IT English

Reading and Writing

高级IT英语读写教程

Advanced

1

总　主　编　司炳月
分册主编　刘晓静
分册副主编　郑　桥　马俊涛　郭　鹏　宋　辉
主　　　审　【美】Don Hong　【加】Marion Wyse

清華大學出版社

IT English

Reading and Writing

高级IT英语读写教程

Advanced

1

总 主 编 **司炳月**

分册主编 **刘晓静**

分册副主编 **郑 桥 马俊涛 郭 鹏 宋 辉**

清華大學出版社

北 京

内 容 简 介

为顺应经济全球化和信息技术的发展趋势，培养兼具 IT 专业技能和外语能力的人才，以适应 IT 行业发展需要，特编写了本教材。全书共有 8 个单元，内容涉及 IT 职业规划、编程语言、嵌入式设计、网站设计、信息安全、软件开发、IT 项目管理及 IT 行业前景八个方面。每个单元分为 Section A 和 Section B 两大部分，每部分包含一篇主课文和与主课文相关的生词和短语，并设计了大量形式多样、内容丰富的练习，同时还配有科技文写作和学术写作技能方面的指导，以全面提高学生的阅读、写作和翻译能力。本书部分课后习题的参考答案请读者访问 ftp://ftp.tup.tsinghua.edu.cn/ 下载使用。

本书适合作为 IT 及其相关专业本科高年级学生和科技英语专业学生的英语教材，也可作为从事 IT 相关工作人士自我提升的参考资料。

图书在版编目（CIP）数据

高级 IT 英语读写教程 . 1 / 司炳月总主编；刘晓静分册主编 . —北京：清华大学出版社，2017
ISBN 978-7-302-47965-9

Ⅰ. ①高… Ⅱ. ①司… ②刘… Ⅲ. ① IT 产业 – 英语 – 阅读教学 – 高等学校 – 教材 ② IT 产业 – 英语 – 写作 – 高等学校 – 教材 Ⅳ. ① F49

中国版本图书馆 CIP 数据核字（2017）第 211297 号

责任编辑：刘 艳 张维华
封面设计：平 原
责任校对：王凤芝
责任印制：宋 林

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京鑫丰华彩印有限公司
装 订 者：三河市溧源装订厂
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　张：**16　　**字　数：**346 千字
版　　次：2017 年 8 月第 1 版　　**印　次：**2017 年 8 月第 1 次印刷
印　　数：1~2000
定　　价：49.00 元

产品编号：073640-01

高级 IT 英语读写教程 1
Advanced IT English Reading and Writing 1

编　写　组

总 主 编　司炳月

分册主编　刘晓静

分册副主编　郑　桥　马俊涛　郭　鹏　宋　辉

编　　者　孙玉莹　王　茉　张贵玫　张晓博　曹　放　曹　麟
刘菁菁　刘　欣　邵　林　王晓华　于　芳　张婉婷
张雅欣　王珞珈

本教材是 2016 年辽宁省社会科学规划基金项目“专门用途英语理论在大学英语教学中的实践与应用——基于辽宁省 IT 英语人才培养模式的研究”（项目编号：L16DYY005）的阶段性成果；是辽宁省教育科学“十三五”规划 2017 年度立项课题：“IT 专门用途英语教材体系建设研究”（项目编号：JG17DB103）的阶段性成果；也是 2016 年国家社科基金项目：信息技术背景下中国外语学习环境“生态给养”转化有效性研究（项目编号：16BYY093）的阶段性成果。

前言

一、编写背景

1. 《国家中长期教育改革和发展规划纲要（2010—2020 年）》

信息时代的悄然而至，使得我国教育在面临难得的改革与发展机遇的同时，也面临着全新的挑战。传统的教育教学理念、教育模式、教学内容、教学方式、教学手段、教育结构乃至整个教育体制都将随之发生变革。2010 年，教育部颁发了《国家中长期教育改革和发展规划纲要（2010—2020 年）》（以下简称《纲要》），《纲要》中提出要“优化学科专业、类型、层次结构，促进多学科交叉和融合。扩大应用型、复合型、技能型人才培养规模”。在对创新人才培养模式的论述中提出，要“加强教材建设，确定不同教育阶段学生必须掌握的核心内容，形成教学内容更新机制”。

