

计算机专业系列教材
计算机软件
计算机网络
高等职业教育

计算机专业专业英语

Special English on Computer Application

主 编 蒋忠仁
副主编 刘胤杰 沈 刚 戴 酉
主 审 吴敏金 蒋忠理

重庆大学出版社

内容简介

本书所选取的内容基本覆盖了计算机软件、网络与计算机应用的各个领域,内容新,知识面广,趣味性强,具有广泛性和综合性。在每节的后面,给出难点词句的注释,附有相关专业的术语;每章都给出参考译文(英译中)、各种练习题及其答案,以帮助读者更好地掌握课文内容。

本书内容包括计算机组成、操作系统、网络与通信技术、数据库应用、移动应用、图像处理、Web 页面制作等。全书内容新颖,取材广泛,具有应用型特色,既可以作为应用型计算机及相关专业的英语教材使用,又可供计算机硬件、软件和信息处理等专业的工程技术人员使用。对于那些想了解计算机基础知识、掌握计算机应用技能的一般计算机用户,也是一本全新的、好用的英文读物。

图书在版编目(CIP)数据

计算机应用专业英语/蒋忠仁主编. —重庆:重庆大学出版社,2006.5

ISBN 7-5624-2996-0

I. 计... II. 蒋... III. 电子计算机—英语—高等学校—教材 IV. H31

中国版本图书馆 CIP 数据核字(2006)第 032773 号

计算机应用专业英语

主 编 蒋忠仁

副主编 刘胤杰 沈 刚 戴 酉

主 审 吴敏金 蒋忠理

责任编辑:王海琼 版式设计:王海琼

责任校对:任卓惠 责任印制:秦 梅

*

重庆大学出版社出版发行

出版人:张鸽盛

社址:重庆市沙坪坝正街174号重庆大学(A区)内

邮编:400030

电话:(023)65102378 65105781

传真:(023)65103686 65105565

网址:<http://www.cqup.com.cn>

邮箱:fxk@cqup.com.cn(市场营销部)

全国新华书店经销

重庆科情印务有限公司印刷

*

开本:787×1092 1/16 印张:11 字数:274千

2006年5月第1版 2006年5月第1次印刷

印数:1—3 000

ISBN 7-5624-2996-0

定价:16.00元

本书如有印刷、装订等质量问题,本社负责调换
版权所有,请勿擅自翻印和用本书
制作各类出版物及配套用书,违者必究

系列教材编写委员会

顾 问 邱玉辉

主 任 樊启宙 张学礼

副主任 杨滨生 任德齐 刘彩琴

委 员 (以姓氏笔画为序)

王 津 吴 焱 孙 辉

陈 晴 张洪星 张 英

黄顺强 袁开榜 龚小勇

序

高等职业教育具有“高等”和“职业”的双重特征,其目标是培养生产、建设、管理、服务第一线需要的高等技术应用型专门人才,是世界教育发展的共同趋势。近年来,我国高等教育的结构改革极大地促进了高等职业教育事业的发展,高等职业教育已成为我国高等教育的重要组成部分。

为了适应我国高等教育的改革,进一步满足高等职业教育计算机软件、计算机网络专业的教学及学科建设的需要,在全国各高等职业技术学院的支持下,重庆大学出版社采取学校、企业合作的形式,在全国十余所高等职业技术学院及企业(武汉职业技术学院、邢台职业技术学院、江苏信息职业技术学院、南昌工程学院、昆明冶金高等专科学校、重庆电子职业技术学院、重庆正大软件技术学院、重庆正大软件有限公司等)计算机相关专业的专家、学者中成立了编委会,并组建了一批具有丰富教学和实践经验的“双师型”作者队伍,力求编写出一套适合高等职业教育特点的高质量系列教材。

教学与生产相结合、理论和实践相结合、学校和社会相结合是高等职业教育的生命线;以技术应用能力和职业素质为主线来设计教学体系是高等职业教育教学改革的方向。依据高等职业教育的发展方向,本系列教材将强调理论知识的应用,注重基本能力、专业能力、综合能力及其技能的培养作为编写宗旨。

