊情况，这个一般问题就是找出模式(*pattern*)。今天想要参考 *time* 的人，明天可能参考 *readln*，后天参考 *writeln*，下周可能参考一些不相关正文中的另一完全不同的模式了。如果用红笔作记号将永无休止。解决这个一般问题的方法是，给出一个通用的查找程序 *find*。它能找出指定的模式，并打印出它们所在的行。然后任何人都可通过

find pattern

命令使用它工作。*find* 就是一个工具。它使用了这台机器，它解决了一个一般问题，而不是一种特殊情况;它很容易使用，人们不用再去建立各自的查找程序，使用它就可以了。

拿着红铅笔的程序员太多了。有些是完完全全地拿着红铅笔的人，他们用手工做那些应由机器做的事。另一些是象征性地手拿红铅笔的人,他们使用机器的方法很笨，以至他们用手工也可以做好。本书的第一个目的是教你如何建立工具——一些帮助人们用机器而不是用红铅笔做事的程序，以及如何很好地使用这些工具。我们达到这一目的手段不是谈论它的一般性，而是写出实在的工具程序，根据我们的经验，这些程序都是很有用的。本书中的每个程序都精心地运行和测试过了，它们都是机器可阅读的形式。所有这些程序，都可不加修改地在多种机器上编译运行。

本书关心的第二个问题是如何写好的程序。随着讨论深入，我们希望给你展示出:好的设计令你写出既能工作又容易修改和维护的程序，运行工程学令你可以方便地使用它们;可靠性原则令你得到正确的答案;有效性令你把它们运行起来。

我们认为只读那些有关程序设计的陈词滥调是不可能学会程序设计的。仅

仅学习一些小的例子也不够。与其把结构程序设计和自顶向下设计等思想作为抽象的原理介绍给读者，还不如把每种思想的主要贡献抽出来放入我们的编程实践中。这样就可以看清楚它们的含义，在实际的问题中怎么使用它们，以及它们带来的好处。

我们还试图向读者展示出我们是怎样一步一步构造这些程序的，而不是仅仅给出最后的产品，或假装通过机械过程达到了最后结果。对于每个程序，我们都讨论了它的目的，怎么设计才能使它容易使用，影响它的结构和实现它的某些考虑，现有的其它选择。我们并不主张有什么最好的程序，我们也不主张我们使用的方法是设计和书写这些程序的唯一方法，但是即使你按不同的方法进行研究这些写的很好的、有用的一组连贯程序的开发，也有助于你更好地理解这些思想和某些重要意义，并且最后让你变成一个较好的程序员。

我们向你展示的工具并不太多。绝大多数都是些容易改变的程序，一个人可用一个小时，一天或一个礼拜就可以写出来。很清楚，我们不能介绍象操作系统或大型编译这类巨型程序;我们没有时间研究这些东西。反过来，我们把精力集中在你可能会用到的那些工具上，集中在有助于你最有效地使用操作系统和现有语言的那些程序上，其中有一个重要的问题:大小适当的精心选择和设计的程序，可用来设立与那些又大又差的程序之间的舒适而有效的接口。

只要可能我们就从较简单的构造更复杂的程序。只要可能，我们就尽量使用现有的(独立或组合的)工具，而避免构造新的。我们的程序一起工作，它们积累起来的效果远比你从一堆不易连接起来的类似程序中得到的效果大得多。本书的末尾为你介绍了一组工具，可以解决作为程序员会遇到的许多问题。

选择哪种工具呢?设计是一个广阔的天地，我们不能开始就复盖每种应用。因而我们把精力集中在那些堪称程序设计核心的机构上——有助于开发其它程序的程序。它们是我们真正使用的程序，绝大多数是天天用的，在我们写这本书时，几乎用到了所有这些版本。事实上我们之所以选用这些程序，还因为在我们工作的所有这些机器上证明这些程序有许多用法。虽然我们不敢说我们选择的这些程序能满足你的要求，但是有些你肯定会直接用上的。把你们系统中的有关程序与我们的设计作一对比，可以引导你去改进他们 。习惯上按工具思考问题，将会促使你书写那些只解决你的问题中的独特部分的程序，然后用它和完成其余部分的现有程序接口。

不管用作什么，较好的程序设计语言都是最重要的工具。没有它就很难书写和理解程序。用来对付语言的时间往往比用于编写程序的时间还长。有关程序设计的问题之一就是为这些程序选择一种语言，所有读者都熟悉，可以在所有的机器上运行，而又容易阅读的语言是不存在的，因此必须折衷。

