普通高等教育“十三五”计算机类规划教材

计算机专业英语

◎ 霍宏涛　主编

Jisuanji
Zhuanye Yingyu

第2版

机械工业出版社
CHINA MACHINE PRESS

普通高等教育“十三五”计算机类规划教材

计算机专业英语

第2版

Professional English in Computer Field

主编　霍宏涛
参编　王　宇　邵　翀　王任革
主审　刘　舒

机械工业出版社

本书是为开设计算机专业英语课程的普通高校和广大有志于自学计算机英语的人员而编写的。全书共15章，分为两大部分：一部分是传统的计算机科学技术领域，包括计算机系统、操作系统、数据库、数据结构、计算机网络等内容；另一部分是新兴的领域，包括信息安全、信息安全技术、地理信息系统、图像处理和电子商务等内容。每章包括两篇正文和两篇阅读材料以及练习。每篇正文后附有单词注解和难句翻译。正文属于精读、精讲的内容。阅读材料主要侧重相应领域的最新发展动态，选文主要来源于近三年的外文期刊、技术报告和有关网站，目的是开拓学生的专业视野，激发学生的学习和阅读兴趣。本书内容新颖，选文来源宽泛，注重保持选文内容的原始性。

本书可以作为计算机类专业本科或者专科的专业英语教材，也可以作为信息类专业的选修教材，还可供参加计算机水平考试的考生和IT行业的工程技术人员学习和参考。

图书在版编目（CIP）数据

计算机专业英语/霍宏涛主编. —2版. —北京：机械工业出版社，2015.5

普通高等教育“十三五”计算机类规划教材

ISBN 978-7-111-49704-2

Ⅰ.①计… Ⅱ.①霍… Ⅲ.①电子计算机—英语—高等学校—教材 Ⅳ.①H31

中国版本图书馆CIP数据核字（2015）第054176号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）

策划编辑：刘丽敏 责任编辑：刘丽敏 责任校对：张 薇

封面设计：张 静 责任印制：刘 岚

北京中兴印刷有限公司印刷

2015年5月第2版第1次印刷

184mm×260mm · 25.75印张 · 640千字

标准书号：ISBN 978-7-111-49704-2

定价：53.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务 网络服务

服务咨询热线：010-88379833 机 工 官 网：www.cmpbook.com

读者购书热线：010-88379649 机 工 官 博：weibo.com/cmp1952

教育服务网：www.cmpedu.com

封面无防伪标均为盗版 金 书 网：www.golden-book.com

第2版前言

自从2007年本教材第1版发行后，计算机科学技术已经发生了很大的变化，以大数据、物联网、云计算等为代表的新技术和新应用层出不穷。计算机和信息网络技术已经在很大程度上影响着人们的生产生活，网络社会化趋势日趋明显。与此相对应，新名词和新术语不断涌现，教材内容的更新迫在眉睫。

在继承第1版教材基本架构、优势和特点的基础上，第2版力求进一步突出“与时俱进”的理念。每章4篇文章至少更新其中的1篇文章，涉及新兴信息网络技术的第12～15章则做到了每章更新2篇文章，1篇为正文，1篇为阅读材料。更新文章主要以近3年内的最新学术论文、技术报告、教材和网络信息为主，文章主题涵盖大数据、物联网、数字取证等新兴技术和应用。对于计算机基础、数据结构、操作系统、数据库、计算机网络等涉及计算机基础理论和技术的内容（第1～11章），考虑到主要技术架构和体系没有变化，主要是更新了阅读材料中的文章，部分章节更新了正文内容，将原来的正文改为阅读材料。此外，考虑到学生的英语水平在不断提升，本次更新的阅读材料的长度有所增加，进一步提升教材的信息容量，锻炼学生阅读长篇英文文献的能力。与上一版本相比，这一版本的内容更新约为40%。这一版本的另外一个变化是将此次更新的文章出处和作者标注在每篇文章后面，为学生进一步阅读原文和更多文献提供帮助，同时也是对作者知识成果的尊重。

本书第2版由霍宏涛拟订编写大纲，并负责全书的统稿和初审工作。王宇和霍宏涛共同负责第1～7章的编写工作，邵翀负责第8～11章的编写工作，霍宏涛和王任革共同负责第12～15章的编写工作。刘舒教授在百忙中再次审阅了本书的全部书稿，并提出了具体的意见和建议，在此表示衷心的感谢。

一些素不相识的教师和学生对本书的上一版本提出了一些很好的意见和建议，为本书的修订提供了很大的帮助。机械工业出版社的刘丽敏编辑对本教材的出版给予了全力的支持，鼓励我们要敢于坚持，直至本书成稿。由于编者水平有限，书中难免存在一些缺点和错误，恳请广大教师和学生批评指正。

编　者

第1版前言

计算机和信息技术的领跑者是以美国为首的英语系国家，所涉及的文献资料都以英语作为载体。因此，英语是IT领域的行业性语言，有着其他语言所无法替代的功能。很难想象一个不熟悉英语的人如何从事与计算机相关的工作和学习。而且，计算机科学技术的发展日新月异，这就要求计算机类专业的本科生必须具备熟练阅读有关的英文参考资料，如原版教材、科技期刊、联机文档和设备说明书的能力。因为计算机领域每天都有新的理论、技术和软件诞生，所以计算机专业英语的教材必须做到与时俱进、及时更新。学生通过一个学期的专业英语学习，应该不仅可以基本具备阅读各类英文文献的能力，而且可以及时了解计算机各领域的最新发展动态，并达到拓宽视野的目的。基于此，我们编写了本书。

全书共分15章，主要分为两大部分：一部分是传统的计算机科学技术领域，包括计算机系统、操作系统、数据库、数据结构、计算机网络等内容；另一部分是新兴的领域，包括信息安全与技术、地理信息系统、图像处理和电子商务等内容。每章包括两篇正文和两篇阅读材料以及练习。每篇正文后附有单词注解和难句翻译。单词以计算机的技术术语为主，句子则以句法复杂、具有专业英语句子特征的长句为主。两篇正文力求从经典文献（原版教材、科技期刊）中选取与本章主题有关的文章，长度为1500～2000个词，具有一定的难度。正文通过教师讲授，学生加以复习即可掌握，属于精读和精讲的内容。阅读材料主要侧重于相应领域的最新发展动态，选文主要来源于近三年的外文期刊和有关网站，目的是开拓学生的专业视野、补充学生的专业知识、激发学生的学习和阅读兴趣。

本教材主要具有以下特色。一是内容新。全书15章基本做到了每章至少有1篇选自近三年的国外期刊和网站的文章。学生在学习掌握经典技术术语、培养英文阅读能力的同时，可以及时掌握最新的技术发展动态，进而为主动追踪有关的技术发展提供语言基础。二是选文来源宽泛。有些专业英语选文侧重于正规外文教材，这样的文章文字严谨，语法、用词规范，但学生学习工作时遇到最多的不是英文教材，而是英文期刊、联机文档和网站信息，这是由本科应用型人才培养的目标所决定的。因此，编者在选文时，既包括教材、期刊，又包括联机文档和IT类的技术网站。文章风格迥异，用词、句法都有差别，能够更好地培养学生的阅读能力，增强其适应性。三是尽量保持选文的原汁原味。一些教材在选文时由于篇幅限制等原因，删减较大，破坏了文章的整体性，也不利于学生阅读能力的培养。我们在选文时尽量保持原文的内容，除了格式和插图的调整外，基本上对内容不作改动。学生可以直观感受到自己阅读一篇专业论文所需的时间和内容掌握程度。四是没有全文翻译。编者认为，专业英语课程的主要目的是培养学生的英语阅读能力，而非“翻译”能力。英语阅读能力的目标就是不需翻译而直接消化、理解文章的内容。随着阅读能力的提高，翻译过程必将消失。同时，我们在翻译难句时，也体会到句子的翻译在一定程度上是一个“信息损失”的过程。因此，本书没有将任何一篇文章进行全文翻译，难句翻译也是尽量压缩。目的就是要克服学生的翻译依赖心理，直接消化内容。

本书可以作为计算机类专业本科或者专科的专业英语教材，也可以作为信息类专业的选

修教材。讲授课时在36学时左右。本书对于计算机专业的学生熟悉、掌握国内外计算机领域的新理论、新技术、新工具具有重要意义，可以帮助学生熟练浏览国外有关的英文网站，阅读有关英文教材、期刊等文献资料。

本书由霍宏涛拟订编写大纲，并负责全书的统稿和初审工作。王宇负责第1～7章的编写工作，邵翀负责第8～11章的编写工作，霍宏涛和王任革共同负责第12～15章的编写工作。龚秋平、杨明两位教师也参与了本书的编写工作。刘舒教授在百忙中审阅了本书的全部书稿，并提出了具体的意见和建议，在此表示衷心的感谢。

编　者

Contents

Chapter One The Fundamental of Computers

1-1 Computer Types

A computer is a machine that can be programmed to **manipulate** symbols. Its principal characteristics are:

- It responds to a specific set of instructions in a **well-defined** manner.
- It can execute a prerecorded list of instructions (a program).
- It can quickly **store** and **retrieve** large amounts of data.

Therefore computers can perform complex and repetitive procedures quickly, precisely and reliably. Modern computers are electronic and digital. The actual **machinery** (wires, transistors, and circuits) is called hardware; the instructions and data are called software. All **general-purpose computers** require the following hardware components:

- Central processing unit (CPU): The heart of the computer; this is the component that actually executes instructions organized in programs ("software") which tell the computer what to do.
- Memory (fast, expensive, short-term memory): Enables a computer to store, at least temporarily, data, programs, and **intermediate** results.
- Mass storage device (slower, cheaper, long-term memory): Allows a computer to permanently retain large amounts of data and programs between jobs. Common mass storage devices include disk drives and tape drives.
- Input device: Usually a keyboard and mouse, the input device is the **conduit** through which data and instructions enter a computer.
- Output device: A display screen, printer, or other device that lets you see what the computer has accomplished.

In addition to these components, many others make it possible for the basic components to work together efficiently.[1] For example, every computer requires a bus that transmits data from one part of the computer to another.

Computer Sizes and Power

Computers can be generally classified by size and power as follows, though there is considerable **overlap**:

- Personal computer: A small, single-user computer based on a microprocessor.
- Workstation: A powerful, single-user computer. A workstation is like a personal computer, but it has a more powerful microprocessor and, in general, a higher-quality

monitor.

- Minicomputer: A multi-user computer capable of supporting up to hundreds of users simultaneously.
- Mainframe: A powerful multi-user computer capable of supporting many hundreds or thousands of users simultaneously.
- Supercomputer: An extremely fast computer that can perform hundreds of millions of instructions per second.

1.1 Supercomputer and Mainframe

Supercomputer is a broad term for one of the fastest computers currently available. Supercomputers are very expensive and are employed for specialized applications that require immense amounts of mathematical calculations. For example, weather forecasting requires a supercomputer. Other uses of supercomputers include scientific simulations, (animated) graphics, fluid dynamic calculations, nuclear energy research, electronic design, and analysis of geological data (e. g. in petrochemical prospecting). Perhaps the best known supercomputer manufacturer is Cray Research.

Mainframe was a term originally referring to the cabinet containing the central processor unit.[2] After the emergence of smaller "minicomputer" designs in the early 1970s, the traditional big iron machines were described as "mainframe computers" and eventually just as mainframes. Nowadays a mainframe is a very large and expensive computer capable of supporting hundreds, or even thousands, of users simultaneously. The chief difference between a supercomputer and a mainframe is that a supercomputer channels all its power into executing a few programs as fast as possible, whereas a mainframe uses its power to execute many programs concurrently. In some ways, mainframes are more powerful than supercomputers because they support more simultaneous programs. But supercomputers can execute a single program faster than a mainframe. The distinction between small mainframes and minicomputers is vague, depending really on how the manufacturer wants to market its machines.

1.2 Minicomputer

It is a midsize computer. In the past decade, the distinction between large minicomputers and small mainframes has blurred, however, as has the distinction between small minicomputers and workstations. But in general, a minicomputer is a multiprocessing system capable of supporting from up to 200 users simultaneously.

1.3 Workstation

It is a type of computer used for engineering applications (CAD/CAM), desktop publishing, software development, and other types of applications that require a **moderate** amount of computing power and relatively high quality graphics capabilities. Workstations generally come with a large, high-resolution graphics screen, at large amount of RAM, built-in network support, and a graphical

user interface. Most workstations also have a mass storage device such as a disk drive, but a special type of workstation, called a diskless workstation, comes without a disk drive. The most common operating systems for workstations are UNIX and Windows NT. Like personal computers, most workstations are single-user computers. However, workstations are typically linked together to form a local-area network, although they can also be used as stand-alone systems.