2. 《全民科学素质行动计划纲要实施方案（2016—2020 年）》

2016 年 3 月，国务院办公厅印发了《全民科学素质行动计划纲要实施方案（2016—2020 年）》（以下简称《方案》）。《方案》中对高等教育中的教材要求有清楚的阐述：“加强各类人群科技教育培训的教材建设。结合不同人群特点和需求，不断更新丰富科技教育培训的教材内容，注重培养具有创意、创新、创业能力的高层次创造性人才。将相关学科内容纳入各级各类科技教育培训教材和教学计划。”

3. 《大学英语教学指南》

《大学英语教学指南》（以下简称《指南》）是新时期普通高等学校制定大学英语教学大纲、进行大学英语课程建设、开展大学英语课程评价的依据。《指南》在对教材建设和教学资源的论述中明确阐述了：“鼓励各高校建设符合本校定位与特点的大学英语校本数字化课程资源；鼓励本区域内同类高校跨校开发大学英语数字化课程资源。”

二、编写原则

本套教材是与 IT 及其相关专业密切相关的知识课程，符合新形势下国家对复合型人才培养提出的要求，符合语言学习规律和新时代大学生的认知水

平，也满足大学生专业学习和未来职业发展的实际需要，有利于促进复合型人才培养目标的实现。本套教材在设计与编写过程中遵循以下原则：

1. 满足社会对于复合型人才培养的需求

当代大学生正面临多元化社会带来的冲突和挑战，复合型人才的培养成为国家、社会发展的需求。因此，为社会提供既具有专业知识又具备跨语言交际能力、能够直接参与国际交流与竞争的国际化通用型人才是高校人才培养的重点和难点，也是全球化对人才提出的更高、更新的要求。

2. 满足学生对于专业与外语知识相结合的需求

高校开设大学英语课程，一方面满足了国家、社会发展的需求，为国家改革开放和经济社会发展服务；另一方面，也满足了学生专业学习、国际交流、继续深造、工作就业等方面的需要。本套教材旨在满足 IT 及其相关专业学生的需求，帮助他们在掌握专业知识的同时提高英语水平。此外，教材亦体现了专门用途英语理论对大学英语教学课程设置的具体要求。

3. 满足大学英语教学大纲和教学目标的要求

大学英语的教学目标是培养学生的英语应用能力，增强学生的跨文化交际意识和交际能力；同时发展其自主学习能力，提高综合文化素养，使他们在学习、生活、社会交往和未来工作中能够有效地使用英语，满足国家、社会、学校和个人发展的需要。本套教材编写的目的就是使学生能够在 IT 专业领域中使用英语进行有效的交流；能够有效地运用有关篇章、语用等知识；能够较好地理解有一定语言难度、内容较为熟悉或与本人所学专业相关的口头或书面材料；能够对不同来源的信息进行综合、对比、分析，并得出自己的结论或形成自己的认识。

三、编写依据

1. “专业知识”+“外语能力”的“复合型”人才培养目标

大学英语课程作为高等学校人文教育的一部分，兼具工具性和人文性。在进一步提高学生英语听、说、读、写、译基本能力的基础上，学生可以通过学习与专业或未来工作有关的学术英语或职业英语获得在学术或职业领域进行交流的相关能力。本套教材是根据大学英语教学大纲和教学目标的要求，采用系统、科学的教材编写原则和方法编写而成。从教材的前期策划和准备、单元设计、教学资源开发、编写团队、内容设置和编排到教学效果的评价和评估都有整体的体系构建，以满足教学大纲和课程目标的要求。本套教材不但注重培养学生听、说、读、写、译这些语言基本技能，而且强化学生思辨、创新能力的培养。