本系列教材将计算机与信息技术行业的标准及其技术岗位的需求作为组织编写的依据;在保证理论够用的基础上,根据产业结构、技术岗位体系以及职业岗位要求的要求组织理论和实训教材,并将职业教育的教学模式和方法融入其中。为了便于教学,今后将进一步建立学习资源网站,开发立体化教材。

本系列教材特点如下:

1. 以培养计算机网络、软件应用型人才为目标,遵循教育规律,系列教材的各分册相互衔接,并具有相关性和独立性。
2. 教材编写模块化。即将两个专业各自划分为若干个模块,它们既拥有共享的基础模块,又各自拥有一定选择余地的专业模块。

各门专业课程教材均可以一条逐步深化的主线将教学贯穿于学生学习的始终,形成“基础”、“提高”和“应用”3个层次的分阶段教学模式,学生在不断提高应用水平后可以直接承揽工程。

本系列教材的体系结构如下:

通用 模块	基础模块	计算机应用专业英语	* 计算机应用数学(上)	计算机应用电子技术	
		* 计算机网络技术基础	计算机应用数学(下)	* Java 程序设计基础	
		Delphi 程序设计基础	Visual Basic 程序设计基础	* Visual C++ 程序设计基础	
		* 计算机网络操作系统	计算机硬件技术基础	网页设计与网站建设	
	数据库模块	* 数据库技术基础与应用	数据库技术提高	数据库技术应用	
专业 模块	软件专业	软件工程模块	* 软件工程	软件测试技术	
		可视化编程模块	Java 程序设计提高	Visual Basic 程序设计提高	* Delphi 程序设计提高与应用
			Java 程序设计应用	Visual Basic 程序设计应用	Delphi 程序设计应用
			Visual C++ 程序设计提高	Visual C++ 程序设计应用	
	多媒体编程模块	* 多媒体程序设计(VB版)			
	网络编程模块	网络程序设计			
	网络专业	局域网模块	网络专业局域网技术基础	局域网技术应用	
		广域网模块	广域网技术应用		
		工程模块	* 网络安全与防火墙技术	网络系统集成与综合布线工程技术	

注:① * 课程为秋季推出的教材,其他课程将继续推出,实训教材正在筹划之中。

② 希望各院校和企业教师、专家参与本系列教材的建议,并请毛遂自荐担任后续教材的主编或参编,联系 E-mail:lich@cqup.com.cn。

3. 理论知识以够用为度,以实例、项目的工程实现为主线,将重点放在应用及操作技能上。

4. 力求创新。将新技术、新工艺纳入教材,尽可能体现文化性、社会性和艺术性,以利于提高学生的综合素质。

5. 思考题和习题具有启迪性和创新性。在编程、网络工程类教材的各章习题中大都包含与教材内容同步的中小型工程习题(或试验),全书最终将完成多个完整的工程实例。

本系列教材面向高等职业教育,适合于各类高等专科学校、高等职业学校、成人高等学校及高等院校主办的二级职业技术学院,并可作为从事计算机工作的工程技术人员的自学参考书。

该套教材的出版,重庆大学出版社的领导和编辑做了大量的工作,各教材的作者付出了艰辛的努力。但是,由于教材从策划到出版仅用了一年多的时间,承担教材编写任务的教师大多都担负着繁重的教学任务。在时间紧、任务重的情况下,教材中难免有不尽如人意之处,诚挚希望读者提出批评和建议,以便再版时改进。

编委会

2004年8月

前 言

我们正处在知识不断更新、计算机技术不断发展的信息社会。计算机的使用已经非常普及,而要用好计算机,计算机专业英语的熟练掌握是必不可少的基本功。编写本书的目的就是让读者在使用计算机时有所启发、接受训练、掌握操作技能、提高专业素质。