1.4 Personal Computer

It can be defined as a small, relatively inexpensive computer designed for an individual user. In price, personal computers range anywhere from a few hundred pounds to over five thousand pounds. All are based on the microprocessor technology that enables manufacturers to put an entire CPU on one chip. Businesses use personal computers for word processing, accounting, desktop publishing, and for running **spreadsheet** and database management applications. At home, the most popular use for personal computers is for playing games and recently for **surfing** the Internet. Personal computers first appeared in the late 1970s. One of the first and most popular personal computers was the Apple II, introduced in 1977 by Apple Computer. During the late 1970s and early 1980s, new models and competing operating systems seemed to appear daily. Then, in 1981, IBM entered the **fray** with its first personal computer, known as the IBM PC. The IBM PC quickly became the choice of personal computer, and most other personal computer manufacturers fell by the wayside. PC is short for personal computer or IBM PC. One of the few companies to survive IBM's **onslaught** was Apple Computer, which remains a major player in the personal computer marketplace. Other companies adjusted to IBM's dominance by building IBM **clones**, computers that were internally almost the same as the IBM PC, but that cost less. Because IBM clones used the same microprocessors as IBM PCs, they were capable of running the same software. Over the years, IBM has lost much of its influence in directing the evolution of PCs. Therefore after the release of the first PC by IBM the term PC increasingly came to mean IBM or IBM-compatible personal computers, to the exclusion of other types of personal computers, such as Macintoshes. In recent years, the term PC has become more and more difficult to **pin down**. In general, though, it applies to any personal computer based on an Intel microprocessor, or on an Intel-compatible microprocessor. For nearly every other component, including the operating system, there are several options, all of which fall under the rubric of PC. [3]

Today, the world of personal computers is basically divided between Apple Macintoshes and PCs. The principal characteristics of personal computers are that they are single-user systems and are based on microprocessors. However, although personal computers are designed as single-user systems, it is common to link them together to form a network. In terms of power, there is great variety. At the **high-end**, the distinction between personal computers and workstations has faded. High-end models of the Macintosh and PC offer the same computing power and graphics capability as **low-end** workstations by Sun Microsystems, Hewlett-Packard, and DEC.

Key Words

manipulate	*v.* 操作，控制
well-defined	*adj.* 明确定义的
store	*v.* 存储
retrieve	*v.* 获取，获得
machinery	*n.* 机械装置
general-purpose computer	通用计算机，一般用途的计算机
intermediate	*adj.* 中间的
conduit	*n.* 导线管
overlap	*n.* 重叠
mainframe	*n.* 大型计算机
moderate	*adj.* 中等的，适度的
spreadsheet	*n.* 电子表格程序
surf	*v.* 浏览（专指上网浏览，和 Internet 连用）
fray	*n.* 竞争
onslaught	*n.* 猛攻
clone	*n.* 克隆，仿制，仿造
pin down	确定，明确
high-end	*adj.* 高端的
low-end	*adj.* 低端的

Notes

1. In addition to these components, many others make it possible for the basic components to work together efficiently.

除了那些组成元素之外，还有一些其他的部分使得这些基本的组成元素能够一起有效地工作。

2. Mainframe was a term originally referring to the cabinet containing the central processor unit.

大型计算机这个词最早是指包含中央处理器单元的柜子。

3. For nearly every other component, including the operating system, there are several options, all of which fall under the rubric of PC.

包括操作系统在内的几乎每个组成元素都有多个符合 PC 规定的选项。

1-2 The Year 2038 Problem

1. What's So Special about 2038?

Early UNIX programmers had quite a sense of humor. In their documentation for the tunefs utility, a command-line program that fine-tuned the file system on the machine's hard disk, a note at the end reads "You can tune a file system, but you can't tune a fish." A later generation of UNIX authors, who were fearful that stuffy, humorless corporate drones would remove this cherished **pun**, added a programmer's comment inside the documentation's source code that read, "Take this out and a UNIX demon will dog your steps until the time_t's **wrap around**!"[1]

On January 19, 2038, that is precisely what's going to happen.

For the uninitiated, time_t is a data type used by C and C++ programs to represent dates and times internally. (You, Windows programmers out there might, also recognize it as the basis for the CTime and CTimeSpan classes in MFC.) time_t is actually just an integer, a whole number, that counts the number of seconds since January 1, 1970 at 12:00 AM Greenwich Mean Time. A time_t value of 0 would be 12:00:00 AM (exactly midnight) 1-Jan-1970, a time_t value of 1 would be 12:00:01 AM (one second after midnight) 1-Jan-1970, etc. Since one year lasts for a little over 31,000,000 seconds, the time_t representation of January 1, 1971 is about 31,000,000, the time_t representation for January 1, 1972 is about 62,000,000, et cetera.

By the year 2038, the time_t representation for the current time will be over 2,140,000,000. And that's the problem. A modern 32-bit computer stores a "signed integer" data type, such as time_t, in 32 bits. The first of these bits is used for the positive/negative sign of the integer, while the remaining 31 bits are used to store the number itself. The highest number these 31 data bits can store works out to exactly 2,147,483,647. A time_t value of this exact number, 2,147,483,647, represents January 19, 2038, at 7 seconds past 3:14 AM Greenwich Mean Time. So, at 3:14:07 AM GMT on that fateful day, every time_t used in a 32-bit C or C++ program will reach its upper limit.

One second later, on 19-January-2038 at 3:14:08 AM GMT, disaster strikes.

2. What Will the Time_t's Do When This Happens?

Signed integers stored in a computer don't behave exactly like an automobile's **odometer**. When a 5-digit odometer reaches 99,999 miles, and then the driver goes one extra mile, the digits all "turn over" to 00,000. But when a signed integer reaches its maximum value and then gets incremented, it wraps around to its lowest possible negative value. This means a 32-bit signed integer, such as a time_t, set to its maximum value of 2,147,483,647 and then incremented by 1, will become -2,147,483,648. Note that "-" sign at the beginning of this large number. A time_t value of -2,147,483,648 would represent December 13, 1901 at 8:45:52 PM GMT.

So, if all goes normally, 19-January-2038 will suddenly become 13-December-1901 in every

time _ t across the globe, and every date calculation based on this figure will go **haywire**. And it gets worse. Most of the support functions that use the time _ t data type cannot handle negative time _ t values at all. They simply fail and return an error code. Now, most "good" C and C ++ programmers know that they are supposed to write their programs in such a way that each function call is checked for an error return, so that the program will still behave nicely even when things don't go as planned. But all too often, the simple, basic, everyday functions they call will "almost never" return an error code, so an error condition simply isn't checked for. It would be too tedious to check everywhere; and besides, the extremely rare conditions that result in the function's failure would "hardly ever" happen in the real world.[2] (Programmers: when was the last time you checked the return value from printf () or malloc ()?) When one of the time _ t support functions fails, the failure might not even be detected by the program calling it, and more often than not this means the calling program will crash.

3. Will Fixing Year 2000 Bugs Help Fix Year 2038 Bugs?

No. time _ t is never, ever at fault in any Year 2000 bug.[3] Year 2000 bugs usually involve one of three things: The user interface, i. e. , what year do you assume if the user types in "00"; a database where only the last two digits are stored, i. e. , what year do you assume if the database entry contains a 00 for its year; and, in rare instances, the use of data items (such as the struct tm data structure's tm _ year member in a C or C ++ program) which store the number of years since 1900 and can result in displays like "19100" for the year 2000.

Year 2038 bugs, on the other hand, occur when a program reads in a date and carries it around from one part of itself to another.

You see, time _ t is a convenient way to handle dates and times inside a C or C ++ program. For example, suppose a program reads in two dates, date A and date B, and wants to know which date comes later. A program storing these dates as days, months, and years would first have to compare the years, then compare the months if the years were the same, then compare the days if the months were the same, for a total of 3 comparison operations. A program using time _ t would only have to compare the two time _ t values against each other, for a total of 1 comparison operation. Additionally, adding one day to a date is much easier with a time _ t than having to add 1 to the day, then see if that puts you past the end of the month, then increase the month and set the day back to 01 if so, then see if that puts you past the end of the year, et cetera. If dates are manipulated often, the advantage of using time _ t quickly becomes obvious. Only after the program is done manipulating its time _ t dates, and wants to display them to the user or store them in a database, will they have to be converted back into days, months, and years.

So, even if you were to fix every Year 2000 bug in a program in such a way that users and databases could use years as large as 9999, it wouldn't even brush on any of the Year 2038 bugs **lurking** within the same program.

4. The Problem with Pooh-Poohing

Admittedly, some of my colleagues don't feel that this **impending** disaster will strike too many people. They reason that, by the time 2038 rolls around, most programs will be running on 64-bit or even 128-bit computers. In a 64-bit program, a time _ t could represent any date and time in the future out to 292, 000, 000, 000 A. D. , which is about 20 times the currently estimated age of the universe.

The problem with this kind of optimism is the same root problem behind most of the Year 2000 concerns that **plagued** the software industry in previous years: **Legacy Code**. Developing a new piece of software is an expensive and time-consuming process. It's much easier to take an existing program that we know works, and code one or two new features into it, than it is to throw the earlier program out and write a new one from **scratch**. This process of enhancing and maintaining "legacy" source code can go on for years, or even decades. The MS-DOS layer still at the heart of Microsoft's Windows 98 and Windows ME was first written in 1981, and even it was a quick "port" (without many changes) of an earlier operating system called CP/M, which was written in the 1970s. Much of the financial software hit by the various Year 2000 bugs had also been used and maintained since the 1970s, when the year 2000 was still thought of as more of a science fiction movie title than an actual impending future. Surely, if this software had been written in the 1990s its Year 2000 Compliance would have been crucial to its authors, and it would have been designed with the year 2000 in mind. But it wasn't.

I should also mention that computer designers can no longer afford to make a "clean break" with the computer architectures of the past. No one wants to buy a new kind of PC if it doesn't run all their old PC's programs. So, just as the new generation of Microsoft Windows operating systems has to be able to run the old 16-bit programs written for Windows 3 or MS-DOS, so any new PC architecture will have to be able to run existing 32-bit programs in some kind of "**backward compatibility**" mode.

Even if every PC in the year 2038 has a 64-bit CPU, there will be alot of older 32-bit programs running on them. And the larger, more complex, and more important any program is, the better are its chances that it'll be one of these old 32-bit programs.[4]

5. What about Making Time _ t Unsigned in 32-bit Software?

One of the quick-fixes that has been suggested for existing 32-bit software is to re-define time _ t as anunsigned integer instead of a signed integer. An unsigned integer doesn't have to waste one of its bits to store the plus/minus sign for the number it represents. This doubles the range of numbers it can store. Whereas a signed 32-bit integer can only go up to 2, 147, 483, 647, an unsigned 32-bit integer can go all the way up to 4, 294, 967, 295. A time _ t of this magnitude could represent any date and time from 12:00:00 AM 1-Jan-1970 all the way out to 6:28:15 AM 7-Feb-2106, surely giving us more than enough years for 64-bit software to dominate the planet.

It sounds like a good idea at first. We already know that most of the standard time _ t handling

functions don't accept negative time _ t values anyway, so why not just make time _ t into a data type that only represents positive numbers?

Well, there's a problem. time _ t isn't just used to store absolute dates and times. It's also used, in many applications, to store differences between two date/time values, i. e. to answer the question of "how much time is there between date A and date B?". (MFC's CTimeSpan class is one notorious example.) In these cases, we do need time _ t to allow negative values. It is entirely possible that date B comes before date A. **Blindly** changing time _ t to an unsigned integer will, in these parts of a program, make the code unusable.

Changing time _ t to an unsigned integer would, in most programs, be robbing Peter to pay Paul. You'd fix one set of bugs (the Year 2038 Problem) only to introduce a whole new set (time differences not being computed properly).

6. Not Very Obvious, Is It?

The greatest danger with the Year 2038 Problem is its invisibility. The more-famous Year 2000 is a big, round number; it only takes a few seconds of thought, even for a **computer-illiterate** person, to imagine what might happen when 1999 turns into 2000. But January 19, 2038 is not nearly as obvious. Software companies will probably not think of trying out a Year 2038 **scenario** before **doomsday** strikes. Of course, there will be some warning ahead of time. Scheduling software, billing programs, personal reminder calendars, and other such pieces of code that set dates in the near future will fail as soon as one of their target dates exceeds 19-Jan-2038, assuming a time _ t is used to store them.

But the healthy paranoia that surrounded the search for Year 2000 bugs will be absent. Most software development departments are managed by people with little or no programming experience. (Dilbert's boss is an extreme case of this, but computer-illiterate software managers are more common than you might think.) It's the managers and their V. P. s that have to think up long-term plans and worst-case scenarios, and insist that their products be tested for them. Testing for dates beyond January 19, 2038 simply might not occur to them. And, perhaps worse, the parts of their software they had to fix for Year 2000 Compliance will be completely different from the parts of their programs that will fail on 19-Jan-2038, so fixing one problem will not fix the other.

Key Words

pun	*n.* 俏皮话
wrap around	回卷（指运算过界后，进位导致的符号位不合理的改变而引起的大数变小、小数变大的行为）
odometer	*n.* 里程表
haywire	*adj.* 混乱的
lurk	*v.* 潜伏
pooh-pooh	*v.* 轻视