因为 *Pascal* 已广为使用且又有良好的支持，所以在本书中我选用它作为基础。在当前的大学计算机科学课程中，*Pascal* 已是一门重要语言，差不多所有机器都有 *Pascal* ，而且足够标准了。

绝大部分程序员都具备阅读 *Pascal* 的知识。如果你用别的语言，学习本书也不会有什么困难，因为对于固定的结构程序，大部分语言谈起来都是一样

的。我们回避了 *Pascal* 中特异的东西,并把那些不可避免的藏在了良好定义的模块中。

虽然我们现在写的不是 *Pascal* 手册，但每遇到新的结构时，我们还要加以解释。第一章只讲了很少几个工具，主要用来引入 *Pascal* 编码格式和我们的约定。

大量的程序都有一个输入和一个输出，随着数据的通过，进行某些变换，我们称这种程序为过滤器。有些过滤器非常简单，你可以不把它们作为工具，但是选择过滤器，让它们协同工作可以处理非常复杂的问题，在第二章中收集了几个小过滤器，其中包括一个功能很强的字符翻译程序。

并非所有程序都是过滤器，第三章讨论的程序将以更复杂的方式与其周围环境相互作用，象文件的包含、比较及打印，以及一个管理一组文件的档案系统，严格地说，把程序从一个环境移到另一个环境的主要问题仍是程序如何与它的本地操作系统通讯的问题，我们通过指定一组访问这个环境的原语操作来处理。

程序的移植性问题,我们的所有程序都是根据这些原语写的，所以，与操作系统相关的那部分只限于少数几个过程和函数，使用这些原语的程序也可移到其它任何一台机器上去，只需在这台机器上能实现这些原语就可以了。我们已把本书的所有工具都未加改变地移到三种不同的机器上去了，充分证明了这种移植办法是可行的。

有些过滤器太长，必须另立几章，第四章中的排序程序，第五章的模式查找和替换，第八章的宏处理程序都属这一类，模式查找使用了第二章中字符翻译程序的绝大部分编码去识别字符类，字符类仍是可以指定的更大的模式集合之一。虽然这些过滤器都是面向子程序开发的，但是在任何应用中，过滤器这一概念都是很有价值的，它促使我们接受这样一个观点:在一个较大的过程中，一个程序就是一级，它们应该既简单又容易连接。还有另一个观点就是所有的文件和 *I/O* 设备都应是可互换的，所以任何一个程序都可以用任何文件和设备来工作。

第六章讲的是一个正文编辑，它比通常的分时系统中的那些编辑程序更富有启发性，把第五章中的大部分代码都并入这个正文编辑中了，所以它可识别的模式与第五章的一样，把正文编辑与某些介绍过的其它程序结合使用，可以完成那些需要你书写特殊程序的工作。即使你没在交互性环境下工作，也会发现正文编辑是很有用的。

第七章包含一个正文排版程序，本书就是用类似的软件编排的。

正如前面提到的，第八章是一个宏处理程序，这个程序虽说朴实无华，却很有用，你可用它扩充任何程序设计语言。

从上面的各章简介中,可以看出，我们着重于正文处理，程序员每天所作的大部分工作都是正文处理— 编辑源程序，准备输入数据，仔细查阅输出结果，编写文章说明。这些活动是程序设计的心脏，应尽量用机器实现它们。程序开发是工具能发挥其巨大效力的地方。因为正文处理程序大小不一，各种尺

寸的都有。所以，从中至少体现出象语言处理器和数值程序那样广泛的程序设计技术。

正如你将看到的，本书是按着应用而不是按照程序设计过程的不同侧面组织起来的。这不是一本有关算法，数据结构，或 *Pascal* 的参考书。没有按照设计编码、测试、调整、有效性、运行工程学、编写文件说明，或其它流行课题划分章节。我们从事于建立工具的工作。随着书写和讨论每个程序时，不同程度地，介绍了程序设计的所有侧面。在此过程中，我们试图向你传授建立工具的方法，所以你可以继续设计，构造和使用你自己的工具。

最后一个问题主要是程序如何与它的操作系统通讯。输入／输出是 *Pascal* 的最缺乏良好定义的部分，也是设计最差的一部分。我们通过定义 *get*、*put* 这类原语处理输入输出和相关问题，这些原语提供了清楚的，良好定义的一组操作系统函数。我们的全部程序都使用这些原语，它们必须在每个 *Pascal* 系统上实现，在第三章这一点会有更圆满的讨论，在附录中有些典型处理。