本教材的特点是内容新、知识面广、应用性强。其核心技术包括计算机科学与技术、计算机网络技术、电子信息科学技术和计算机综合应用技术。本书所选取的移动应用、XML 编程、Web 页面制作等内容,不仅给阅读带来了趣味性,而且有具体的操作指导,比较实用;在 SQL 应用和图像处理中,力求配备图表,使其对公式和模式的解读更加明朗。在突出新技术和应知、应会方面,遵循循序渐进、由浅入深的原则,配给每节的习题都由易到难、全面兼顾,读者可根据各自的需求,灵活选择。

为了实现上述教学目标,作者按上述核心技术将本书分为 12 章,除了必要的基础部分的章节如计算机组成、数据结构、语言及编程和数据库之外,大部分章节都是计算机应用部分:如图形和图像、多媒体、网络与通信、网页制作、计算机控制、信息管理和项目管理、人工智能、游戏、无线通信与移动应用、CAD、CAM 等。在每节的后面,给出难点词句的注释,附有相关专业的术语;另外,还给出每章的参考译文段、各种练习题及其答案,以帮助读者更好地理解课文、掌握计算机应用。

本书由上海应用技术学院蒋忠仁主编、负责策划和设计,刘胤杰、沈刚、戴西担任副主编,并由吕阿璐、蒋忠理共同参加编写。其中蒋忠仁编写第 4、11 章,刘胤杰编写第 7、8 章,沈刚编写第 1、5、10 章及附录,戴西编写第 2、9 章,吕阿璐编写第 3、6 章,蒋忠理编写第 12 章。在编写与出版过程中,得到了上海应用技术学院领导和重庆大学出版社领导的关心和支持,在此表示衷心感谢。同时全体编者

对引用到的国内外技术资料的作者表示衷心感谢。

华东师范大学吴敏金教授和上海建桥学院蒋忠理副教授担任主审,对书稿的修改提出了宝贵意见,编者在此表示衷心感谢。由于编写时间仓促,书中难免有疏漏之处,敬请广大读者批评指正。

编 者

2006 年 1 月

Contents

Chapter 1 Languages and Programming

1.1 Programming Concepts	1
1.2 File Access in C	3
1.3 Operator Overloading in C++	5
Exercises 1	11

Chapter 2 Computer Graphics

2.1 Building Basic Graphic Element — the Point and Line	12
2.2 Histogram Processing	15
2.3 Image Compressing and Coding	19
Exercises 2	22

Chapter 3 Multimedia

3.1 MPC	24
3.2 MIDI Interface	29
3.3 Video Compression	31
Exercises 3	33

Chapter 4 Network and Communication

4.1 What is the Internet	34
4.2 Access Networks	38
4.3 Electronic Mail in the Internet	41
Exercises 4	45

Chapter 5 Making Web Pages

5.1 Web Design Principles	46
5.2 XHTML and HTML	50
5.3 Cascading Style Sheets	53
Exercises 5	56

Chapter 6 Database

6.1 Brief Overview of Database	57
6.2 Brief Introduction to SQL	61
Exercises 6	67

Chapter 7 Computer Control

7.1 Information Theory	68
7.2 Automata Theory	71
7.3 Introduction to Modern Control Theory	75
Exercises 7	78

Chapter 8 Information Management

8.1 Electronic Commerce	80
8.2 Computer Security	85
Exercises 8	90

Chapter 9 Artificial Intelligence

9.1 What Is AI	91
9.2 Intelligent Agents	94
9.3 Robot Vision	96
Exercises 9	100

Chapter 10 Games

10.1 The Education of a Computer Game Designer	101
10.2 Artificial Intelligence in Game Design	105
10.3 Localization	109
Exercises 10	112

Chapter 11 Wireless and Mobile Network

11.1 How Does a Wireless Network Work	114
11.2 Types of Wireless Technology	117
11.3 Advanced Signaling Techniques Used to Mitigate Multi-path	121
Exercises 11	124

Chapter 12 CAD & CAM

12.1 Computer Aided Design 125
12.2 Engineering Analysis 127
12.3 Computer Aided Manufacturing 128
12.4 Computer Integrated Manufacturing 130
Exercises 12 132

APPENDIX

Appendix-1 Exercises Answer Key 134
Appendix-2 Glossary 157
Appendix-3 Bibliography 162

Chapter 1

Languages and Programming

1.1 Programming Concepts

Here are a few basic programming concepts to help you get started.