2. “学生为主体”+“教师为主导”的“双主”教学理念

《指南》中提出大学英语教学应贯彻分类指导、因材施教的原则，以适应个性化教学的实际需要。新一轮的大学英语教学改革中也明确提出了“以教师为主导，以学生为主体”的“双主”教学理念。在教学过程中，教师的主导作用主要体现在课堂教学设计、教学组织、教学策略使用、教学管理和协调、课堂教学评价和评估等方面，而教师对课堂的主导方向要以满足学生的个性需求、促进学生的个性发展和自主学习为目的，只有两者相互结合，方能相得益彰，顺利实现大学英语教学改革目标。

3. “语言输入”+“语言输出”的“双向”驱动教学体系

本套教材在课堂教学活动和课后练习中设计了很多“语言输入”和“语言输出”的互动环节，教材采用任务式、合作式、项目式、探究式等教学方法，体现以教师为主导、以学生为主体的教学理念，使教学活动满足从“语言输入”到“语言输出”的需求。课后练习的设计关注学生自主学习能力的培养，引导和帮助他们掌握学习策略，学会学习；促使学生从“被动学习”向“主动学习”转变，真正让学生成为学习过程中的主体，实现课内和课外学习“不断线”。

4. “平面教材”+“立体化教材”的“双辅”交互优势

本套教材将大力推进最新信息技术与课程教学的融合，凸显现代学习方式的自主性、移动性、随时性等特点，发挥现代教育技术的推介作用。积极创建多元的教学与学习环境，利用互联网等信息基础设施和网络交流平台，使“平面教材”呈现出信息化教育的特征，形成“立体化教材”的特征。

此外，本套教材鼓励教师建设和使用微课、慕课，拓展教学内容，实施基于“教材平面内容”和“网上立体课件”的混合式教学模式，使学生朝着主动学习、自主学习和个性化学习方向发展，实现教学资源网络化、教学环境虚拟化、教学个性化、学习评估过程化等。

5. 以教材为引导、推动教师的自主专业发展，实现“教”“学”相长

《纲要》明确指出，要“建设高素质教师队伍。提升教师素养，努力造就一支师德高尚，业务精湛，结构合理，充满活力的高素质专业化教师队伍”。教师的专业发展能力受多种主客观因素的影响，需要外在环境和管理机制的保障。教师专业发展的规律性特点可归纳为长期性、动态性、实践性和环境依托性。本套教材的编写和使用正是根据实践性和环境依托性的特点，编写和使用新教材的过程也是教师更新教学理念、提高教学技能的专业发展必经过程。

四、教材结构

本套教材共包含“读写”和“听说”两大系列。其中，“读写”系列分为初级、中级、高级三个级别，共六个分册。“听说”系列分为初级和中级两个级别，共四个分册。

在“读写”系列中，每册书有 8 个单元。每个单元分为 Section A 和 Section B 两部分。Section A 根据大学英语教学大纲的要求编制，包含一篇精读课文，课文后有生词表、短语和表达、缩略词、术语和课后练习。Secton B 是按照专业英语学生的培养目标和要求编写，包含一篇与 Section A 同主题的阅读文章，旨在补充和强化专业阅读内容。两篇文章一易一难，每个单元都可以满足分级教学的需要和不同程度学生水平的需求，两个部分的练习形式多样，具有丰富性和系统性的特点。练习设计遵循语言学习的规律，针对不同层次、不同年级的学生，选材的难易程度、知识侧重点等方面均有所不同。

在“听说”系列中，每册书有 16 个单元，每个单元分为 Section A、Section B 及 Section C 三部分。其中，Section A 为听力技能训练，听力内容围绕 IT 相关主题展开。该部分由 Text A 和 Text B 两部分组成，前者针对 IT 及相关专业（非英语专业）学生，题目设计相对简单；后者针对英语专业（如科技英语）学生，题目设计难度有所增加。Section B 为口语技能训练，旨在培养学生的口头交际能力。Section C 为听力考试强化训练，该部分侧重应试，根据当下国内外几大英语考试（如大学英语四六级、托福、雅思等），全方位、多角度满足学生对英语学习的需求。希望通过题型多样、题量丰富的强化训练，让学生一方面熟悉并适应听力考试的多样题型，另一方面让学生检测自己的英语听力水平，提高自主学习能力。

五、教材特色

1. 素材原汁原味

本套教材的所有阅读和听力文本均选自英美国家真实的 IT 专业文本，包括 IT 相关专业的学术网站、期刊及英语原版教材。编者在选择文本时尽量选择新颖、有趣的分支话题，文章的语言也尽量避免过于严肃和刻板，使学生在理解和分析课文的过程中既能利用专业知识进行思考和判断，又不觉枯燥。

2. 内容注重实用性

本套教材的“读写”系列避免了国内同类教材培养目标单一、片面的缺陷，注重提高学生的多种技能。每个单元不仅包括阅读板块、翻译板块和写作板块，还针对 IT 及其相关专业的英语阅读、翻译、学术写作等技能进行系统的学习和训练。而在“听说”系列中，编者在选择听说文本的话题时，一方面迎合当今 IT 产业就业的发展趋势，另一方面也考虑与高校 IT 专业课程紧密相关，并参考国内各大重点高校 IT 专业设置，挑选出 IT 领域相关的热门话题，这些话题广泛涉及 IT 相关专业学生所关心的 IT 就业方面的问题、IT 专业知

识的学习方法、全国重点高校 IT 相关专业课程中开设的典型编程语言、当今的网络环境、时下 IT 领域多项前沿技术等内容，以便在提升学生英语语言能力的同时了解和学习与 IT 相关的专业知识，突出语言运用，通过文本传递 IT 知识，重现真实 IT 场景。

3. 练习内容和形式丰富多样

本套教材在阅读和听力理解、语言知识学习及技能训练方面都设计了大量的练习，而且练习形式富于变化，如简答、判断、填空、选择、配对、翻译、图表、口语交际等，学生不仅可以学习词汇、短语等语言点，还可以提高阅读和听力理解能力、分析语言的能力及表达能力。

六、适用对象

本套教材特别适合计算机科学与技术、信息管理与信息系统、软件工程和网络工程等与 IT 相关专业的学生学习和使用，可以分阶段或分学期选用；也特别适合从事软件系统需求分析、设计、开发、测试、运行及维护工作的工程师和管理人员查阅和参考。编者在选材上保证与 IT 信息技术密切相关的同时，努力确保文章内容贴近生活，所选材料涵盖了当前教育、工作和社会领域的诸多热点，文字形象生动、可读性强。因此，本套教材也比较适合那些有一定英语基础，同时也喜爱计算机应用技术和互联网文化的人士阅读，以扩展知识，开拓视野。

七、编写团队

本套教材由大连外国语大学软件学院教师担任主编团队。参与编写的编者有来自全国各高校的大学英语教师、专业英语教师、计算机专业的教师、IT 职场的企业专家以及旅居海外的专家和学者。

本套教材在编写过程中得到校企合作教材编写组的大力支持，在此表示衷心感谢。校企合作编写组成员包括李鸿飞、王文智、姜超、韩参、蒋振彬、梁浩、刘志强（排名不分先后）。

本套教材在编写过程中也得到了大连外国语大学软件学院的领导与英语教研室所有老师的鼎力支持，在此表示感谢。

由于编者水平有限，错误与缺点在所难免，恳请读者批评指正。

司炳月

2017 年 6 月

Contents

Unit 1 Finding My Niche

Have the courage to follow your heart and intuition. They somehow know what you truly want to become.

— *Steve Jobs*

Most people overestimate what they can do in one year and underestimate what they can do in ten years.

— *Bill Gates*

Preview

INFORMATION TECHNOLOGY continues to change the way we live, play, and do business. The dominance of the IT job market is due in part to numerous factors, including the prolific growth of the Internet and e-commerce, lower hardware and software prices that allow more businesses to upgrade their technology, increased demand for information security specialists spurred by the escalating frequency and sophistication of cyber-crimes, the advent of smarter applications that enable companies to analyze data and develop unprecedented business intelligence, and the dawn of the mobile computing era. However, despite rapid growth and increased opportunity, simply showing up will not guarantee success. The IT job market will continue to get more competitive as people go where the money and jobs are. This is why it's important to clearly identify your career objectives and develop a learning plan with the necessary skills, computer training and IT certifications to build a competitive edge and achieve your goals.

Section A

Pre-reading Activities

1. Check (√) the statements which you think are important for programmers.

☐ a) Math skills

☐ b) Good design

☐ c) Attention to details

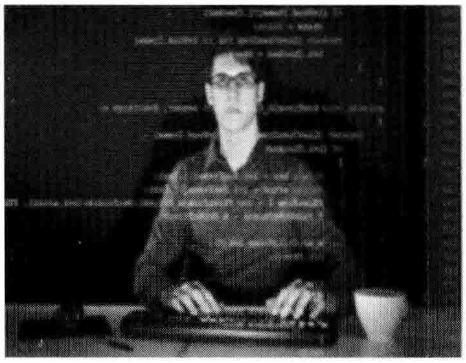
☐ d) Patience

☐ e) Self-learning skills

☐ f) Logical, precise, rigorous thinking

☐ g) Problem-solving skills

☐ h) Good communication skills



2. Work in pairs and discuss the following questions.

a) What job in IT field attracts you most? Why?

b) What should you do before you choose your career?

c) Why do IT majors need to learn math?

Text A

What Does It Take to Be a Programmer?

1. Many people want to know if they **have what it takes** to be a good programmer. There's no simple, check-these-boxes answer to the question, but there are some helpful **traits** that you may have or that you can develop.

2. Do I need to know Math? First, let me **eliminate** a **myth**: some people think that math skills are important, but I've seen great mathematicians who are **mediocre** programmers and lots of great programmers who are certainly not mathematicians (and probably never expected to be).

3. Programming is more of a designer's task—to be a good programmer, **having an eye for** style and good design is extremely helpful. I don't mean the type of style that **governs** where you put pieces of **syntax**. For instance, in C, there are several places where you can put **curly braces**[1] to surround blocks of code, and while there are heated debates about whether

```
if (...) {
}
```

is better than

```
if (...)
{
}
```

4. These are small points of little consequence, and as long as you are consistent, this trait will eventually come naturally. What I really mean by style is that you have to **have a good sense for discriminating** between "good" and "bad" approaches to attacking problems.

5. Design is important. When you first sit down to write a program, you probably don't know exactly what it should do (or how to do it). If you**'re disciplined about** it, you'll take some time to plan things out on paper and figure out more or less what you'd like your program to do. That's great, but it won't substitute for having actually used the program and noticed that, yes, it would be fantastic to add this one little feature here.

6. The secret is that adding little features can be very hard! This seems surprising to someone who's never programmed before: all you need to do is have it **print**[2] this one piece of data, or take this one type of input, etc. The problem is that inside the program, the **architecture** might not be designed to support that kind of information. For instance, let's say that you wanted to move a button from one place to another on a simple **graphical user interface**[3]. If the program has been well-designed, this shouldn't be too much of a problem, but if it hasn't, well, consider this possibility: the position of the button is governed by its location in **pixels**. All button locations are **hard-coded**[4] into the program. Now, if you move one button, you may have to go back and change where every single button is located both in the **routine** to draw the buttons and in the routine to accept the input. This can be quite a **hassle**!

7. Clearly, you want some way of having a notion of button positions that isn't quite so hard to change. But if you started out your program and didn't consider that it would be nice to be able

to move the buttons around, you have to go back and change possibly 20 or more lines of code (say, two for each button) just to move one of ten buttons. And if you make a mistake with one button, you're likely to see unforeseen results whose cause is hard to discover.

8. This kind of program design is **brittle**: it can work at first, but when you need to change something, it's not flexible. Each button depends on every other button and relies on the programmer to make the changes. A much better approach would be one in which the positions of buttons when they're drawn and when they're clicked on are linked—changing one wouldn't mean you have to change the other.

9. The more willing you are to put in the **up-front** thinking before designing your program, the easier you will find the actual writing of code. This is not to say that when you're first learning you shouldn't just write some simple programs without worrying too much about these issues. But you should be prepared to pay attention to these things and what problems your first programs did have.

10. The second trait that you need is patience. At some point in your programming career, you will certainly make small mistakes that cost you hours of **debugging** only to realize that you were misspelling a **variable** name so the **compiler** thought it was another variable. These things happen even to good programmers—and the better you get as you practice, the more you find that your bugs are interesting—but still hard to find. If you're not willing to patiently work through possible hypotheses and test each one in turn, you're probably going to find programming to be frustrating as much as it is **exhilarating**.

11. If you're looking to eventually have a programming job full-time, you'll want to acquire **exceptional** patience because you'll almost certainly be expected to spend a great deal of time working on **documenting** your code for other programmers and possibly even hunting bugs in someone else's code.

12. The benefit of all of this is that you gain an eye for small details that can have ripple effects and you become much better at the process of asking yourself what could go wrong and how you can test it. Finally, you have a lot of tools at your disposal to help **mitigate** the problems; you can use the compiler to find syntax errors and **debuggers** to find **runtime errors**[5]. Life is not **bleak**: not all of your time will be spent finding bugs!

13. Third, you need to be able to think in a logical, precise, **rigorous** way—you have to be willing and able to specify all of the details in a process and understand exactly what **goes into** it. This can lead to some amazing realizations—for instance, you will understand almost anything better once you've written a program to actually do it. One story goes that a group of programmers discovered a flaw in a state law passed by the **legislature** when trying to program the logic of the law—it turned out two paragraphs made contradictory statements! Nobody noticed until they tried to make it easy enough so that a computer could understand it. It means that you need to have the ability to eventually understand the **entirety** of a process at the level of detail required for a computer to be able to **mechanically** reproduce it.

14. At the same time, you must be capable of **framing** problems the right way and be or become a good problem solver. While your program may need to accomplish a certain task, don't **get caught up in** the first way you tried to solve the problem. For instance, if you need to store 20 phone numbers, it might make more sense to use an **array** than 20 separate variables. Even though you could eventually write the program that way, it would be much better to write it with the array. It would be a shorter program and an easier program to maintain. Often, restating the problem is a good way of **reframing** it. This is a skill you'll learn over time; you don't need to have mastered it before you start programming.

15. If you are **persistent**, willing to pay attention to issues of design and focus on both problem solving and precise solutions to problems, you will go as far as a programmer. If not, a programming career may turn out to be **exhausting** and tedious.

(Adapted from Alex Allain's "What Does It Take to Be a Programmer?"
on *Cprogramming.com*, Dec. 2011)

New Words

trait /treɪt/	*n.*	[C] element in sb's personality; distinguishing characteristic 人的个性；显著的特点；特征
eliminate /ɪ'lɪmɪneɪt/	*vt.*	**~ sb/sth (from sth)** to remove (esp. sb/sth that is not wanted or needed) 消除；清除；排除（尤指不必要或不需要的某人 / 某物）
myth /mɪθ/	*n.*	1. [C] a story from ancient times, especially one that was told to explain natural events or to describe the early history of a people 神话 2. [C] something that many people believe but that does not exist or is false 很多人相信却不存在或不真实的事或想法
mediocre /miːdɪ'əʊkə(r)/	*adj.*	not very good; second-rate; moderate; inferior in quality 不太好的；平庸的；第二流的
govern /'gʌvn/	*vt.*	1.(grammar) to require to be in a certain grammatical case, voice, or mood 支配；限定；需要 2. to influence (sth/sb) decisively; determine 支配某事物 / 某人；决定
	v.	to rule (a country, etc.); control or direct the public affairs of (a city, country, etc.) 统治（国家等）；控制，支配，治理，管辖（城市、国家等的公共事务）
syntax /'sɪntæks/	*n.*	1. [U] (linguistics) (rules for the) arrangement of words into phrases and phrases into sentences 句法；语句结构 2. [U] (computer science) the rules that describe how words and phrases are used in a computer language [计] 语法，一种程序设计语言的拼写和文法
discriminate /dɪ'skrɪmɪneɪt/	*vt.*	**~ between A and B**; **~ A from B** to see or make a difference (between two things) 分别，辨别，区分（两事物）

	vi.	**~ against sb/in favour of sb** to treat (one person or group) worse/better than others 歧视或偏袒（某人或某些人）
architecture /'ɑːkɪtektʃə(r)/	*n.*	1. [C] (computer science) the structure and organization of a computer's hardware or system software [计] 体系结构；架构 2. [U] the discipline dealing with the principles of design and construction and ornamentation of fine buildings 建筑学；设计建造结构的科学
graphical /'græfɪk(ə)l/	*adj.*	1. relating to or presented by a graph 图解的 2. written or drawn or engraved 绘画的；生动的
pixel /'pɪksəl/	*n.*	[C] the smallest discrete component of an image or picture on a CRT screen (usually a colored dot) (显示器或电视机图像的）像素
routine /ruː'tiːn/	*n.*	1. [C, U] an unvarying or habitual method or procedure（日常）程序；例行程式 2. [C] a computer program, or part of a program, that performs a specific function [计] 程序
hassle /'hæsl/	*n.*	1. [C, U] (informal) difficulty; trouble 麻烦；困难 2. [C] (informal) disorderly fighting; dispute 激战；争吵
	vt.	(informal) to annoy continually or chronically 使……烦恼；搅扰
brittle /'brɪtl/	*adj.*	1. hard but easily broken; fragile 硬而易碎的；脆弱的；(fig.) easily damaged; insecure 容易损坏的；不安全的 2. (of a sound) unpleasantly hard and sharp（指声音）尖利的 3. (of a person) lacking in warmth; hard（指人）冷淡的；难相处的
up-front /'ʌpfrʌnt/	*adj.*	advance; frank and honest 提前的；预先的；坦率的
debug /ˌdiː'bʌg/	*vt.*	1. (informal) to find and remove defects in (a computer program, machine, etc.) 检测并排除（计算机程序、机器等）中的故障 2. (informal) to find and remove hidden microphones from (a room, house, etc.) 从（房屋等）中找出并去除窃听器
variable /'veərɪəbl/	*adj.*	1. varying; changeable 变化的；可变的；易变的 2. (astronomy) (of a star) periodically varying in brightness（指星星）亮度周期变化的
	n.	1. [C] (often pl.) variable thing or quantity 可变的事物；可变的量 2. [C] (technical) a mathematical quantity which can represent several different amounts 变量
compiler /kəm'paɪlə(r)/	*n.*	1. [C] someone who collects different pieces of information to be used in a book, report, or list 编辑者 2. [C] (technical) a set of instructions in a computer that changes a computer language known to the computer user into the form needed by the computer 从高级语言原始码制造程序的程序；编译器
exhilarating /ɪg'zɪləreɪtɪŋ/	*adj.*	very exciting and enjoyable 使人高兴的；令人振奋的
exceptional /ɪk'sepʃənl/	*adj.*	very unusual; outstanding 异常的；罕见的；特殊的；杰出的；突出的
document /'dɒkjʊmənt/	*n.*	[C] paper, form, book, etc. giving information about sth; evidence or proof of sth 文件；公文；文献
	vt.	1. to prove or support (sth) with documents 用文件证实或证明（某事） 2. to record the details of an event, a process, etc. 记录；纪实性