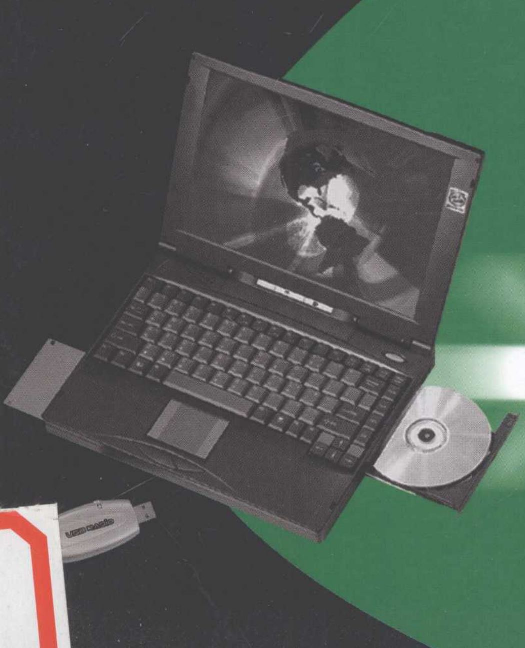
计算机应用专业英语

■主　编　蒋忠仁　副主编　刘胤杰　沈　刚　戴　酉

■主　审　吴敏金　蒋忠理

Jisuanji Yingyong Zhuanye Yingyu



重庆大学出版社

1211335

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

Jisuanji Yingyong Zhuanye Yingyu

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

请按此裁下寄回我社或在网上下载此表格填好后E-mail发回

教师信息反馈表

为了更好地为教师服务，提高教学质量，我社将为您的教学提供电子和网络支持。请您填好以下表格并经系主任签字盖章后寄回，我社将免费向您提供相关的电子教案、网络交流平台或网络化课程资源。

书名：		版次	
书号：			
所需要的教学资料：			
您的姓名：			
您所在的校(院)、系：	校(院)　　　系		
您所讲授的课程名称：			
学生人数：	____人　____年级	学时：	
您的联系地址：			
邮政编码：		联系电话	（家） （手机）
E-mail：(必填)			
您对本书的建议：		系主任签字 盖章	

请寄：重庆市沙坪坝正街174号重庆大学(A区)重庆大学出版社市场部

邮编：400030

电话：023-65111124

传真：023-65103686

网址：http://www.cqup.com.cn

E-mail：fxk@cqup.com.cn

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

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 ['bainəri] *adj.* 二元的，二进制的	holdover ['həuld,əuvə(r)] *n.* 过时的东西
decimal ['desiməl] *adj.* 十进制的	explicit [iks'plisit] *adj.* 明确的
arithmetic [ə'riθmətik] *n.* 运算	permanent ['pə:mənənt] *adj.* 永久的
differentiate [ˌdifə'renʃieit] *v.* 差异	arbitrary ['ɑ:bitrəri] *adj.* 多变的
representation [ˌreprizen'teiʃən] *n.* 表示，表述	periodically [ˌpiəri'ɔdikəli] *adv.* 周期性地
symbol ['simbəl] *n.* 符号	

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 [kɔnˈkætineit] *v.* 连接	definition [ˌdefiˈniʃən] *n.* 定义
external [eksˈtəːnl] *adj.* 外部的	declaration [ˌdekləˈreiʃən] *n.* 声明
housekeeping [ˈhauskiːpiŋ] *n.* 内部管理	exemplify [igˈzemplifai] *vt.* 例证
negotiation [niˌgəuʃiˈeiʃən] *n.* 协调	argument [ˈɑːgjumənt] *n.* 参数，变量
subsequent [ˈsʌbsikwənt] *adj.* 随后的	discard [disˈkɑːd] *vt.* & *v.* 丢弃

Notes

①cat，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"。

②that 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"。

③text 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'）。某些系统中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'"。

④stdin、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:

```
String name;
name = "Sherlock"; // use of operator=( char * )
```

The second set of overloaded operators defines one operator - the subscript operator:

```
// overloaded subscript operator
char& operator[]( int ) const;
```

This operator allows programs to index into objects of class String the same way we index into an object of built-in array type[①]:

```
if ( name[0] ! = 'S' )
    cout << "oops, something went wrong\n";
```

The third set of overloaded operators defines equality operators for objects of class String. A program can compare two objects of class String for equality, or can compare an object of class String with a C-style character string for equality.

```
// overloaded set of equality operators
// strl == str2;
bool operator==( const String & ) const;
bool operator==( const char * ) const;
```

Overloaded operators allow objects of class type to be used with the operators, allowing the manipulation of objects of class type to be as intuitive as that of objects of built-in types. For example, if we want to define an operation to support the concatenation of two objects of type String, we could decide to implement this new operation as a member function named concat(). But why choose the name concat() and not append(), for example? [②] Although the name chosen is both logical and mnemonic, users may forget the exact name we chose. It is often easier to remember the name of an operation if we define it as an overloaded operator. Instead of concat(), for example, we prefer to name the new String operation operator+=().

An overloaded operator is declared in the class body in the same way as an ordinary member function, except that its name consists of the keyword operator followed by one of a large subset of the predefined C++ operators. Operator+=() might be declared as follows in class String,

```
class String {
public:
    // overloaded set of += operators
    String & operator +=( const String & );
    String & operator +=( const char * );
    // ...
private:
    // ...
};
```

Class Member versus Nonmember

Let's look at our String class equality operators in a little bit more detail. The first operator allows us to compare for equality two objects of class String, and the second operator allows us to compare an object of class String with a C-style character string.③ For example:

```
#include "String. h"
int main() {
    String flower;
    // set flower to something
    if ( flower == "lily" ) // ok
        // ...
    else
    if ( "tulip" == flower ) // error
        // ...
}
```

The first use of the equality operator in main( ) calls the String class overloaded operator==(const char *). However, the second use of the equality operator results in a compiler error. How can this be?

The problem is that an overloaded operator that is a class member is only considered when the operator is used with a left operand that is an object of class type. Because the left operand is not of class type, the compiler tries to find a built-in operator that can take a left operand that is a C-style character string and a right operand that is of class String. Of course, no such operator exists and the compiler issues an error message for the second use of the equality operator in main().

But, you will say, it is possible to create an object of class String from a C-style character string using the class constructor. Why doesn't the compiler implicitly do the following conversion:

```
if ( String( "tulip" ) == flower ) // ok: calls member operator
```

The short answer is efficiency. Overloaded operators do not require that both operands be of the same type.

So, to find the equality operator for this comparison, the compiler would have to look at all the class definitions, to find all the constructors that can convert the left operand to a class type, and then find the associated overloaded equality operators for each of these class types to see if any can perform the equality operation. The compiler would then need to decide which combination of constructor and equality operator, if any, best matches the right-hand operand! If the compiler were required to do this, the time it would take to compile C++ programs would increase significantly. Instead, the compiler only considers