[1]Memory

A computer's memory is a very large set of bytes in which it stores the numbers (and letters and so on, but they're really all just bits) using at the moment. When you write a letter using a word processor, for example, the computer loads the word processing program file into memory, and also keeps the letter in memory as you write it. When you have finished the letter you can save it, and exit the word processor program, which is then discarded from memory along with your letter (but the files stay on disk where you can use them to load the program again later).

I say the memory is a collection of bytes because the bytes are arranged and numbered in order from zero to some very large number (if you have 128 Mbytes of memory, for example, the bytes are numbered from zero to 134 217 727). The number of each byte is called its address.^①

[2]Program

The whole point of programming is to create programs, so it's important to know what a program is. A program is a list of step-by-step instructions telling the computer how to do something.

Computers are very stupid, so they need explicit, detailed, step-by-step instructions in order to do something. Reading a file from a disk into memory, displaying a word on the screen, and so on are all accomplished by telling the computer exactly which signals need to be sent to the hardware (the disk drive, or the video controller) at what time. A collection of these instructions strung together to make the computer do something useful (or at least do something) is a program.^②

[3]File

A file is a collection of data stored together under one name on a disk (or some other permanent media).^③ The computer must read the data from a file into memory before it can

do anything with it, and in order to save data so that it will not be lost when the computer is turned off (or the program ends), the data will usually have to be written to a file.

[4] Variables

A variable in math is a symbol representing an arbitrary number. In programming, a variable is also a symbol (the variable name) that represents an arbitrary number (the value of the variable). However, that is pretty much where the similarity ends.^④ In math you write and manipulate equations to prove things that are true for all possible values of a variable, or to find the set of possible values for which the equation is true. In programming a variable has a particular value at any given time, and the reason it is called a variable is because that value can be changed as the program runs. A variable can be used to store a number input by a user, or the position of the mouse (periodically updated), or the result of a calculation, and so on.

Each variable in C or C++ has a type. The type of a variable determines its size in memory (the number of bytes it takes to store) and its representation, such as `int` for integer words, `char` for character bytes, and `float` for floating point numbers.

[5] Assignment and Arithmetic

Variables can be used to perform arithmetic. That is, you can add, subtract, multiply and divide using variables (as well as ordinary numbers, or constants). You have to assign the result of such arithmetic to a variable (or otherwise make use of it immediately). Thus the following statement:

$$x = y + 4$$

This assigns the result of adding four to the current value of the variable `y` to the variable `x`. Note that this is very different from what the same set of symbols means in math. To further illustrate, the following statement doesn't make much sense in math:

$$x = x + 1$$

There is no (ordinary) number which is equal to itself plus one, which is what the above would mean in math, but in programming it is perfectly ordinary and results in the variable `x` having a value one greater after the statement has been performed.