我们宁可使用标准 *Pascal* 而不采用限制较少的变种文本，是要保证可移植性。*Pascal* 有许多扩展，每个都克服了上述的某些问题，但是扩展彼此不同，这就破坏了达到可移植性的愿望。我们回避了扩展，因此我们的工具几乎可以在任何机器上运行。

目　　录

第一章　起步 (Getting Started)

在这一章里，我们将要简单地介绍一下我们是如何使用 Pascal 的以及贯穿全书的一些思想和约定，为使讨论的内容具体一些，还给出了几个小而精的程序。偶尔，我们不加解释就使用一些概念，因为不这么做，就无法介绍完整的程序，所以开头时，你必须相信某些事情，否则我们将陷入无休止的解释而无法自拔。

1.1 文件的复制 (File Copying)

我们要解释的第一个问题是一个程序怎样与它的周围环境进行通讯。因为我们的许多程序都与正文处理有关，所以从某一输入源读字符是一个基本问题。因此，我们设立了一个函数，称为 getc，它将读入下一字符，并用这个字符作为它的函数值。每调用 getc 一次，它都返回一个新字符。现在我们先不考虑这些字符是从哪里来的，虽然读者可能会想象出来，它们来自交互性终端或象磁盘这类辅助存贮设备。

我们暂不讨论 getc 可读的字符集，先解释一下由 getc 返回的两个特殊的值：一个是标志输入结束的文件结束符，一个是标志到达一行末尾的行结束符。它们与所有输入字符的代码都不一样。我们也不考虑有效性问题，虽然我们完全知道读一个字符至少要用几秒钟的时间。暂时我们要忽略掉我们能掩盖起来的一切细节。

接下来我们再设立一个 putc，它与 getc 正相反，putc 要把一个字符放到某处去，比如终端上，打字机或磁盘上。它的可接收自变量的值中有一个是通知正文行结束的。同样，我们既不关心精确的细节，也不考虑操作的有效性，我们的主导思想是 getc 和 putc 能一起工作——由 putc 把 getc 得来的字符放到另外一个地方去。

如果有人提供这两个基本操作，你就可以做许多有用的工作了，无需知道这个操作是怎样实现的。做为一个最简单的例子，你可以把 getc／putc 这一对函数放在一个循环中：

```
while ( getc(c) is not at end of input ) do
        putc(c)
```

这样，你就得到了一个程序，它把输入复制到输出上，然后退出。getc 和 putc 取消了某些细节(如果你想知道 getc 和 putc 怎么才能工作，不久我们为你展示一个标准 Pascal 的简单文本，并要说明我们为什么不用 read 和 write)。

象 getc 和 putc 这类函数，我们统称为原语(primitives) ——与"外部世界"接口的函数。通过它们再去调用具体的操作系统和编译程序中所用的输入

输出例程。对于那些使用 getc 和 putc 的程序来说，它们定义了字符的标准内部表示，并且提供了一种能在许多不同的计算机上交叉使用的输入输出机构。如果我们使用原语，我们就可以设计和书写与任何操作系统特性完全无关的程序。原语使得一个程序和它的操作系统环境隔离开来，并使得要执行的高级任务用很小的一组精心定义的基本操作清楚地表示出来。

上面展示的那个程序是用"伪码"(pseudo-code)书写的。"伪码"是一种表面上类似于实际程序设计语言的语言，其中夹杂着一些普通英语，以此避开某些细节。在我们尚未详细拟定出一个程序的全貌之前，可用伪码指出这个程序足够用的那个小部分。对于比较大的程序，往往是从伪码开始，然后逐步求精，直到它能执行为止。这种办法很有价值。你可以在一个较高的层次上修正和改进设计，既不用书写任何可执行码，却又保持接近于可执行的形式。

下一步是按 Pascal 书写的一个准备编译和运行的 copy 程序。

```
{ copy — copy input to output }
procedure copy;
var
    c :character;
begin
    while (getc(c) <> ENDFILE ) do
        putc(c)
end;
```

一些解释：首先，对于用过 Pascal 的人来说最清楚，这并不是一个完整的程序 —它只是一个过程。所以它能运行之前，还需要一些前后文把它包起来。我们打算按这种方法介绍我们的所有程序，把它们作为一些适于较大的前后文的过程，这样我们就可以把精力集中在本质的思想上。为使 copy 能够运行，我们实际上还需要下面这样一些前后文：

```
{ complete copy — to show one possible implementation }
program copyprog (input, output);
const
   ENDFILE = -1;
   NEWLINE = 10;{ ASCII value }
type
   character  = -1..127;{ ASCII,  plus ENDFILE }
{ getc — get one character from standard input }
function getc (var c :character) : character;
var
    ch : char;
begin
   if (eof) then
       c := ENDFILE
```

```
    else if (eoln) then begin
        readln;
        c := NEWLINE
    end
    else begin
        read(ch);
        c := ord(ch)
    end;
    getc := c
end;
{ putc -- put one character on standard output }
procedure putc (c : character);
begin
    if (c= NEWLINE) then
        writeln
    else
        write(chr(c))
end;
{ copy -- copy input to output }
procedure copy;
var
    c : character;
begin
    while (getc(c) <> ENDFILE) do
        putc(c)
end;
begin  { main program }
    copy
end.
```

这里展示出的前后文定义了 copy 所需的一切常量，类型和函数。为了用 Pascal 程序里所熟悉的术语解释 getc 和 putc 的性能并证明我们可按所有的 Pascal 系统支持的方式实现这些原语，我们用标准 Pascal 给出了上述的前后文。为使 getc 和 putc 尽可能有效，在实现时，总是给它们加上一些特殊的处理。

把一个程序包在一个外壳中的好处是：随着一些程序逐渐成熟，不断把它们逐渐地加到周围环境上，而不是每当给出一个新程序时都重复许多描述。本书中的程序的标准前后文要比这里展示的大得多。具体来说，我们把这些象 putc 和 getc 这类函数和过程的定义，象 ENDFILE 的常量，以及象 character 这种类型都放在外部的模块中，使它们对整个程序都有效。在第三

章和第八章中，我们将展示出一些程序，它们有助于自动地收集一个程序的一些片段。附录中展示了我们使用的说明部分。以后不加说明时，我们一律假定所有程序都是包含在这种外壳中的。

现在我们回到 copy 上来。第一行是注释，它简述出这个过程是干什么的。本书中的每个过程和函数都有这种说明，我们用大括号 { 和 } 作注释的分界符。如果你的字符集中不包含大括号，那就必须用 (* 和 *)。行

```
var
    c : character;
```

定义 c 是 character 型的变量。注意，character 和标准类型 char 不同，因为它必须使用 char 类型合法值之外的值来表示 ENDFILE 这类常量。行

```
while (getc(c) <> ENDFILE) do
    putc(c)
```

仍是 copy 所做的全部工作，其中的 while 语句指定了一个循环，只要括号中的条件为真，循环体(即本例中的 putc)将要反复执行。最后，条件变为假，循环结束。然后 copy 返回到调用者，整个程序结束。在这个 while 循环中测试条件是

```
getc(c) <> ENDFILE
```

记号 <> 表示"不等"，所以 getc 返回的字符不是 ENDFILE 时，这个循环将继续下去。

getc(c)把下一个字符放入它的自变量 c 中，同时又用它作为它的函数值。所以这个值既可用于测试，又可存起来以后使用，这些都由一个简单语句来完成。虽然这是一个不符合 Pascal 约定的用法，但它合法。我们经常这么用，觉得很顺手。

ENDFILE 是一个符号常量，它是 getc 和 getc 的用户约定好的值，用来表示输入已结束，即没有字符了(ENDFILE 不是存在外部介质上的字符)。上面选用的具体值并不神秘；甚至不同的机器上可以送回不同的值。唯一的限制是 ENDFILE 必须不同于 getc 可从文件上得到的任何字符。

在我们的程序中有足够的符号常量，它们为程序的可阅读性带来了极大的好处。只要你看一下，你就会知道测试

```
while (getc(c) <> ENDFILE ) do
```

的用意，这是因为 ENDFILE 比任何一个象 -1 之类的神秘的数意义更清楚。

在 Pascal 里，常量说明是随着程序被编译就用适当的值代替 ENDFILE 这类名字的一种很顺利的方法。

在第八章中，我们还要介绍一种程序，它让你能用更普通的方法定义和翻译符号常量，所以你可以在任意前后文中使用它们，并且在编译之前自动地将准确定义的字符串写入原码中。

为什么 copy 是有用的呢？当你在书写程序时，绝大部分操作系统都允许你指定某种外部文件或数据集合或物理 I/O 设备对应于你用的内部文件。

这种对应是在代码被编译完之后，它实际运行时建立的。那就意味着你手头上可以有一些程序，准备好运行，并且你可以在最后时刻确定使用哪种文件和设备。它也意味着你可以把这些程序当做黑箱来处理，忘记其内部处理也无所谓。如果你有原语 getc，它从标准输入(比如与 Pascal 程序有关的输入文件)读入，和原语 putc，它写到标准输出(比如文件输出)上。当你运行这个程序时，你就可以把它们连到适当文件或设备上去。第三章我们将展示一种方便的表示法，指出连接哪种文件。

顺便指出，"文件"这个概念，在不同的计算机系统上，有着不同的含义。现在，我们通俗的说，文件就是我们得到信息(经 getc)或放置信息(经 putc)的地方。它可能是组织成永久性文件系统的磁盘，或任意 I/O 设备。

我们的程序把来自任一种源的字符流复制到任一目标去。在象我们描述的这种环境下，你可以用它把在终端输入的字符放到一个文件上去，把一个文件送到打印机上印出来。

虽然还有别的方法也可以构造它，必要时，你还可以按许多方式去改进它，使它更好更快。copy 是一个基本工具，除了它本身的用途外，还可以用它构造其它更好的程序。如果你坚持把细节推后的原则，用那些从抽象的读入，写到抽象目标去的原语来编写程序，那么你的新工具将与先前的工具兼容。你将构起一个同时工作的工具包。

copy 提交使用之前，还有一个重要问题要解决，那就是供用户使用的说明文件，对于象 copy 这么简单的程序，这可能是傻事，因为只要看一眼代码，就会知道它能干啥。但这么简单的程序必竟是少数，用源码做参考并不总能办得到，所以我们给出一种称为"功能卡(manual page)"的说明文件。

程序

copy　将输入拷贝到输出

用法

copy

功能

copy 将输入原样拷贝到输出．它对于把文件从终端拷贝到文件，文件间的拷贝，甚至终端之间的拷贝都是很有用的．可以通过将文件拷贝到终端，来显示文件的内容．

例

回打(echo)键入的字符

```
copy
hello there, are you listening?
hello there, are you listening?
yes I am
yes I am
<ENDFILE>
```

这个功能卡是很有启发性的，但很简单，它给你提出了一些清楚的用法，并给出了一个具体的例子。用户使用前可用这个例子测试一下，(a) 确保 copy 已正确的装配好。(b) 增强对这个说明的理解。

顺便说一下，这个例子并不算琐碎。当你遇到一个新语言时，一个新的操作环境时，或者在一台计算机上做事的新方法时，第一个需要清除的障碍是学习怎样运行一个程序。大概，你必须掌握在这台计算机上怎么注册，怎么用正文编辑设立文件，怎么运行编译和链接程序，怎么修改文件，和启动你最后建立起来的程序!与此同时，我们最后还需要有一个复杂的过程，它暴露出它自己在上述这些易出问题的地方的困难。这样 copy 总是在新环境下建立起来的第一批程序中的一个。

练习 1-1　使 copy 程序在你的计算机上运行。

练习 1-2　你能否想出一个简单的程序，用它在系统上作为第一个站得住脚的程序。

1.2 字符计数 (Counting Characters)

大家多次遇到这样一个问题，想要知道一个文件含有多少字符，多少行，或多少字。如果你的文件注留在磁盘这种永久性存贮介质上的话，操作系统会为你做这些事。如果你文件存贮在磁带上，那么最容易的办法是让这个文件通过一个程序，由这个程序计数你想要的内容。

如果你不能立即想到为什么有人想要计数字符数，那也没关系。我们这是一本工具书，工具最好和其它东西一起工作，不久就会出现许多应用。

计数字符的基本操作如下:

```
{ charcount --- count characters  in standard input }
procedure charcount;
var
      nc : integer;
      c : character;
begin
      nc := 0;
      while (getc(c) <> ENDFILE) do
            nc := nc + 1;
      putdec(nc, 1);
      putc(NEWLINE)
end;
```

这个程序只比前面那个稍稍复杂一点点。计数变量 nc 用的是整型。nc 累计读入的字符数。为了打印出数字形式的字符数 charcount，调用 putdec，它把一个数转换成适合打印的字符串，并用 putc 输出这个字符串。按这种方