PEARSON

C和C++实务精选
品味岁月积淀，读享技术菁华

C语言接口与实现

创建可重用软件的技术

[美] | David R. Hanson | 著
郭旭 | 译

C Interfaces and Implementations:
Techniques for Creating Reusable Software

- C语言接口的参考手册
- 揭秘C语言编程技巧
- 剖析全部24个API和8个示例应用的源代码

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

PEARSON

[美] | David R. Hanson | 著
郭旭 | 译

C语言接口与实现

创建可重用软件的技术

C Interfaces and Implementations:
Techniques for Creating Reusable Software

人民邮电出版社
北京

图书在版编目（CIP）数据

C语言接口与实现 ：创建可重用软件的技术 /（美）汉森（Hanson, D. R.）著 ；郭旭译. — 北京 ：人民邮电出版社，2016.3
ISBN 978-7-115-40252-3

Ⅰ. ①C… Ⅱ. ①汉… ②郭… Ⅲ. ①C语言—程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2016)第039860号

版 权 声 明

◆ 著　　[美] David R. Hanson
译　　郭 旭
责任编辑　傅道坤
责任印制　张佳莹　焦志炜
◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市海波印务有限公司印刷
◆ 开本：800×1000　1/16
印张：23.75
字数：520 千字　　2016 年 3 月第 1 版
印数：1 - 3 000 册　　2016 年 3 月河北第 1 次印刷
著作权合同登记号　图字：01-2010-1455 号

定价：79.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

内 容 提 要

本书概念清晰、实例详尽，是一本有关设计、实现和有效使用C语言库函数，掌握创建可重用C语言软件模块技术的参考指南。书中提供了大量实例，重在阐述如何用一种与语言无关的方法将接口设计实现独立出来，从而用一种基于接口的设计途径创建可重用的API。

本书是所有C语言程序员不可多得的好书，也是所有希望掌握可重用软件模块技术的人员的理想参考书，适合各层次的面向对象软件开发人员、系统分析员阅读。

前　言

如今的程序员忙于应付大量关于 API（Application Programming Interface）的信息。但是，大多数程序员都会在其所写的几乎每一个应用程序中使用 API 并实现 API 的库，只有少数程序员会创建或发布新的能广泛应用的 API。事实上，程序员似乎更喜欢使用自己搞的东西，而不愿意查找能满足他们要求的程序库，这或许是因为写特定应用程序的代码要比设计可广泛使用的 API 容易。

不好意思，我也未能免俗：lcc（我和 Chris Fraser 为 ANSI/ISO C 编写的编译器）就是从头开始编写的 API。（在 *A Retargetable C Compiler: Design and Implementation* 一书中有关于 lcc 的介绍。）编译器是这样一类应用程序：可以使用标准接口，并且能够创建在其他地方也可以使用的接口。这类程序还有内存管理、字符串和符号表以及链表操作等。但是 lcc 仅使用了很少的标准 C 库函数的例程，并且它的代码几乎都无法直接应用到其他应用程序中。

本书提倡的是一种基于接口及其实现的设计方法，并且通过对 24 个接口及其实现的描述详细演示了该方法。这些接口涉及很多计算机领域的知识，包括数据结构、算法、字符串处理和并发程序。这些实现并不是简单的玩具，而是为在产品级代码中使用而设计的。实现的代码是可免费提供的。

C 编程语言基本不支持基于接口的设计方法，而 C++和 Modula-3 这样的面向对象的语言则鼓励将接口与实现分离。基于接口的设计跟具体的语言无关，但是它要求程序员对像 C 一样的语言有更强的驾驭能力和更高的警惕性，因为这类语言很容易破坏带有隐含实现信息的接口，反之亦然。

然而，一旦掌握了基于接口的设计方法，就能够在服务于众多应用程序的通用接口基础上建立应用程序，从而加快开发速度。在一些 C++环境中的基础类库就体现了这种效果。增加对现有软件（接口实现库）的重用，能够降低初始开发成本，同时还能降低维护成本，因为应用程序的更多部分都建立在通用接口的实现之上，而这些实现无不经过了良好的测试。

本书中的 24 个接口引自几本参考书，并且针对本书特别做了修正。一些数据结构（抽象数据类型）中的接口源于 lcc 代码和 20 世纪 70 年代末到 80 年代初所做的 Icon 编程语言的实现代码（参见 R. E. Griswold 和 M. T. Griswold 所著的 *The Icon Programming Language*）。其他的接口来自另外一些程序员的著作，我们将会在每一章的“扩展阅读”部分给出详细信息。

书中提供的一些接口是针对数据结构的，但本书不是介绍数据结构的，本书的侧重点在算法工程（包装数据结构以供应用程序使用），而不在数据结构算法本身。然而，接口设计的好坏总是取决于数据结构和算法是否合适，因此，本书可算是传统数据结构和算法教材（如 Robert Sedgewick 所著的 *Algorithms in C*）的有益补充。

大多数章节会只介绍一个接口及其实现，少数章节还会描述与其相关的接口。每一章的“接口”部分将会单独给出一个明确而详细的接口描述。对于兴趣仅在于接口的程序员来说，这些内容就相当于一本参考手册。少数章节还会包含“例子”部分，会说明在一个简单的应用程序中接口的用法。

每章的“实现”部分将会详细地介绍本章接口的实现代码。有些例子会给出一个接口的多种实现方法，以展示基于接口设计的优点。这些内容对于修改或扩展一个接口或是设计一个相关的接口将大有裨益。许多练习题会进一步探究一些其他可行的设计与实现的方法。如果仅是为了理解如何使用接口，可以不用阅读“实现”一节。

接口、示例和实现都以文学（literate）程序的方式给出，换句话说，源代码及其解释是按照最适合理解代码的顺序交织出现的。代码可以自动地从本书的文本文件中抽取，并按 C 语言所规定的顺序组合起来。其他也用文学程序讲解 C 语言的图书有 *A Retargetable C Compiler* 和 D. E. Knuth 写的 *The Stanford GraphBase: A Platform for Combinatorial Computing*。

本书架构

本书材料可分成下面的几大类：

基础	1. 引言
	2. 接口与实现
	4. 异常与断言
	5. 内存管理
	6. 再谈内存管理
数据结构	7. 链表
	8. 表
	9. 集合
	10. 动态数组
	11. 序列
	12. 环
	13. 位向量
字符串	3. 原子
	14. 格式化
	15. 低级字符串

	16. 高级字符串
算法	17. 扩展精度算术
	18. 任意精度算术
	19. 多精度算术
线程	20. 线程

建议大多数读者通读第 1 章至第 4 章的内容，因为这几章形成了本书其余部分的框架。对于第 5 章至第 20 章，虽然某些章会参考其前面的内容，但影响不大，读者可以按任何顺序阅读。

第 1 章介绍了文学程序设计和编程风格与效率。第 2 章提出并描述了基于接口的设计方法，定义了相关的术语，并演示了两个简单的接口及其实现。第 3 章描述了 `Atom` 接口的实现原型，这是本书中最简单的具有产品质量的接口。第 4 章介绍了在每一个接口中都会用到的异常与断言。第 5 章和第 6 章描述了几乎所有的实现都会用到的内存管理接口。其余各章都分别描述了一个接口及其实现。

教学使用建议

我们假设本书的读者已经在大学介绍性的编程课程中了解了 C 语言，并且都实际了解了类似《C 算法》一书中给出的基本数据结构。在普林斯顿，本书是大学二年级学生到研究生一年级的系统编程课程的教材。许多接口使用的都是高级 C 语言编程技巧，比如说不透明的指针和指向指针的指针等，因此这些接口都是学习这些内容非常好的实例，对于系统编程和数据结构课程非常有用。

这本书可以以多种方式在课堂上使用，最简单的就是用在面向项目的课程中。例如，在编译原理课程中，学生通常需要为一个玩具语言编写一个编译器。在图形学课程中同样也经常有一些实际的项目。本书中许多接口消除了新建项目所需要的一些令人厌烦的编程工作，从而简化了这类课程中的项目。这种用法可以帮助学生认识到在项目中重用代码可以节省大量劳动，并且引导学生在其项目中对自己所做的部分尝试使用基于接口的设计。后者在团队项目中特别有用，因为“现实世界”中的项目通常都是团队项目。

普林斯顿大学二年级系统编程课程的主要内容是接口与实现，其课外作业要求学生成为接口的用户、实现者和设计者。例如其中的一个作业是这样的，我给出了 8.1 节中描述的 `Table` 接口、它的实现的目标代码以及 8.2 节中描述的单词频率程序 `wf` 的说明，让学生只使用我们为 `Table` 设计的目标代码来实现 `wf`。在下一个作业中，`wf` 的目标代码就有了，他们必须实现 `Table`。有时我会颠倒这些作业的顺序，但是这两种顺序对大部分学生来说都是很新颖的。他们不习惯在大部分程序中只使用目标代码，并且这些作业通常都是他们第一次接触到在接口和程序说明中使用半正式表示法。

最初布置的作业也介绍了作为接口说明必要组成部分的可检查的运行时错误和断言。同样，只有做过几次这样的作业之后，学生们才开始理解这些概念的意义。我禁止了突发性崩溃，即不

是由断言错误的诊断所宣布的崩溃。运行崩溃的程序将被判为零分，这样做似乎过于苛刻，但是它能够引起学生们的注意，而且也能够让学生理解安全语言的好处，例如 ML 和 Modula-3，在这些语言中，不会出现突发性崩溃。（这种评分方法实际上没有那么苛刻，因为在分成多个部分的作业中，只有产生冲突的那部分作业才会判为错误，而且不同的作业权重也不同。我给过许多 0 分，但是从来没有因此导致任何一个学生的课程总成绩降低达 1 分。）

一旦学生们有了自己的几个接口后，接下来就让他们设计新的接口并沿用以前的设计选择。例如，Andrew Appel 最喜欢的一个作业是一个原始的测试程序。学生们以组为单位设计一个作业需要的任意算术精度的接口，作业的结果类似于第 17 章到第 19 章中描述的接口。不同的组设计的接口不同，完成后对这些接口进行比较，一个组对另一个组设计的接口进行评价，这样做很有启迪作用。Kai Li 的那个需要一个学期来完成的项目也达到了同样的学习实践效果，该项目使用 Tcl/Tk 系统（参见 J. K. Ousterhout 所著的 *Tcl and the Tk Toolkit*）以及学生们设计和实现的编辑程序专用的接口，构建了一个基于 X 的编辑程序。Tk 本身就是一个很好的基于接口的设计。

在高级课程中，我通常把作业打包成接口，学生可以自行修改和改进，甚至改变作业的目标。给学生设置一个起点可以减少他们完成作业所需的时间，允许他们做一些实质性的修改鼓励了有创造性的学生去探索新的解决办法。通常，那些不成功的方法比成功的方法更让学生记忆深刻。学生不可避免地会走错路，为此也付出了更多的开发时间。但只有当他们事后再回过头来看，才会了解所犯的错误，也才会知道设计一个好的接口虽然很困难，但是值得付出努力，而且到最后，他们几乎都会转到基于接口的设计上来。

如何得到代码

本书中的代码已经在以下平台上通过了测试。

处 理 器	操作系统	编 译 器
SPARC	SunOS 4.1	lcc 3.5 gcc 2.7.2
Alpha	OSF/1 3.2A	lcc 4.0 gcc 2.6.3 cc
MIPS R3000	IRIX 5.3	lcc 3.5 gcc 2.6.3 cc
MIPS R3000	Ultrix 4.3	lcc 3.5 gcc 2.5.7
Pentium	Windows 95 Windows NT 3.51	Microsoft Visual C/C++ 4.0

其中几个实现是针对特定机器的。这些实现假设机器使用的是二进制补码表示的整数和 IEEE 浮点算术，并且无符号的长整数可以用来保存对象指针。

本书中所有的源代码在 ftp.cs.princeton.edu 的目录 pub/packages/cii 下，匿名登录就可以下载。使用 ftp 客户端软件连接到 ftp.cs.princeton.edu，转到 pub/packages/cii 目录，下载 README 文件，文件中说明了目录的内容以及如何下载。

大多数最新的实现通常都是以 ciixy.tar.gz 或 ciixy.zip 的文件名存储的，其中 xy 是版本号，例如 10 是指版本 1.0。ciixy.tar.gz 是用 gzip 压缩的 UNIX tar 文件，而 ciixy.zip 是与 PKZIP 2.04g 版兼容的 ZIP 文件。ciixy.zip 中的文件都是 DOS/Windows 下的文本文件，每行均以回车和换行符结束。ciixy.zip 同时也可以在美国在线、CompuServe 以及其他在线服务器上下载。

登录 http://www.cs.princeton.edu/software/cii/同样也可以得到相应的信息。该页面还解释了如何报告勘误。

致谢

自 20 世纪 70 年代末以来，在我的科研项目以及在亚利桑那大学和普林斯顿大学的讲课中，我就已经使用过本书中的一些接口。选这些课程的学生最早试用了我设计的这些接口。这些年来他们的反馈凝结在本书代码与说明之中。我要特别感谢普林斯顿大学选修 COS 217 和 COS 596 课程的学生，正是他们在不知不觉中参与了本书中大多数接口的初步设计。

利用接口开发是 DEC 公司[①]的系统研究中心（System Research Center，SRC）的主要工作方式，1992 年和 1993 年暑假我在 SRC 从事 Modula-3 项目开发，亲身的工作经历消除了我对这种方法有效性的怀疑。我非常感谢 SRC 对我工作的支持，以及 Bill Kalsow、Eric Muller 和 Greg Nelson 与我进行的多次富有启迪的讨论。

我还要感谢 IDA 在普林斯顿的通信研究中心（Center for Communications Research，CCR）和 La Jolla，感谢他们在 1994 年暑假和 1995~1996 整个休假年对我的支持。还要感谢 CCR 为我提供了一个理想的地方，让我从容规划并完成了本书。

与同事和学生的技术交流也在许多方面完善了本书。一些即使看上去不相关的讨论也促使我对代码及其说明做了改进。感谢 Andrew Appel、Greg Astfalk、Jack Davidson、John Ellis、Mary Fernandez、Chris Fraser、Alex Gounares、Kai Li、Jacob Navia、Maylee Noah、Rob Pike、Bill Plauger、John Reppy、Anne Rogers 和 Richard Stevens。感谢 Rex Jaeschke、Brian Kernighan、Taj Khattra、Richard O'Keefe、Norman Ramsey 和 David Spuler，他们仔细阅读了本书的代码和内容，为本书的成功做出了不可磨灭的贡献。

① DEC 公司已被 Compaq 收购。——编者注

目　录

第1章 引 言

一个大程序由许多小的模块组成。这些模块提供了程序中使用的函数、过程和数据结构。理想情况下，这些模块中大部分都是现成的并且来自于库，只有那些特定于现有应用程序的模块需要从头开始编写。假定库代码已经全面测试过，而只有应用程序相关的代码会包含 bug，那么调试就可以仅限于这部分代码。

遗憾的是，这种理论上的理想情况实际上很少出现。大多数程序都是从头开始编写，它们只对最低层次的功能使用库，如 I/O 和内存管理。即使对于此类底层组件，程序员也经常编写特定于应用程序的代码。例如，将 C 库函数 `malloc` 和 `free` 替换为定制的内存管理函数的应用程序也是很常见的。

造成这种情况的原因无疑有诸多方面。其中之一就是，很少有哪个普遍可用的库包含了健壮、设计良好的模块。一些可用的库相对平庸，缺少标准。虽然 C 库自 1989 年已经标准化，但直至现在才出现在大多数平台上。

另一个原因是规模问题：一些库规模太大，从而导致对库本身功能的掌握变成了一项沉重的任务。哪怕这项工作的工作量似乎稍逊于编写应用程序所需的工作量，程序员可能都会重新实现库中他们所需的部分功能。最近出现颇多的用户界面库，通常会有这种问题。

库的设计和实现是困难的。在通用性、简单性和效率这 3 个约束之间，设计者必须如履薄冰，审慎前行。如果库中的例程和数据结构过于通用，那么库本身可能难以使用，或因效率较低而无法达到预定目标。如果库的例程和数据结构过于简单，又可能无法满足应用程序的需求。如果库太难于理解，程序员干脆就不会使用它们。C 库本身就提供了一些这样的例子，例如其中的 `realloc` 函数，其语义混乱到令人惊讶的地步。

库的实现者面临类似的障碍。即使设计做得很好，糟糕的实现同样会吓跑用户。如果某个实现太慢或太庞大，或只是感觉上如此，程序员都将自行设计替代品。最糟的是，如果实现有 bug，它将使上述的理想状况彻底破灭，从而使库也变得无用。

本书描述了一个库的设计和实现，它适应以 C 语言编写的各种应用程序的需求。该库导出了一组模块，这些模块提供了用于小规模程序设计（programming-in-the-small）的函数和数据结构。在几千行长的应用程序或应用程序组件中，这些模块适于用作零部件。

在后续各章中描述的大部分编程工具，都涵盖在大学本科数据结构和算法课程中。但在本书中，我们更关注将这些工具打包的方式，以及如何使之健壮无错。各个模块都以一个接口及其实现的方式给出。这种设计方法学在第2章中进行了解释，它将模块规格说明与其实现相分离，以提高规格说明的清晰度和精确性，而这有助于提供健壮的实现。

1.1　文学程序

本书并不是以“技巧”的形式来描述各个模块，而是通过例子描述。各章完整描述了一两个接口及其实现。这些描述以文学程序（literate program）的形式给出。接口及其实现的代码与对其进行解释的正文交织在一起。更重要的是，各章本身就是其描述的接口和实现的源代码。代码可以从本书的源文件文本中自动提取出来，所见即所得。

文学程序由英文正文和带标签的程序代码块组成。例如，

```
⟨compute x • y⟩≡
  sum = 0;
  for (i = 0; i < n; i++)
       sum += x[i]*y[i];
```

定义了名为⟨*compute* x • y⟩的代码块，其代码计算了数组x和y的点积。在另一个代码块中使用该代码块时，直接引用即可：

```
⟨function dotproduct⟩≡
  int dotProduct(int x[], int y[], int n) {
    int i, sum;

    ⟨compute x • y⟩
    return sum;
}
```

当⟨*function* dotproduct⟩代码块从本章对应的源文件中抽取出来时，将逐字复制其代码，用到代码块的地方都将替换为对应的代码。抽取⟨*function* dotproduct⟩的结果是一个只包含下述代码的文件：

```
int dotProduct(int x[], int y[], int n) {
    int i, sum;

    sum = 0;
    for (i = 0; i < n; i++)
         sum += x[i]*y[i];
    return sum;
}
```

文学程序可以按各个小片段的形式给出，并附以完备的文档。英文正文包含了传统的程序注释，这些并不受程序设计语言的注释规范的限制。

代码块的这种特性将文学程序从编程语言强加的顺序约束中解放出来。代码可以按最适于理解的顺序给出，而不是按语言所硬性规定的顺序（例如，程序实体必须在使用前定义）。

本书中使用的文学编程系统还有另外一些特性，它们有助于逐点对程序进行描述。为说明这些特性并提供一个完整的C语言文学程序的例子，本节其余部分将描述double程序，该程序检测输入中相邻的相同单词，如“the the”。

```
% double intro.txt inter.txt
intro.txt:10: the
inter.txt:110: interface
inter.txt:410: type
inter.txt:611: if
```

上述UNIX命令结果说明，“the”在intro.txt文件中出现了两次，第二次出现在第10行；而在inter.txt文件中，interface、type和if也分别在给出的行出现第二次。如果调用double时不指定参数，它将读取标准输入，并在输出时略去文件名。例如：

```
% cat intro.txt inter.txt | double
10: the
143: interface
343: type
544: if
```

在上述例子和其他例示中，由用户键入的命令显示为斜代码体，而输出则显示为通常的代码体。

我们先从定义根代码块来实现double，该代码块将使用对应于程序各个组件的其他代码块：

```
⟨double.c 3⟩≡
  ⟨includes 4⟩
  ⟨data 4⟩
  ⟨prototypes 4⟩
  ⟨functions 3⟩
```

按照惯例，根代码块的标签设置为程序的文件名，提取⟨*double.c* 3⟩代码块，即可提取整个程序。其他代码块的标签设置为double的各个顶层组件名。这些组件按C语言规定的顺序列出，但也可以按任意顺序给出。

⟨*double.c* 3⟩中的3是页码，表示该代码块的定义从书中哪一页开始。⟨*double.c* 3⟩中使用的代码块中的数字也是页码，表示该代码块的定义从书中哪一页开始。这些页码有助于读者浏览代码时定位。

main函数处理double的参数。它会打开各个文件，并调用doubleword扫描文件：

```
⟨functions 3⟩≡
  int main(int argc, char *argv[]) {
      int i;

      for (i = 1; i < argc; i++) {
          FILE *fp = fopen(argv[i], "r");
          if (fp == NULL) {
              fprintf(stderr, "%s: can't open '%s' (%s)\n",
                  argv[0], argv[i], strerror(errno));
              return EXIT_FAILURE;
```

```
        } else {
                doubleword(argv[i], fp);
                fclose(fp);
            }
        }
        if (argc == 1) doubleword(NULL, stdin);
        return EXIT_SUCCESS;
    }
```

```
⟨includes 4⟩≡
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
```

doubleword 函数需要从文件中读取单词。对于该程序来说，一个单词由一个或多个非空格字符组成，不区分大小写。getword 从打开的文件读取下一个单词，复制到 buf [0..size -1] 中，并返回 1；在到达文件末尾时该函数返回 0。

```
⟨functions 3⟩+≡
  int getword(FILE *fp, char *buf, int size) {
      int c;

      c = getc(fp);
      ⟨scan forward to a nonspace character or EOF 5⟩
      ⟨copy the word into buf[0..size-1] 5⟩
      if (c != EOF)
          ungetc(c, fp);
      return ⟨found a word? 5⟩;
  }
```

```
⟨prototypes 4⟩≡
  int getword(FILE *, char *, int);
```

该代码块说明了另一个文学编程特性：代码块标签 ⟨*functions* 3⟩ 后接的+≡表示将 getword 的代码附加到代码块 ⟨*functions* 3⟩ 的代码的后面，因此该代码块现在包含 main 和 getword 的代码。该特性允许分为多次定义一个代码块中的代码，每次定义一部分。对于一个“接续”代码块来说，其标签中的页码指向该代码块的第一次定义处，因此很容易找到代码块定义的开始处。

因为 getword 在 main 之后定义，在 main 中调用 getword 时就需要一个原型，这就是 ⟨*prototypes* 4⟩ 代码块的用处。该代码块在一定程度上是对 C 语言“先声明后使用”（declaration-before-use）规则的让步，但如果该代码定义得一致并在根代码块中出现在 ⟨*functions* 3⟩ 之前，那么函数可以按任何顺序给出。

getword 除了从输入获取下一个单词之外，每当遇到一个换行字符时都对 linenum 加 1。doubleword 输出时将使用 linenum。

```
⟨data 4⟩≡
  int linenum;
```

```
⟨scan forward to a nonspace character or EOF 5⟩≡
  for ( ; c != EOF && isspace(c); c = getc(fp))
      if (c == '\n')
          linenum++;

⟨includes 4⟩+≡
  #include <ctype.h>
```

linenum 的定义，也例证了代码块的顺序不必与 C 语言的要求相同。linenum 在其第一次使用时定义，而不是在文件的顶部或 getword 定义之前，后两种做法才是合乎 C 语言要求的。

size 的值限制了 getword 所能存储的单词的长度，getword 函数会丢弃过多的字符并将大写字母转换为小写：

```
⟨copy the word into buf[0..size-1] 5⟩≡
  {
      int i = 0;
      for ( ; c != EOF && !isspace(c); c = getc(fp))
          if (i < size - 1)
              buf[i++] = tolower(c);
      if (i < size)
          buf[i] = '\0';
  }
```

索引 i 与 size-1 进行比较，以保证单词末尾有空间存储一个空字符。在 size 为 0 时，if 语句保护了对缓存的赋值操作。在 double 中不会出现这种情况，但这种防性程序设计（defensive programming）有助于捕获“不可能发生的 bug”。

剩下的代码逻辑是，如果 buf 中保存了一个单词则返回 1，否则返回 0：

```
⟨found a word? 5⟩≡
  buf[0] != '\0'
```

该定义表明，代码块不必对应于 C 语言中的语句或任何其他语法单位，代码块只是文本而已。

doubleword 读取各个单词，并将其与前一个单词比较，发现重复时输出。它只查看以字母开头的单词：

```
⟨functions 3⟩+≡
  void doubleword(char *name, FILE *fp) {
      char prev[128], word[128];

      linenum = 1;
      prev[0] = '\0';
      while (getword(fp, word, sizeof(word)) {
          if (isalpha(word[0]) && strcmp(prev, word)==0)
              ⟨word is a duplicate 6⟩
          strcpy(prev, word);
      }
  }
⟨prototypes 4⟩+≡
```