Words

binary ['baɪnəri] *adj.* 二元的, 二进制的

decimal ['desɪmə] *adj.* 十进制的

arithmetic [ə'riθmətik] *n.* 运算

differentiate [ˌdɪfə'renʃieɪt] *v.* 差异

representation [ˌreprɪzən'teɪʃən] *n.* 表示, 表述

symbol ['sɪmbəl] *n.* 符号

holdover ['həʊld,əʊvə(r)] *n.* 过时的东西

explicit [ɪks'plɪsɪt] *adj.* 明确的

permanent [ˌpɜ:mənənt] *adj.* 永久的

arbitrary [ˌɑ:bitrəri] *adj.* 多变的

periodically [ˌpiəri'ɒdɪkəli] *adv.* 周期性地

Notes

- ①numbered 在此意为编号,与表示数有所不同。该句可译为“内存中各个字节按次序排列并从 0 开始进行编号直到某个很大的数,例如 128 MB 内存将从 0 编号到 134 217 727。每个字节的编号称之为地址”。
- ②strung 为 string 的过去分词,注意与名词形式的 string 常专用于表示字符串不同。该句可译为:“程序是一组串联起来的指令,可使计算机完成一定的任务(至少做点什么)”。
- ③under 在此处意为通过、依据。该句可译为“文件是数据的集合,它通过文件名在磁盘或者其他存储媒质上加以存储”。
- ④However 引出转折 that is pretty much where the similarity ends. 该句可译为“数学中的变量是指一个表示可变数值的符号,在程序设计中,变量也是指一个符号——变量名,用于表示一个可变的数值,即变量的值。但是它们之间的相似性仅限于此”。

Terms

bit

A fundamental unit of information having just two possible values, as either of the binary digits 0 or 1.

1.2 File Access in C

The examples so far have all read the standard input and written the standard output, which are automatically defined for a program by the local operating system.

The next step is to write a program that accesses a file that is not already connected to the program. One program that illustrates the need for such operations is `cat`, which concatenates a set of named files onto the standard output.^① `Cat` is used for printing files on the screen, and as a general-purpose input collector for programs that do not have the capability of accessing files by name. For example, the command

```
cat x.c y.c
```

prints the contents of the files `x.c` and `y.c` (and nothing else) on the standard output.

The question is how to arrange for the named files to be read — that is, how to connect the external names that a user thinks of to the statements that read the data.^②

The rules are simple. Before it can be read or written, a file has to be opened by the library function `fopen`. `Fopen` takes an external name like `x.c` or `y.c`, does some housekeeping and negotiation with the operation system (details of which needn't concern us), and returns a pointer to be used in subsequent reads or writes of the file.

This pointer, called the file pointer, points to a structure that contains information about the file, such as the location of a buffer, the current character position in the buffer,

whether the file is being read or written, and whether errors or end of file have occurred. Users don't need to know the details, because the definitions obtained from `<stdio.h>` include a structure declaration call `FILE`. The only declaration needed for a file pointer is exemplified by

```
FILE * fp;
FILE * fopen(char * name, char * mode);
```

This says that `fp` is a pointer to a `FILE`, and `fopen` returns a pointer to a `FILE`. Note that `FILE` is a type name, like `int`, not a structure tag; it is defined with a typedef.

The call to `fopen` in a program is

```
fp = fopen(name, mode);
```

The first argument of `fopen` is a character string containing the name of the file. The second argument is the mode, also a character string, which indicates how one intends to use the file. Allowable mode includes read ("r"), write ("w"), and append ("a"). Some systems distinguish between text and binary files; for the latter, a "b" must be appended to the mode string.^③

If a file that does not exist is opened for writing or appending, it is created if possible. Opening an existing file for writing causes the old contents to be discarded, while opening for appending preserves them. Trying to read a file that does not exist is an error, and there may be other causes of error as well, like trying to read a file when you don't have permission. If there is any error, `fopen` will return to `NULL`.

The next thing needed is a way to read or write the file once it is open. There are several possibilities, of which `getc` and `putc` are the simplest. `getc` returns the next character from a file; it needs the file pointer to tell it which file.

```
int getc(FILE * fp)
```

`getc` returns the next character from the stream referred to by `fp`; it returns to EOF for end of file or error.

`putc` is an output function:

```
int putc(int c, FILE * fp)
```

`putc` writes the character `c` to the file `fp` and returns to the character written, or EOF if an error occurs. Like `getchar` and `putchar`, `getc` and `putc` may be macros instead of functions.

When a C program is started, the operation system environment is responsible for opening three files and providing file pointers for them. These files are the standard input, the standard output, and the standard error; the corresponding file pointers are called `stdin`, `stdout`, and `stderr`, and are declared in the `<stdio.h>`. Normally `stdin` is connected to the keyboard and `stdout` and `stderr` are connected to the screen, but `stdin` and `stdout` may be redirected to files or pipes.^④

`Getchar` and `putchar` can be defined in terms of `getc`, `putc`, `stdin`, and `stdout` as follows:

```
#define getchar() getc(stdin)
```

```
# define putchar(c) putc((c), stdout)
```

For formatted input or output of files, the functions `fscanf` and `fprintf` may be used. These are identical to `scanf` and `printf`, except that the first argument is a file pointer that specifies the file to be read or written; the format string is the second argument.

```
int fscanf(FILE * fp, char * format, ...)
```

```
int fprintf(FILE * fp, char * format, ...)
```

Words

`concatenate` [kən'kæteɪneɪt] *v.* 连接

`external` [eks'tɜːnl] *adj.* 外部的

`housekeeping` ['haʊski:pɪŋ] *n.* 内部管理

`negotiation` [ni:ɡəʊfi'eɪʃən] *n.* 协调

`subsequent` ['sʌbsɪkwənt] *adj.* 随后的

`definition` [ˌdefɪ'nɪʃən] *n.* 定义

`declaration` [ˌdeklə'reɪʃən] *n.* 声明

`exemplify` [ɪɡ'zemplɪfaɪ] *vt.* 例证

`argument` ['ɑ:gjʊmənt] *n.* 参数, 变量

`discard` [dɪs'kɑ:d] *vt. & v.* 丢弃

Notes

- ① `cat`, 是一条常用的 UNIX 命令。该句可译为“接下来要做的是写一个程序可以对尚未与程序连接的文件进行处理, 一个很好的例子是 `cat` 命令, 它用于将一系列指定的文件连接起来并输出到标准输出上”。
- ② `that is`, 引出同位语。该句可译为“问题在于如何使得命名文件能够被读取? 也就是说, 如何在用户所希望的外部文件与读取数据的语句之间建立联系”。
- ③ `text files`: 文本文件, `binary files`: 二进制文件。该句可译为“可选的模式有读(‘r’)、写(‘w’)、追加(‘a’)。某些系统对文本文件和二进制文件加以区别, 对于后者, 可以在模式字符串的后面加上‘b’”。
- ④ `stdin`、`stdout`、`stderr` 是 C 语言中 3 个预先打开的标准文件名。`pipe`(管道)是操作系统中一个与程序输入输出相关的概念。该句可译为“通常 `stdin` 与键盘相连接, 而 `stdout` 和 `stderr` 则连接到屏幕。但 `stdin` 和 `stdout` 都可以被重定向到文件或者管道”。

Terms

`typedef`

`typedef` is a C keyword allows you to create a new type from an existing type.

`pipe`

A pipe is an operating system mechanism originating in UNIX, which allows the user to direct the output of one process as the input of another process.

1.3 Operator Overloading in C++

Operator overloading allows the programmer to define versions of the predefined

operators for operands of class type. For example, the String class defines many overloaded operators. Here is the definition:

```
#include <iostream>
class String;
istream & operator>>( istream &, String & );
ostream & operator<<( ostream &, const String & );
class String {
public:
    // overloaded set of constructors
    // provide automatic initialization
    String( const char * = 0 );
    String( const String & );
    // destructor: automatic deinitialization
    String();
    // overloaded set of assignment operators
    String & operator=( const String & );
    String & operator=( const char * );
    // overloaded subscript operator
    char& operator[]( int ) const;
    // overloaded set of equality operators
    // str1 == str2;
    bool operator==( const char * ) const;
    bool operator==( const String & ) const;
    // member access functions
    int size() { return _size; }
    char * c_str() { return _string; }
private:
    int _size;
    char * _string;
};
```

The class String has three sets of overloaded operators. The first set defines the assignment operators for class String:

```
// overloaded set of assignment operators
String & operator=( const String & );
String & operator=( const char * );
```

The first assignment operator is the copy assignment operator, which supports the assignment of one object of type String to another. The second assignment operator supports the assignment of a C-style character string to an object of type String, as follows: