C/C++头文件一览
C、传统 C++
**#include <assert.h>　　//设定插入点**
**使用断言**
**assert()宏是用于保证满足某个特定条件，用法是：**
**assert(表达式);**
**如果表达式的值为假，整个程序将退出，并输出一条错误信息。如果表达式的值为真则继续执行后面的语句。**
**使用这个宏前需要包含头文件 assert.h**
**例如**

```
#include <stdio.h>
#include <assert.h>
void main()
{
float a,b;
scan("%f %f",&a,&b);
assert(b!=0);
printf("%f\n",a/b);
}
```

**以上的程序要计算 A/B 的值，因此要求 b!=0,所以在程序中使用了 assert()用于确保 b!=0,如果 b==0,则程序会退出。**

```
#include <ctype.h>　　//字符处理
```

isalnum 判断一个字符是否是字符类的数字或者字母
isalpha 判断一个字符是否是字母
isblank 判断一个字符是空白字符（空格和水平制表符 Tab）
iscntrl 判断一个控制符（ascii 码 0-31 之间的字符）
isdigit 判断一个字符是否是字符类的数字
isgraph 判断一个字符是否是可打印字符（ascii 码 33-126 之间的字符）
islower 判断一个字符是否是小写字母
isprint 判断一个字符是否是包含空格在内的可打印字符（ascii 码 32-126 之间的字符）
ispunct 判断一个字符是否是除空格，字母，数字外的标点符号
isspace 判断一个字符是空白字符（空格，换行符(\n)，走纸符(\f)，回车符(\r)，垂直制表符(\v)，水平制表符(\t)）
isupper 判断一个字符是否是大写字母
isxdigit 判断一个字符是否是一个十六进制的数字
tolower 将大些字符转换为小写
toupper 将小写字符转换为大写
isalnum()函数的作用是判断一个字符是否是字符类的数字或者字母：

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    if(isalnum('a'))
        printf("It's True"); // 显示 It's True
    if(isalnum(4))
        printf("It's True"); // 显示 ''
    if(isalnum('4'))
        printf("It's True"); // 显示 It's True
    return 0;
}
```

isalpha()函数的作用是判断一个字符是否是字母：

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    if(isalpha('a'))
        printf("It's True"); // 显示 It's True
    if(isalpha(4))
        printf("It's True"); // 显示 ''
    if(isalpha('4'))
        printf("It's True"); // 显示 ''
```

```
    return 0;
}
```
isblank()函数的作用是判断一个字符是空白字符（空格和水平制表符 Tab），isspace()函数和 isblank()函数类似，但是还包含空格，换行符(\n)，走纸符(\f)，回车符(\r)，垂直制表符(\v)，水平制表符(\t)：
```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    if(isblank(' ')) // 空格
        printf("It's True"); // 显示 It's True
    if(isblank('\t')) // Tab
        printf("It's True"); // 显示 It's True
    if(isblank('\n')) // 换行
        printf("It's True"); // 显示 ''
    if(isblank('\r')) // 回车
        printf("It's True"); // 显示 ''
    if(isspace(' ')) // 空格
        printf("It's True"); // 显示 It's True
    if(isspace('\t')) // Tab
        printf("It's True"); // 显示 It's True
    if(isspace('\n')) // 换行
        printf("It's True"); // 显示 It's True
    if(isspace('\r')) // 回车
        printf("It's True"); // 显示 It's True
    return 0;
}
```
iscntrl()函数的作用是判断一个控制符（ascii 码 0-31 之间的字符）：
```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    if(isblank(' ')) // 空格
        printf("It's True"); // 显示 ''
    if(isblank('\t')) // Tab
        printf("It's True"); // 显示 It's True
    if(isblank('\n')) // 换行
        printf("It's True"); // 显示 It's True
    if(isblank('\r')) // 回车
        printf("It's True"); // 显示 It's True
    return 0;
}
```
isdigit()函数的作用是判断一个字符是否是字符类的数字：
```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    if(isdigit('4'))
        printf("It's True"); // 显示 It's True
    if(isdigit(4))
        printf("It's True"); // 显示 ''
    if(isdigit('a'))
        printf("It's True"); // 显示 ''
    return 0;
}
```
isgraph()函数的作用是判断一个字符是否是可打印字符（ascii 码 33-126 之间的字符），isprint()函数功能和 isgraph()函数类似，区别是 isprint()函数包含空格在内（ascii 码 32-126 之间的字符）：
```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
```

```
    if(isgraph('a'))
        printf("It's True"); // 显示 It's True
    if(isgraph('.'))
        printf("It's True"); // 显示 It's True
    if(isgraph(' ')) // 空格
        printf("It's True"); // 显示 ''
    if(isprint('a'))
        printf("It's True"); // 显示 It's True
    if(isprint('.'))
        printf("It's True"); // 显示 It's True
    if(isprint(' ')) // 空格
        printf("It's True"); // 显示 It's True
    return 0;
}
```

islower()函数的作用是判断一个字符是否是小写字母，isupper()函数的作用是判断一个字符是否是大写字母：

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    if(islower('a'))
        printf("It's True"); // 显示 It's True
    if(islower('A'))
        printf("It's True"); // 显示 ''
    if(isupper('a'))
        printf("It's True"); // 显示 ''
    if(isupper('A'))
        printf("It's True"); // 显示 It's True
    return 0;
}
```

ispunct()函数的作用是判断一个字符是否是除空格，字母，数字外的标点符号：

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    if(ispunct('a'))
        printf("It's True"); // 显示 ''
    if(ispunct('.'))
        printf("It's True"); // 显示 It's True
    if(ispunct('<'))
        printf("It's True"); // 显示 It's True
    return 0;
}
```

isxdigit()函数的作用是判断一个字符是否是一个十六进制的数字：

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
    if(isxdigit('4'))
        printf("It's True"); // 显示 It's True
    if(isxdigit('xE'))
        printf("It's True"); // 显示 It's True
    if(isxdigit('xF'))
        printf("It's True"); // 显示 ''
    return 0;
}
```

tolower()函数的作用是将大些字符转换为小写，toupper()函数的作用是将小写字符转换为大写：

```
#include <stdio.h>
#include <ctype.h>
int main(void)
{
```

```
char n,m,i,j;
n = tolower('A');
m = tolower('a');
i = toupper('a');
j = toupper('.');
        printf("%c %c %c %c", n, m, i, j); // 显示 a a A .
return 0;
#include <errno.h>          //定义错误码
#include <float.h>          //浮点数处理
#include <fstream.h>        //文件输入/输出
#include <iomanip.h>        //参数化输入/输出
#include <iostream.h>       //数据流输入/输出
#include <limits.h>         //定义各种数据类型最值常量
#include <locale.h>         //定义本地化函数
#include <math.h>           //定义数学函数
#include <stdio.h>          //定义输入/输出函数
<stdio.h>下面的类型，宏，函数都是分类的
其他：
      size_t sizeof 返回的值
      NULL     空指针
文件：
      FILE     文件的类型
     fpos_t 文件中指针的位置
      EOF      文件末尾<0
      FILENAME_MAX    文件名最大值>0
      FOPEN_MAX       同时打开文件的最大值>8
      SEEK_SET        文件头
     SEEK_CUR         文件当前位置
      SEEK_END        文件末尾
打开文件
FILE *fopen(const char *filename,const char *mode);
      更改当前流相关的文件
     FILE *freopen(const char *filename,const char *mode,FILE *stream);
关闭文件
      int fclose(FILE *stream);
      清除流中的错误标志或文件末尾标志
      void clearerr(FILE *stream);
      测试流上的文件末尾标志
      int feof(FILE *stream);
      测试流上的错误标志
      int ferror(FILE *stream);
      将一个字符放回到流中
      int ungetc(int c, FILE *stream);
      从流中读一个字符
      int fgetc(FILE *stream);
      int getc(FILE *stream);/*  与 fgetc 相同但是可以用宏实现该函数  */
写一个字符到一个流
      int fputc(int c, FILE *stream);
      int putc(int c, FILE *stream);
      从流中获取一个字符串
      char *fgets(char *s, int n, FILE *stream);
      写一个字符串到一个流
      int fputs(const char *s, FILE *stream);
      打印一个格式化数据到一个流
      int fprintf(FILE *stream,const char *format, ...);
      使用一个参量列表指针格式化到流的数据
      int vfprintf(FILE *stream,const char *format, va_list ap);
      从一个流中读取格式化数据
      int fscanf(FILE *stream, const char *format, ...);
      从一个流中读数据
```

size_t fread(char *buffer,size_t size,size_t count,FILE *stream);
写数据到一个流
int fwrite(const char *buffer, size_t size, size_t count,
FILE *stream);
获取流的文件位置指示符
int fgetpos(FILE *stream, fpos_t *pos);
设置流位置指示符
int fsetpos(FILE *stream, const fpos_t *pos);
移动文件指针到一个指定的位置
int fseek(FILE *stream, long offset, int origin);
获得文件指针相对于文件头的偏移量
long ftell(FILE *stream);
重新定位一个文件指针到文件开头
void rewind(FILE *steam);
删除一个文件
int remove(const char *path);
更改一个文件或目录
int rename(const char *oldname, const char *newname);
缓冲区:
_IOFBF
_IOLBF
_IONBF 缓冲区类型
BUFSIZE 缓冲区尺寸>=256
刷新一个流并清空与流相关的缓冲区的内容
int fflush(FILE *stream);
控制流的缓冲区,已经被 setvbuf 代替
void setbuf(FILE *stream, char *buffer);
控制流的缓冲区类型和缓冲区大小
int setvbuf(FILE *stream, char *buffer, int mode, size_t size);
将一个格式化数据写入一个字符串
int sprintf(char *buffer, const char *format, ...);
从字符串中读格式化数据
int sscanf(const char *buffer, const char *format, ...);
从参量列表指针格式化到字符串
int vsprintf(char *buffer, const char *format, va_list ap);
临时文件
L_tmpnam 临时文件名长度>0
TMP_MAX 产生唯一文件名的最大数目>=25
以二进制读写的方式建立一个临时文件
FILE *tmpfile(void);
建立一个临时文件名
char *tmpname(char *string);
标准流:
stdin 标准输入流
stdout 标准输出流
stderr 标准错误输出流
从 stdin 获得一个字符
int getchar(void);
把字符写道 stdout
int putchar(int c);
从 stdin 中获取一行
char *gets(char *buffer);
写一个字符串到 stdout
int puts(const char *string);
打印一个错误消息到 stderr
void perror(const char *error);
打印格式化数据到 stdout
int printf(const char *format, ...);
从 stdin 读格式化数据
int scanf(const char *format, ...);

从参量列表指针格式化到 stdout
int vprintf(const char *format, va_list ap);

```
#include <stdlib.h>          //定义杂项函数及内存分配函数
#include <string.h>          //字符串处理
#include <strstrea.h>        //基于数组的输入/输出
#include <time.h>            //定义关于时间的函数
#include <wchar.h>           //宽字符处理及输入/输出
#include <wctype.h>          //宽字符分类
```

标准 C++ （同上的不再注释）

```
#include <algorithm>         //STL 通用算法
#include <bitset>            //STL 位集容器
#include <cctype>
#include <cerrno>
#include <clocale>
#include <cmath>
#include <complex>           //复数类
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <deque>             //STL 双端队列容器
#include <exception>         //异常处理类
#include <fstream>
#include <functional>        //STL 定义运算函数（代替运算符）
#include <limits>
#include <list>              //STL 线性列表容器
#include <map>               //STL 映射容器
#include <iomanip>
#include <ios>               //基本输入/输出支持
#include <iosfwd>            //输入/输出系统使用的前置声明
#include <iostream>
#include <istream>           //基本输入流
#include <ostream>           //基本输出流
#include <queue>             //STL 队列容器
#include <set>               //STL 集合容器
#include <sstream>           //基于字符串的流
#include <stack>             //STL 堆栈容器
#include <stdexcept>         //标准异常类
#include <streambuf>         //底层输入/输出支持
#include <string>            //字符串类
#include <utility>           //STL 通用模板类
#include <vector>            //STL 动态数组容器
#include <cwchar>
#include <cwctype>

using namespace std;

//////////////////////////////////////////////////////////////////////////////
```

C99 增加

```
#include <complex.h>         //复数处理
#include <fenv.h>            //浮点环境
#include <inttypes.h>        //整数格式转换
#include <stdbool.h>         //布尔环境
#include <stdint.h>          //整型环境
#include <tgmath.h>          //通用类型数学宏
```

分类函数,所在函数库为 ctype.h
int isalpha(int ch)　若 ch 是字母('A'-'Z','a'-'z')返回非 0 值,否则返回 0
int isalnum(int ch)　若 ch 是字母('A'-'Z','a'-'z')或数字('0'-'9')
　　返回非 0 值,否则返回 0
int isascii(int ch)　若 ch 是字符(ASCII 码中的 0-127)返回非 0 值,否则返回 0
int iscntrl(int ch)　若 ch 是作废字符(0x7F)或普通控制字符(0x00-0x1F)
　　返回非 0 值,否则返回 0
int isdigit(int ch)　若 ch 是数字('0'-'9')返回非 0 值,否则返回 0
int isgraph(int ch)　若 ch 是可打印字符(不含空格)(0x21-0x7E)返回非 0 值,否则返回 0
int islower(int ch)　若 ch 是小写字母('a'-'z')返回非 0 值,否则返回 0
int isprint(int ch)　若 ch 是可打印字符(含空格)(0x20-0x7E)返回非 0 值,否则返回 0
int ispunct(int ch)　若 ch 是标点字符(0x00-0x1F)返回非 0 值,否则返回 0
int isspace(int ch)　若 ch 是空格(' '),水平制表符('\t'),回车符('\r'),
　　走纸换行('\f'),垂直制表符('\v'),换行符('\n')
　　返回非 0 值,否则返回 0
int isupper(int ch)　若 ch 是大写字母('A'-'Z')返回非 0 值,否则返回 0
int isxdigit(int ch)　若 ch 是 16 进制数('0'-'9','A'-'F','a'-'f')返回非 0 值,
　　否则返回 0
int tolower(int ch)　若 ch 是大写字母('A'-'Z')返回相应的小写字母('a'-'z')
int toupper(int ch)　若 ch 是小写字母('a'-'z')返回相应的大写字母('A'-'Z')

数学函数,所在函数库为 math.h、stdlib.h、string.h、float.h
int　　abs(int i)　　返回整型参数 i 的绝对值
double　cabs(struct complex znum)　　返回复数 znum 的绝对值
double　fabs(double x)　　返回双精度参数 x 的绝对值
long　　labs(long n)　　返回长整型参数 n 的绝对值
double　exp(double x)　　返回指数函数 ex 的值
double frexp(double value,int *eptr)　返回 value=x*2n 中 x 的值,n 存贮在 eptr 中
double ldexp(double value,int exp);　返回 value*2exp 的值
double　log(double x)　　返回 logex 的值
double log10(double x)　　返回 log10x 的值
double　pow(double x,double y)　　返回 xy 的值
double pow10(int p)　　返回 10p 的值
double　sqrt(double x)　　返回+ √x 的值
double　acos(double x)　　返回 x 的反余弦 cos-1(x)值,x 为弧度
double　asin(double x)　　返回 x 的反正弦 sin-1(x)值,x 为弧度
double　atan(double x)　　返回 x 的反正切 tan-1(x)值,x 为弧度
double atan2(double y,double x)　　返回 y/x 的反正切 tan-1(x)值,y 的 x 为弧度
double　cos(double x)　　返回 x 的余弦 cos(x)值,x 为弧度
double　sin(double x)　　返回 x 的正弦 sin(x)值,x 为弧度
double　tan(double x)　　返回 x 的正切 tan(x)值,x 为弧度
double　cosh(double x)　　返回 x 的双曲余弦 cosh(x)值,x 为弧度
double　sinh(double x)　　返回 x 的双曲正弦 sinh(x)值,x 为弧度
double　tanh(double x)　　返回 x 的双曲正切 tanh(x)值,x 为弧度
double hypot(double x,double y)　　返回直角三角形斜边的长度(z),
　　x 和 y 为直角边的长度,z2=x2+y2
double　ceil(double x)　　返回不小于 x 的最小整数
double floor(double x)　　返回不大于 x 的最大整数
void　　srand(unsigned seed)　　初始化随机数发生器
int　　rand()　　产生一个随机数并返回这个数
double　poly(double x,int n,double c[])从参数产生一个多项式
double　modf(double value,double *iptr)将双精度数 value 分解成尾数和阶
double　fmod(double x,double y)　　返回 x/y 的余数
double frexp(double value,int *eptr)　将双精度数 value 分成尾数和阶
double　atof(char *nptr)　　将字符串 nptr 转换成浮点数并返回这个浮点数
double　atoi(char *nptr)　　将字符串 nptr 转换成整数并返回这个整数
double　atol(char *nptr)　　将字符串 nptr 转换成长整数并返回这个整数
char　　*ecvt(double value,int ndigit,int *decpt,int *sign)
　　将浮点数 value 转换成字符串并返回该字符串

char *fcvt(double value,int ndigit,int *decpt,int *sign)
将浮点数 value 转换成字符串并返回该字符串
char *gcvt(double value,int ndigit,char *buf)
将数 value 转换成字符串并存于 buf 中,并返回 buf 的指针
char *ultoa(unsigned long value,char *string,int radix)
将无符号整型数 value 转换成字符串并返回该字符串,radix 为转换时所用基数
char *ltoa(long value,char *string,int radix)
将长整型数 value 转换成字符串并返回该字符串,radix 为转换时所用基数
char *itoa(int value,char *string,int radix)
将整数 value 转换成字符串存入 string,radix 为转换时所用基数
double atof(char *nptr) 将字符串 nptr 转换成双精度数,并返回这个数,错误返回 0
int atoi(char *nptr) 将字符串 nptr 转换成整型数, 并返回这个数,错误返回 0
long atol(char *nptr) 将字符串 nptr 转换成长整型数,并返回这个数,错误返回 0
double strtod(char *str,char **endptr)将字符串 str 转换成双精度数,并返回这个数,
long strtol(char *str,char **endptr,int base)将字符串 str 转换成长整型数,
并返回这个数,
int matherr(struct exception *e)
用户修改数学错误返回信息函数(没有必要使用)
double _matherr(_mexcep why,char *fun,double *arg1p,
double *arg2p,double retval)
用户修改数学错误返回信息函数(没有必要使用)
unsigned int _clear87() 清除浮点状态字并返回原来的浮点状态
void _fpreset() 重新初使化浮点数学程序包
unsigned int _status87() 返回浮点状态字

目录函数,所在函数库为 dir.h、dos.h
int chdir(char *path) 使指定的目录 path（如:"C:\\WPS"）变成当前的工作目录,成功返回 0
int findfirst(char *pathname,struct ffblk *ffblk,int attrib)查找指定的文件,成功返回 0
pathname 为指定的目录名和文件名,如"C:\\WPS\\TXT"
ffblk 为指定的保存文件信息的一个结构,定义如下:

```
struct ffblk
{
  char ff_reserved[21]; /*DOS 保留字*/
  char ff_attrib;          /*文件属性*/
  int   ff_ftime;          /*文件时间*/
  int   ff_fdate;          /*文件日期*/
  long ff_fsize;           /*文件长度*/
  char ff_name[13];        /*文件名*/
}
```

attrib 为文件属性,由以下字符代表

| | |
|---|---|
| FA_RDONLY 只读文件 | FA_LABEL 卷标号 |
| FA_HIDDEN 隐藏文件 | FA_DIREC 目录 |
| FA_SYSTEM 系统文件 | FA_ARCH 档案 |

例:

```
struct ffblk ff;
findfirst("*.wps",&ff,FA_RDONLY);
```

int findnext(struct ffblk *ffblk) 取匹配 finddirst 的文件,成功返回 0
void fumerge(char *path,char *drive,char *dir,char *name,char *ext)
此函数通过盘符 drive(C:、A:等),路径 dir(\TC、\BC\LIB 等),
文件名 name(TC、WPS 等),扩展名 ext(.EXE、.COM 等)组成一个文件名
存与 path 中.
int fnsplit(char *path,char *drive,char *dir,char *name,char *ext)
此函数将文件名 path 分解成盘符 drive(C:、A:等),路径 dir(\TC、\BC\LIB 等),

文件名 name(TC、WPS 等),扩展名 ext(.EXE、.COM 等),并分别存入相应的变量中.
int getcurdir(int drive,char *direc) 此函数返回指定驱动器的当前工作目录名称
drive 指定的驱动器(0=当前,1=A,2=B,3=C 等)
direc 保存指定驱动器当前工作路径的变量 成功返回 0
char *getcwd(char *buf,iint n) 此函数取当前工作目录并存入 buf 中,直到 n 个字
节长为为止.错误返回 NULL
int getdisk() 取当前正在使用的驱动器,返回一个整数(0=A,1=B,2=C 等)
int setdisk(int drive) 设置要使用的驱动器 drive(0=A,1=B,2=C 等),
返回可使用驱动器总数
int mkdir(char *pathname) 建立一个新的目录 pathname,成功返回 0
int rmdir(char *pathname) 删除一个目录 pathname,成功返回 0
char *mktemp(char *template) 构造一个当前目录上没有的文件名并存于 template 中
char *searchpath(char *pathname) 利用 MSDOS 找出文件 filename 所在路径,
,此函数使用 DOS 的 PATH 变量,未找到文件返回 NULL

进程函数,所在函数库为 stdlib.h、process.h
void abort() 此函数通过调用具有出口代码 3 的_exit 写一个终止信息于 stderr,
并异常终止程序。无返回值
int exec…装入和运行其它程序
int execl( char *pathname,char *arg0,char *arg1,…,char *argn,NULL)
int execle( char *pathname,char *arg0,char *arg1,…,
char *argn,NULL,char *envp[])
int execlp( char *pathname,char *arg0,char *arg1,…,NULL)
int execlpe(char *pathname,char *arg0,char *arg1,…,NULL,char *envp[])
int execv( char *pathname,char *argv[])
int execve( char *pathname,char *argv[],char *envp[])
int execvp( char *pathname,char *argv[])
int execvpe(char *pathname,char *argv[],char *envp[])
exec 函数族装入并运行程序 pathname，并将参数
arg0(arg1,arg2,argv[],envp[])传递给子程序,出错返回-1
在 exec 函数族中,后缀 l、v、p、e 添加到 exec 后，
所指定的函数将具有某种操作能力
有后缀 p 时，函数可以利用 DOS 的 PATH 变量查找子程序文件。
l 时，函数中被传递的参数个数固定。
v 时，函数中被传递的参数个数不固定。
e 时，函数传递指定参数 envp，允许改变子进程的环境，
无后缀 e 时，子进程使用当前程序的环境。

void _exit(int status)终止当前程序,但不清理现场
void exit(int status) 终止当前程序,关闭所有文件,写缓冲区的输出(等待输出),
并调用任何寄存器的"出口函数",无返回值

int spawn…运行子程序
int spawnl( int mode,char *pathname,char *arg0,char *arg1,…,
char *argn,NULL)
int spawnle( int mode,char *pathname,char *arg0,char *arg1,…,
char *argn,NULL,char *envp[])
int spawnlp( int mode,char *pathname,char *arg0,char *arg1,…,
char *argn,NULL)
int spawnlpe(int mode,char *pathname,char *arg0,char *arg1,…,
char *argn,NULL,char *envp[])
int spawnv( int mode,char *pathname,char *argv[])
int spawnve( int mode,char *pathname,char *argv[],char *envp[])
int spawnvp( int mode,char *pathname,char *argv[])
int spawnvpe(int mode,char *pathname,char *argv[],char *envp[])
spawn 函数族在 mode 模式下运行子程序 pathname,并将参数
arg0(arg1,arg2,argv[],envp[])传递给子程序.出错返回-1
mode 为运行模式
mode 为 P_WAIT 表示在子程序运行完后返回本程序
P_NOWAIT 表示在子程序运行时同时运行本程序(不可用)

P_OVERLAY 表示在本程序退出后运行子程序
在 spawn 函数族中,后缀 l、v、p、e 添加到 spawn 后，
所指定的函数将具有某种操作能力
有后缀 p 时，函数利用 DOS 的 PATH 查找子程序文件
l 时，函数传递的参数个数固定.
v 时，函数传递的参数个数不固定.
e 时，指定参数 envp 可以传递给子程序,允许改变子程序运行环境.
当无后缀 e 时,子程序使用本程序的环境.

int system(char *command) 将 MSDOS 命令 command 传递给 DOS 执行

转换子程序,函数库为 math.h、stdlib.h、ctype.h、float.h
char *ecvt(double value,int ndigit,int *decpt,int *sign)
将浮点数 value 转换成字符串并返回该字符串
char *fcvt(double value,int ndigit,int *decpt,int *sign)
将浮点数 value 转换成字符串并返回该字符串
char *gcvt(double value,int ndigit,char *buf)
将数 value 转换成字符串并存于 buf 中,并返回 buf 的指针
char *ultoa(unsigned long value,char *string,int radix)
将无符号整型数 value 转换成字符串并返回该字符串,radix 为转换时所用基数
char *ltoa(long value,char *string,int radix)
将长整型数 value 转换成字符串并返回该字符串,radix 为转换时所用基数
char *itoa(int value,char *string,int radix)
将整数 value 转换成字符串存入 string,radix 为转换时所用基数
double atof(char *nptr) 将字符串 nptr 转换成双精度数,并返回这个数,错误返回 0
int atoi(char *nptr) 将字符串 nptr 转换成整型数, 并返回这个数,错误返回 0
long atol(char *nptr) 将字符串 nptr 转换成长整型数,并返回这个数,错误返回 0
double strtod(char *str,char **endptr)将字符串 str 转换成双精度数,并返回这个数,
long strtol(char *str,char **endptr,int base)将字符串 str 转换成长整型数,
并返回这个数,
int toascii(int c) 返回 c 相应的 ASCII
int tolower(int ch) 若 ch 是大写字母('A'-'Z')返回相应的小写字母('a'-'z')
int _tolower(int ch) 返回 ch 相应的小写字母('a'-'z')
int toupper(int ch) 若 ch 是小写字母('a'-'z')返回相应的大写字母('A'-'Z')
int _toupper(int ch) 返回 ch 相应的大写字母('A'-'Z')

诊断函数,所在函数库为 assert.h、math.h
void assert(int test) 一个扩展成 if 语句那样的宏，如果 test 测试失败，
就显示一个信息并异常终止程序,无返回值
void perror(char *string) 本函数将显示最近一次的错误信息，格式如下：
字符串 string:错误信息
char *strerror(char *str) 本函数返回最近一次的错误信息,格式如下:
字符串 str:错误信息
int matherr(struct exception *e)
用户修改数学错误返回信息函数(没有必要使用)
double _matherr(_mexcep why,char *fun,double *arg1p,
double *arg2p,double retval)
用户修改数学错误返回信息函数(没有必要使用)

输入输出子程序,函数库为 io.h、conio.h、stat.h、dos.h、stdio.h、signal.h
int kbhit() 本函数返回最近所敲的按键
int fgetchar() 从控制台(键盘)读一个字符，显示在屏幕上
int getch() 从控制台(键盘)读一个字符，不显示在屏幕上
int putch() 向控制台(键盘)写一个字符
int getchar() 从控制台(键盘)读一个字符，显示在屏幕上
int putchar() 向控制台(键盘)写一个字符
int getche() 从控制台(键盘)读一个字符，显示在屏幕上
int ungetch(int c) 把字符 c 退回给控制台(键盘)
char *cgets(char *string) 从控制台(键盘)读入字符串存于 string 中
int scanf(char *format[,argument…])从控制台读入一个字符串,分别对各个参数进行

赋值,使用 BIOS 进行输出

int vscanf(char *format,Valist param)从控制台读入一个字符串,分别对各个参数进行
赋值,使用 BIOS 进行输出,参数从 Valist param 中取得

int cscanf(char *format[,argument…])从控制台读入一个字符串,分别对各个参数进行
赋值,直接对控制台作操作,比如显示器在显示时字符时即为直接写频方式显示

int sscanf(char *string,char *format[,argument,…])通过字符串 string,分别对各个
参数进行赋值

int vsscanf(char *string,char *format,Vlist param)通过字符串 string,分别对各个
参数进行赋值,参数从 Vlist param 中取得

int puts(char *string) 发关一个字符串 string 给控制台(显示器),
使用 BIOS 进行输出

void cputs(char *string) 发送一个字符串 string 给控制台(显示器),
直接对控制台作操作,比如显示器即为直接写频方式显示

int printf(char *format[,argument,…]) 发送格式化字符串输出给控制台(显示器)
使用 BIOS 进行输出

int vprintf(char *format,Valist param) 发送格式化字符串输出给控制台(显示器)
使用 BIOS 进行输出,参数从 Valist param 中取得

int cprintf(char *format[,argument,…]) 发送格式化字符串输出给控制台(显示器),
直接对控制台作操作,比如显示器即为直接写频方式显示

int vcprintf(char *format,Valist param)发送格式化字符串输出给控制台(显示器),
直接对控制台作操作,比如显示器即为直接写频方式显示,
参数从 Valist param 中取得

int sprintf(char *string,char *format[,argument,…])
将字符串 string 的内容重新写为格式化后的字符串

int vsprintf(char *string,char *format,Valist param)
将字符串 string 的内容重新写为格式化后的字符串,参数从 Valist param 中取得

int rename(char *oldname,char *newname)将文件 oldname 的名称改为 newname

int ioctl(int handle,int cmd[,int *argdx,int argcx])
本函数是用来控制输入/输出设备的，请见下表：

| cmd 值 | 功能 |
|---|---|
| 0 | 取出设备信息 |
| 1 | 设置设备信息 |
| 2 | 把 argcx 字节读入由 argdx 所指的地址 |
| 3 | 在 argdx 所指的地址写 argcx 字节 |
| 4 | 除把 handle 当作设备号(0=当前,1=A,等)之外,均和 cmd=2 时一样 |
| 5 | 除把 handle 当作设备号(0=当前,1=A,等)之外,均和 cmd=3 时一样 |
| 6 | 取输入状态 |
| 7 | 取输出状态 |
| 8 | 测试可换性;只对于 DOS 3.x |
| 11 | 置分享冲突的重算计数;只对 DOS 3.x |

int (*ssignal(int sig,int(*action)())()执行软件信号(没必要使用)

int gsignal(int sig) 执行软件信号(没必要使用)

int _open(char *pathname,int access)为读或写打开一个文件,
按后按 access 来确定是读文件还是写文件,access 值见下表

| access 值 | 意义 |
|---|---|
| O_RDONLY | 读文件 |
| O_WRONLY | 写文件 |
| O_RDWR | 即读也写 |
| O_NOINHERIT | 若文件没有传递给子程序,则被包含 |
| O_DENYALL | 只允许当前处理必须存取的文件 |
| O_DENYWRITE | 只允许从任何其它打开的文件读 |
| O_DENYREAD | 只允许从任何其它打开的文件写 |
| O_DENYNONE | 允许其它共享打开的文件 |

int open(char *pathname,int access[,int permiss])为读或写打开一个文件,
按后按 access 来确定是读文件还是写文件,access 值见下表

| access 值 | 意义 |
|---|---|
| O_RDONLY | 读文件 |
| O_WRONLY | 写文件 |
| O_RDWR | 即读也写 |
| O_NDELAY | 没有使用;对 UNIX 系统兼容 |
| O_APPEND | 即读也写,但每次写总是在文件尾添加 |
| O_CREAT | 若文件存在,此标志无用;若不存在,建新文件 |
| O_TRUNC | 若文件存在,则长度被截为 0,属性不变 |
| O_EXCL | 未用;对 UNIX 系统兼容 |
| O_BINARY | 此标志可显示地给出以二进制方式打开文件 |
| O_TEXT | 此标志可用于显示地给出以文本方式打开文件 |

permiss 为文件属性,可为以下值:
S_IWRITE 允许写 S_IREAD 允许读 S_IREAD|S_IWRITE 允许读、写

int creat(char *filename,int permiss) 建立一个新文件 filename，并设定读写性。permiss 为文件读写性，可以为以下值
S_IWRITE 允许写 S_IREAD 允许读 S_IREAD|S_IWRITE 允许读、写

int _creat(char *filename,int attrib) 建立一个新文件 filename，并设定文件属性。attrib 为文件属性，可以为以下值
FA_RDONLY 只读 FA_HIDDEN 隐藏 FA_SYSTEM 系统

int creatnew(char *filenamt,int attrib) 建立一个新文件 filename，并设定文件属性。attrib 为文件属性，可以为以下值
FA_RDONLY 只读 FA_HIDDEN 隐藏 FA_SYSTEM 系统

int creattemp(char *filenamt,int attrib) 建立一个新文件 filename，并设定文件属性。attrib 为文件属性，可以为以下值
FA_RDONLY 只读 FA_HIDDEN 隐藏 FA_SYSTEM 系统

int read(int handle,void *buf,int nbyte)从文件号为 handle 的文件中读 nbyte 个字符存入 buf 中

int _read(int handle,void *buf,int nbyte)从文件号为 handle 的文件中读 nbyte 个字符存入 buf 中,直接调用 MSDOS 进行操作.

int write(int handle,void *buf,int nbyte)将 buf 中的 nbyte 个字符写入文件号为 handle 的文件中

int _write(int handle,void *buf,int nbyte)将 buf 中的 nbyte 个字符写入文件号为 handle 的文件中

int dup(int handle) 复制一个文件处理指针 handle,返回这个指针

int dup2(int handle,int newhandle) 复制一个文件处理指针 handle 到 newhandle

int eof(int *handle)检查文件是否结束,结束返回 1,否则返回 0

long filelength(int handle) 返回文件长度，handle 为文件号

int setmode(int handle,unsigned mode)本函数用来设定文件号为 handle 的文件的打开方式

int getftime(int handle,struct ftime *ftime) 读取文件号为 handle 的文件的时间，并将文件时间存于 ftime 结构中，成功返回 0,ftime 结构如下：

```
struct ftime
{
  unsigned ft_tsec:5; /*秒*/
  unsigned ft_min:6;  /*分*/
  unsigned ft_hour:5; /*时*/
  unsigned ft_day:5;  /*日*/
  unsigned ft_month:4;/*月*/
  unsigned ft_year:1; /*年-1980*/
}
```

int setftime(int handle,struct ftime *ftime) 重写文件号为 handle 的文件时间,新时间在结构 ftime 中.成功返回 0.结构 ftime 如下:

```
struct ftime
{
  unsigned ft_tsec:5; /*秒*/
  unsigned ft_min:6;   /*分*/
  unsigned ft_hour:5; /*时*/
  unsigned ft_day:5;   /*日*/
  unsigned ft_month:4;/*月*/
  unsigned ft_year:1; /*年-1980*/
}
```

long lseek(int handle,long offset,int fromwhere) 本函数将文件号为 handle 的文件的指针移到 fromwhere 后的第 offset 个字节处.
SEEK_SET 文件开关 SEEK_CUR 当前位置 SEEK_END 文件尾
long tell(int handle) 本函数返回文件号为 handle 的文件指针,以字节表示
int isatty(int handle)本函数用来取设备 handle 的类型
int lock(int handle,long offset,long length) 对文件共享作封锁
int unlock(int handle,long offset,long length) 打开对文件共享的封锁
int close(int handle) 关闭 handle 所表示的文件处理,handle 是从_creat、creat、creatnew、creattemp、dup、dup2、_open、open 中的一个处调用获得的文件处理
成功返回 0 否则返回-1,可用于 UNIX 系统
int _close(int handle) 关闭 handle 所表示的文件处理,handle 是从_creat、creat、creatnew、creattemp、dup、dup2、_open、open 中的一个处调用获得的文件处理
成功返回 0 否则返回-1,只能用于 MSDOS 系统

FILE *fopen(char *filename,char *type) 打开一个文件 filename,打开方式为 type，并返回这个文件指针，type 可为以下字符串加上后缀

| type | 读写性 | 文本/2 进制文件 | 建新/打开旧文件 |
|---|---|---|---|
| r | 读 | 文本 | 打开旧的文件 |
| w | 写 | 文本 | 建新文件 |
| a | 添加 | 文本 | 有就打开无则建新 |
| r+ | 读/写 | 不限制 | 打开 |
| w+ | 读/写 | 不限制 | 建新文件 |
| a+ | 读/添加 | 不限制 | 有就打开无则建新 |

可加的后缀为 t、b。加 b 表示文件以二进制形式进行操作，t 没必要使用
例:

```
#include<stdio.h>
main()
{
  FILE *fp;
  fp=fopen("C:\\WPS\\WPS.EXE","r+b");
```

FILE *fdopen(int ahndle,char *type)
FILE *freopen(char *filename,char *type,FILE *stream)
int getc(FILE *stream) 从流 stream 中读一个字符，并返回这个字符
int putc(int ch,FILE *stream)向流 stream 写入一个字符 ch
int getw(FILE *stream) 从流 stream 读入一个整数，错误返回 EOF
int putw(int w,FILE *stream)向流 stream 写入一个整数
int ungetc(char c,FILE *stream) 把字符 c 退回给流 stream，下一次读进的字符将是 c
int fgetc(FILE *stream) 从流 stream 处读一个字符，并返回这个字符
int fputc(int ch,FILE *stream) 将字符 ch 写入流 stream 中
char *fgets(char *string,int n,FILE *stream) 从流 stream 中读 n 个字符存入 string 中
int fputs(char *string,FILE *stream) 将字符串 string 写入流 stream 中
int fread(void *ptr,int size,int nitems,FILE *stream) 从流 stream 中读入 nitems 个长度为 size 的字符串存入 ptr 中
int fwrite(void *ptr,int size,int nitems,FILE *stream) 向流 stream 中写入 nitems 个长度为 size 的字符串,字符串在 ptr 中
int fscanf(FILE *stream,char *format[,argument,…]) 以格式化形式从流 stream 中

读入一个字符串
int vfscanf(FILE *stream,char *format,Valist param) 以格式化形式从流 stream 中
读入一个字符串,参数从 Valist param 中取得
int fprintf(FILE *stream,char *format[,argument,…]) 以格式化形式将一个字符
串写给指定的流 stream
int vfprintf(FILE *stream,char *format,Valist param) 以格式化形式将一个字符
串写给指定的流 stream,参数从 Valist param 中取得
int fseek(FILE *stream,long offset,int fromwhere) 函数把文件指针移到 fromwhere
所指位置的向后 offset 个字节处,fromwhere 可以为以下值:
SEEK_SET 文件开关 SEEK_CUR 当前位置 SEEK_END 文件尾
long ftell(FILE *stream) 函数返回定位在 stream 中的当前文件指针位置,以字节表示
int rewind(FILE *stream) 将当前文件指针 stream 移到文件开头
int feof(FILE *stream) 检测流 stream 上的文件指针是否在结束位置
int fileno(FILE *stream) 取流 stream 上的文件处理，并返回文件处理
int ferror(FILE *stream) 检测流 stream 上是否有读写错误，如有错误就返回 1
void clearerr(FILE *stream) 清除流 stream 上的读写错误
void setbuf(FILE *stream,char *buf) 给流 stream 指定一个缓冲区 buf
void setvbuf(FILE *stream,char *buf,int type,unsigned size)
给流 stream 指定一个缓冲区 buf,大小为 size,类型为 type,type 的值见下表

| type 值 | 意义 |
|---|---|
| _IOFBF | 文件是完全缓冲区,当缓冲区是空时,下一个输入操作将企图填满整个缓冲区.在输出时,在把任何数据写到文件之前,将完全填充缓冲区. |
| _IOLBF | 文件是行缓冲区.当缓冲区为空时,下一个输入操作将仍然企图填整个缓冲区.然而在输出时,每当新行符写到文件,缓冲区就被清洗掉. |
| _IONBF | 文件是无缓冲的.buf 和 size 参数是被忽略的.每个输入操作将直接从文件读,每个输出操作将立即把数据写到文件中. |

int fclose(FILE *stream) 关闭一个流，可以是文件或设备(例如 LPT1)
int fcloseall() 关闭所有除 stdin 或 stdout 外的流
int fflush(FILE *stream) 关闭一个流，并对缓冲区作处理
处理即对读的流，将流内内容读入缓冲区；
对写的流，将缓冲区内内容写入流。成功返回 0
int fflushall() 关闭所有流，并对流各自的缓冲区作处理
处理即对读的流，将流内内容读入缓冲区；
对写的流，将缓冲区内内容写入流。成功返回 0

int access(char *filename,int amode) 本函数检查文件 filename 并返回文件的属性,
函数将属性存于 amode 中，amode 由以下位的组合构成
06 可以读、写 04 可以读 02 可以写 01 执行(忽略的) 00 文件存在
如果 filename 是一个目录,函数将只确定目录是否存在
函数执行成功返回 0,否则返回-1
int chmod(char *filename,int permiss) 本函数用于设定文件 filename 的属性
permiss 可以为以下值
S_IWRITE 允许写 S_IREAD 允许读 S_IREAD|S_IWRITE 允许读、写
int _chmod(char *filename,int func[,int attrib]);
本函数用于读取或设定文件 filename 的属性，
当 func=0 时，函数返回文件的属性；当 func=1 时，函数设定文件的属性
若为设定文件属性，attrib 可以为下列常数之一
FA_RDONLY 只读 FA_HIDDEN 隐藏 FA_SYSTEM 系统
接口子程序,所在函数库为:dos.h、bios.h
unsigned sleep(unsigned seconds)暂停 seconds 微秒(百分之一秒)
int unlink(char *filename)删除文件 filename
unsigned FP_OFF(void far *farptr)本函数用来取远指针 farptr 的偏移量
unsigned FP_SEG(void far *farptr)本函数用来没置远指针 farptr 的段值
void far *MK_FP(unsigned seg,unsigned off)根据段 seg 和偏移量 off 构造一个 far 指针
unsigned getpsp()取程序段前缀的段地址,并返回这个地址
char *parsfnm(char *cmdline,struct fcb *fcbptr,int option)
函数分析一个字符串,通常,对一个文件名来说,是由 cmdline 所指的一个命令行.

文件名是放入一个 FCB 中作为一个驱动器,文件名和扩展名.FCB 是由 fcbptr 所指定的.option 参数是 DOS 分析系统调用时,AL 文本的值.

int absread(int drive,int nsects,int sectno,void *buffer)本函数功能为读特定的磁盘扇区,drive 为驱动器号(0=A,1=B 等),nsects 为要读的扇区数,sectno 为开始的逻辑扇区号,buffer 为保存所读数据的保存空间

int abswrite(int drive,int nsects,int sectno,void *buffer)本函数功能为写特定的磁盘扇区,drive 为驱动器号(0=A,1=B 等),nsects 为要写的扇区数,sectno 为开始的逻辑扇区号,buffer 为保存所写数据的所在空间

void getdfree(int drive,struct dfree *dfreep)本函数用来取磁盘的自由空间,drive 为磁盘号(0=当前,1=A 等).函数将磁盘特性的由 dfreep 指向的 dfree 结构中.dfree 结构如下:

```
struct dfree
{
  unsigned df_avail; /*有用簇个数*/
  unsigned df_total; /*总共簇个数*/
  unsigned df_bsec;   /*每个扇区字节数*/
  unsigned df_sclus; /*每个簇扇区数*/
}
```

char far *getdta() 取磁盘转换地址 DTA

void setdta(char far *dta)设置磁盘转换地址 DTA

void getfat(int drive,fatinfo *fatblkp)
本函数返回指定驱动器 drive(0=当前,1=A,2=B 等)的文件分配表信息并存入结构 fatblkp 中,结构如下:

```
struct fatinfo
{
  char fi_sclus; /*每个簇扇区数*/
  char fi_fatid; /*文件分配表字节数*/
  int   fi_nclus; /*簇的数目*/
  int   fi_bysec; /*每个扇区字节数*/
}
```

void getfatd(struct fatinfo *fatblkp) 本函数返回当前驱动器的文件分配表信息,并存入结构 fatblkp 中,结构如下:

```
struct fatinfo
{
  char fi_sclus; /*每个簇扇区数*/
  char fi_fatid; /*文件分配表字节数*/
  int   fi_nclus; /*簇的数目*/
  int   fi_bysec; /*每个扇区字节数*/
}
```

int bdos(int dosfun,unsigned dosdx,unsigned dosal)本函数对 MSDOS 系统进行调用,dosdx 为寄存器 dx 的值,dosal 为寄存器 al 的值,dosfun 为功能号

int bdosptr(int dosfun,void *argument,unsiigned dosal)本函数对 MSDOS 系统进行调用,argument 为寄存器 dx 的值,dosal 为寄存器 al 的值,dosfun 为功能号

int int86(int intr_num,union REGS *inregs,union REGS *outregs)
执行 intr_num 号中断,用户定义的寄存器值存于结构 inregs 中,
执行完后将返回的寄存器值存于结构 outregs 中.

int int86x(int intr_num,union REGS *inregs,union REGS *outregs,
struct SREGS *segregs)执行 intr_num 号中断,用户定义的寄存器值存于结构 inregs 中和结构 segregs 中,执行完后将返回的寄存器值存于结构 outregs 中.

int intdos(union REGS *inregs,union REGS *outregs)
本函数执行 DOS 中断 0x21 来调用一个指定的 DOS 函数,用户定义的寄存器值存于结构 inregs 中,执行完后函数将返回的寄存器值存于结构 outregs 中

int intdosx(union REGS *inregs,union REGS *outregs,struct SREGS *segregs)
本函数执行 DOS 中断 0x21 来调用一个指定的 DOS 函数,用户定义的寄存器值
存于结构 inregs 和 segregs 中,执行完后函数将返回的寄存器值存于结构 outregs 中
void intr(int intr_num,struct REGPACK *preg)本函数中一个备用的 8086 软件中断接口
它能产生一个由参数 intr_num 指定的 8086 软件中断.函数在执行软件中断前,
从结构 preg 复制用户定义的各寄存器值到各个寄存器.软件中断完成后,
函数将当前各个寄存器的值复制到结构 preg 中.参数如下:
intr_num 被执行的中断号
preg 为保存用户定义的寄存器值的结构,结构如下

```
struct REGPACK
{
  unsigned r_ax,r_bx,r_cx,r_dx;
  unsigned r_bp,r_si,r_di,r_ds,r_es,r_flags;
}
```

函数执行完后,将新的寄存器值存于结构 preg 中
void keep(int status,int size)以 status 状态返回 MSDOS,但程序仍保留于内存中,所占
用空间由 size 决定.
void ctrlbrk(int (*fptr)()) 设置中断后的对中断的处理程序.
void disable() 禁止发生中断
void enable() 允许发生中断
void geninterrupt(int intr_num)执行由 intr_num 所指定的软件中断
void interrupt(* getvect(int intr_num))() 返回中断号为 intr_num 的中断处理程序,
例如: old_int_10h=getvect(0x10);
void setvect(int intr_num,void interrupt(* isr)()) 设置中断号为 intr_num 的中
断处理程序为 isr,例如: setvect(0x10,new_int_10h);
void harderr(int (*fptr)()) 定义一个硬件错误处理程序,
每当出现错误时就调用 fptr 所指的程序
void hardresume(int rescode)硬件错误处理函数
void hardretn(int errcode) 硬件错误处理函数
int inport(int prot) 从指定的输入端口读入一个字,并返回这个字
int inportb(int port)从指定的输入端口读入一个字节,并返回这个字节
void outport(int port,int word) 将字 word 写入指定的输出端口 port
void outportb(int port,char byte)将字节 byte 写入指定的输出端口 port
int peek(int segment,unsigned offset) 函数返回 segment:offset 处的一个字
char peekb(int segment,unsigned offset)函数返回 segment:offset 处的一个字节
void poke(int segment,int offset,char value) 将字 value 写到 segment:offset 处
void pokeb(int segment,int offset,int value) 将字节 value 写到 segment:offset 处
int randbrd(struct fcb *fcbptr,int reccnt)
函数利用打开 fcbptr 所指的 FCB 读 reccnt 个记录.
int randbwr(struct fcb *fcbptr,int reccnt)
函数将 fcbptr 所指的 FCB 中的 reccnt 个记录写到磁盘上
void segread(struct SREGS *segtbl)函数把段寄存器的当前值放进结构 segtbl 中
int getverify() 取检验标志的当前状态(0=检验关闭,1=检验打开)
void setverify(int value)设置当前检验状态,
value 为 0 表示关闭检验,为 1 表示打开检验
int getcbrk()本函数返回控制中断检测的当前设置
int setcbrk(int value)本函数用来设置控制中断检测为接通或断开
当 value=0 时,为断开检测.当 value=1 时,为接开检测

int dosexterr(struct DOSERR *eblkp)取扩展错误.在 DOS 出现错误后,此函数将扩充的
错误信息填入 eblkp 所指的 DOSERR 结构中.该结构定义如下:

```
struct DOSERR
{
  int   exterror;/*扩展错误*/
  char class;    /*错误类型*/
  char action;   /*方式*/
  char locus;    /*错误场所*/
```

```
}
```

int bioscom(int cmd,char type,int port) 本函数负责对数据的通讯工作,

cmd 可以为以下值:

0 置通讯参数为字节 byte 值　　1 发送字符通过通讯线输出

2 从通讯线接受字符　　3 返回通讯的当前状态

port 为通讯端口,port=0 时通讯端口为 COM1,port=1 时通讯端口为 COM2,以此类推

byte 为传送或接收数据时的参数,为以下位的组合:

| byte 值 | 意义 | byte 值 | 意义 | byte 值 | 意义 |
|---|---|---|---|---|---|
| 0x02 | 7 数据位 | 0x03 | 8 数据位 | 0x00 | 1 停止位 |
| 0x04 | 2 停止位 | 0x00 | 无奇偶性 | 0x08 | 奇数奇偶性 |
| 0x18 | 偶数奇偶性 | 0x00 | 110 波特 | 0x20 | 150 波特 |
| 0x40 | 300 波特 | 0x60 | 600 波特 | 0x80 | 1200 波特 |
| 0xA0 | 2400 波特 | 0xC0 | 4800 波特 | 0xE0 | 9600 波特 |

例如:0xE0|0x08|0x00|0x03 即表示置通讯口为 9600 波特,奇数奇偶性,1 停止位,
8 数据位.

函数返回值为一个 16 位整数,定义如下:

第 15 位 超时
第 14 位 传送移位寄存器空
第 13 位 传送固定寄存器空
第 12 位 中断检测
第 11 位 帧错误
第 10 位 奇偶错误
第 9 位 过载运行错误
第 8 位 数据就绪
第 7 位 接收线信号检测
第 6 位 环形指示器
第 5 位 数据设置就绪
第 4 位 清除发送
第 3 位 δ接收线信号检测器
第 2 位 下降边环形检测器
第 1 位 δ数据设置就绪
第 0 位 δ清除发送

int biosdisk(int cmd,int drive,int head,int track,
int sector,int nsects,void *buffer)

本函数用来对驱动器作一定的操作,cmd 为功能号,

drive 为驱动器号(0=A,1=B,0x80=C,0x81=D,0x82=E 等).cmd 可为以下值:

0 重置软磁盘系统.这强迫驱动器控制器来执行硬复位.忽略所有其它参数.
1 返回最后的硬盘操作状态.忽略所有其它参数
2 读一个或多个磁盘扇区到内存.读开始的扇区由 head、track、sector 给出。扇区号由 nsects 给出。把每个扇区 512 个字节的数据读入 buffer
3 从内存读数据写到一个或多个扇区。写开始的扇区由 head、track、sector 给出。扇区号由 nsects 给出。所写数据在 buffer 中，每扇区 512 个字节。
4 检验一个或多个扇区。开始扇区由 head、track、sector 给出。扇区号由 nsects 给出。
5 格式化一个磁道，该磁道由 head 和 track 给出。buffer 指向写在指定 track 上的扇区磁头器的一个表。

以下 cmd 值只允许用于 XT 或 AT 微机：

6 格式化一个磁道，并置坏扇区标志。
7 格式化指定磁道上的驱动器开头。
8 返回当前驱动器参数，驱动器信息返回写在 buffer 中(以四个字节表示)。
9 初始化一对驱动器特性。
10 执行一个长的读，每个扇区读 512 加 4 个额外字节
11 执行一个长的写，每个扇区写 512 加 4 个额外字节
12 执行一个磁盘查找
13 交替磁盘复位

14 读扇区缓冲区
15 写扇区缓冲区
16 检查指定的驱动器是否就绪
17 复核驱动器
18 控制器 RAM 诊断
19 驱动器诊断
20 控制器内部诊
函数返回由下列位组合成的状态字节：
0x00 操作成功
0x01 坏的命令
0x02 地址标记找不到
0x04 记录找不到
0x05 重置失败
0x07 驱动参数活动失败
0x09 企图 DMA 经过 64K 界限
0x0B 检查坏的磁盘标记
0x10 坏的 ECC 在磁盘上读
0x11 ECC 校正的数据错误（注意它不是错误）
0x20 控制器失效
0x40 查找失败
0x80 响应的连接失败
0xBB 出现无定义错误
0xFF 读出操作失败

int biodquip() 检查设备，函数返回一字节，该字节每一位表示一个信息，如下：
第 15 位 打印机号
第 14 位 打印机号
第 13 位 未使用
第 12 位 连接游戏 I/O
第 11 位 RS232 端口号
第 8 位 未使用
第 7 位 软磁盘号
第 6 位 软磁盘号,
00 为 1 号驱动器,01 为 2 号驱动器,10 为 3 号驱动器,11 为 4 号驱动器
第 5 位 初始化
第 4 位 显示器模式
00 为未使用，01 为 40x25BW 彩色显示卡
10 为 80x25BW 彩色显示卡，11 为 80x25BW 单色显示卡
第 3 位 母扦件
第 2 位 随机存贮器容量,00 为 16K,01 为 32K,10 为 48K,11 为 64K
第 1 位 浮点共用处理器
第 0 位 从软磁盘引导

int bioskey(int cmd)本函数用来执行各种键盘操作，由 cmd 确定操作。
cmd 可为以下值：
0 返回敲键盘上的下一个键。若低 8 位为非 0,即为 ASCII 字符；若低 8 位为 0,
则返回扩充了的键盘代码。
1 测试键盘是否可用于读。返回 0 表示没有键可用；否则返回下一次敲键之值。
敲键本身一直保持由下次调用具的 cmd 值为 0 的 bioskey 所返回的值。
2 返回当前的键盘状态，由返回整数的每一个位表示，见下表：

| 位 | 为 0 时意义 | 为 1 时意义 |
|---|---|---|
| 7 | 插入状态 | 改写状态 |
| 6 | 大写状态 | 小写状态 |
| 5 | 数字状态，NumLock 灯亮 | 光标状态，NumLock 灯熄 |
| 4 | ScrollLock 灯亮 | ScrollLock 灯熄 |
| 3 | Alt 按下 | Alt 未按下 |
| 2 | Ctrl 按下 | Ctrl 未按下 |
| 1 | 左 Shift 按下 | 左 Shift 未按下 |

| | | |
|---|---|---|
| 0 | 右 Shift 按下 | 右 Shift 未按下 |

int biosmemory()返回内存大小,以 K 为单位.

int biosprint(int cmd,int byte,int port)控制打印机的输入/输出.

port 为打印机号,0 为 LPT1,1 为 LPT2,2 为 LPT3 等

cmd 可以为以下值:

0 打印字符,将字符 byte 送到打印机

1 打印机端口初始化

2 读打印机状态

函数返回值由以下位值组成表示当前打印机状态

0x01 设备时间超时

0x08 输入/输出错误

0x10 选择的

0x20 走纸

0x40 认可

0x80 不忙碌

int biostime(int cmd,long newtime)计时器控制,cmd 为功能号,可为以下值

0 函数返回计时器的当前值

1 将计时器设为新值 newtime

struct country *country(int countrycmode,struct country *countryp)

本函数用来控制某一国家的相关信息,如日期,时间,货币等.

若 countryp=-1 时,当前的国家置为 countrycode 值(必须为非 0).否则,由 countryp 所指向的 country 结构用下列的国家相关信息填充:

(1)当前的国家(若 countrycode 为 0 或 2)由 countrycode 所给定的国家.

结构 country 定义如下:

```
struct country
{
  int    co_date;              /*日期格式*/
  char co_curr[5];           /*货币符号*/
  char co_thsep[2];          /*数字分隔符*/
  char co_desep[2];          /*小数点*/
  char co_dtsep[2];          /*日期分隔符*/
  char co_tmsep[2];          /*时间分隔符*/
  char co_currstyle;         /*货币形式*/
  char co_digits;            /*有效数字*/
  int (far *co_case)(); /*事件处理函数*/
  char co_dasep;             /*数据分隔符*/
  char co_fill[10];          /*补充字符*/
}
```

co_date 的值所代表的日期格式是:

0 月日年 1 日月年 2 年月日

co_currstrle 的值所代表的货币显示方式是

0 货币符号在数值前,中间无空格

1 货币符号在数值后,中间无空格

2 货币符号在数值前,中间有空格

3 货币符号在数值后,中间有空格

操作函数,所在函数库为 string.h、mem.h

mem…操作存贮数组

void *memccpy(void *destin,void *source,unsigned char ch,unsigned n)

void *memchr(void *s,char ch,unsigned n)

void *memcmp(void *s1,void *s2,unsigned n)

int memicmp(void *s1,void *s2,unsigned n)

void *memmove(void *destin,void *source,unsigned n)

void *memcpy(void *destin,void *source,unsigned n)

void *memset(void *s,char ch,unsigned n)

这些函数,mem…系列的所有成员均操作存贮数组.在所有这些函数中,数组是 n 字节长.
memcpy 从 source 复制一个 n 字节的块到 destin.如果源块和目标块重迭,则选择复制方向,
以例正确地复制覆盖的字节.
memmove 与 memcpy 相同.
memset 将 s 的所有字节置于字节 ch 中.s 数组的长度由 n 给出.
memcmp 比较正好是 n 字节长的两个字符串 s1 和 s2.些函数按无符号字符比较字节,因此,
memcmp("0xFF","\x7F",1)返回值大于 0.
memicmp 比较 s1 和 s2 的前 n 个字节,不管字符大写或小写.
memccpy 从 source 复制字节到 destin.复制一结束就发生下列任一情况:
(1)字符 ch 首选复制到 destin.
(2)n 个字节已复制到 destin.
memchr 对字符 ch 检索 s 数组的前 n 个字节.
返回值:memmove 和 memcpy 返回 destin
memset 返回 s 的值
memcmp 和 memicmp──┬──若 s1<s2 返回值小于 0
├──若 s1=s2 返回值等于 0
└──若 s1>s2 返回值大于 0
memccpy 若复制了 ch,则返回直接跟随 ch 的在 destin 中的字节的一个指针;
否则返回 NULL
memchr 返回在 s 中首先出现 ch 的一个指针;如果在 s 数组中不出现 ch,就返回 NULL.

```
void    movedata(int segsrc,int offsrc,
                 int segdest,int offdest,
                 unsigned numbytes)
```
本函数将源地址(segsrc:offsrc)处的 numbytes 个字节
复制到目标地址(segdest:offdest)
```
void    movemem(void *source,void *destin,unsigned len)
```
本函数从 source 处复制一块长 len 字节的数据到 destin.若源地址和目标地址字符串
重迭,则选择复制方向,以便正确的复制数据.
```
void    setmem(void *addr,int len,char value)
```
本函数把 addr 所指的块的第一个字节置于字节 value 中.

str…字符串操作函数
```
char      stpcpy(char *dest,const char *src)
```
将字符串 src 复制到 dest
```
char      strcat(char *dest,const char *src)
```
将字符串 src 添加到 dest 末尾
```
char      strchr(const char *s,int c)
```
检索并返回字符 c 在字符串 s 中第一次出现的位置
```
int       strcmp(const char *s1,const char *s2)
```
比较字符串 s1 与 s2 的大小,并返回 s1-s2
```
char      strcpy(char *dest,const char *src)
```
将字符串 src 复制到 dest
```
size_t strcspn(const char *s1,const char *s2)
```
扫描 s1,返回在 s1 中有,在 s2 中也有的字符个数
```
char      strdup(const char *s)
```
将字符串 s 复制到最近建立的单元
```
int       stricmp(const char *s1,const char *s2)
```
比较字符串 s1 和 s2,并返回 s1-s2
```
size_t strlen(const char *s)
```
返回字符串 s 的长度
```
char      strlwr(char *s)
```
将字符串 s 中的大写字母全部转换成小写字母,并返回转换后的字符串
```
char      strncat(char *dest,const char *src,size_t maxlen)
```
将字符串 src 中最多 maxlen 个字符复制到字符串 dest 中
```
int       strncmp(const char *s1,const char *s2,size_t maxlen)
```
比较字符串 s1 与 s2 中的前 maxlen 个字符
```
char      strncpy(char *dest,const char *src,size_t maxlen)
```
复制 src 中的前 maxlen 个字符到 dest 中
```
int       strnicmp(const char *s1,const char *s2,size_t maxlen)
```

比较字符串 s1 与 s2 中的前 maxlen 个字符
char strnset(char *s,int ch,size_t n)
将字符串 s 的前 n 个字符置于 ch 中
char strpbrk(const char *s1,const char *s2)
扫描字符串 s1,并返回在 s1 和 s2 中均有的字符个数
char strrchr(const char *s,int c)
扫描最后出现一个给定字符 c 的一个字符串 s
char strrev(char *s)
将字符串 s 中的字符全部颠倒顺序重新排列,并返回排列后的字符串
char strset(char *s,int ch)
将一个字符串 s 中的所有字符置于一个给定的字符 ch
size_t strspn(const char *s1,const char *s2)
扫描字符串 s1,并返回在 s1 和 s2 中均有的字符个数
char strstr(const char *s1,const char *s2)
扫描字符串 s2,并返回第一次出现 s1 的位置
char strtok(char *s1,const char *s2)
检索字符串 s1,该字符串 s1 是由字符串 s2 中定义的定界符所分隔
char strupr(char *s)
将字符串 s 中的小写字母全部转换成大写字母,并返回转换后的字符串

存贮分配子程序,所在函数库为 dos.h、alloc.h、malloc.h、stdlib.h、process.h
int allocmem(unsigned size,unsigned *seg)利用 DOS 分配空闲的内存,
size 为分配内存大小,seg 为分配后的内存指针
int freemem(unsigned seg)释放先前由 allocmem 分配的内存,seg 为指定的内存指针
int setblock(int seg,int newsize)本函数用来修改所分配的内存长度,
seg 为已分配内存的内存指针,newsize 为新的长度

int brk(void *endds)
本函数用来改变分配给调用程序的数据段的空间数量,新的空间结束地址为 endds
char *sbrk(int incr)
本函数用来增加分配给调用程序的数据段的空间数量,增加 incr 个字节的空间

unsigned long coreleft() 本函数返回未用的存储区的长度,以字节为单位
void *calloc(unsigned nelem,unsigned elsize)分配 nelem 个长度为 elsize 的内存空间
并返回所分配内存的指针
void *malloc(unsigned size)分配 size 个字节的内存空间,并返回所分配内存的指针
void free(void *ptr)释放先前所分配的内存,所要释放的内存的指针为 ptr
void *realloc(void *ptr,unsigned newsize)改变已分配内存的大小,ptr 为已分配有内
存区域的指针,newsize 为新的长度,返回分配好的内存指针.

long farcoreleft() 本函数返回远堆中未用的存储区的长度,以字节为单位
void far *farcalloc(unsigned long units,unsigned long unitsz)
从远堆分配 units 个长度为 unitsz 的内存空间,并返回所分配内存的指针
void *farmalloc(unsigned long size)分配 size 个字节的内存空间,
并返回分配的内存指针
void farfree(void far *block)释放先前从远堆分配的内存空间,
所要释放的远堆内存的指针为 block
void far *farrealloc(void far *block,unsigned long newsize)改变已分配的远堆内
存的大小,block 为已分配有内存区域的指针,newzie 为新的长度,返回分配好
的内存指针

时间日期函数,函数库为 time.h、dos.h
在时间日期函数里,主要用到的结构有以下几个:
总时间日期贮存结构 tm

```
┌──────────────────────────────────────┐
│ struct tm                            │
│ {                                    │
│   int tm_sec;     /*秒,0-59*/        │
│   int tm_min;     /*分,0-59*/        │
│   int tm_hour;    /*时,0-23*/        │
```

```
  int tm_mday;    /*天数,1-31*/
  int tm_mon;     /*月数,0-11*/
  int tm_year;    /*自 1900 的年数*/
  int tm_wday;    /*自星期日的天数 0-6*/
  int tm_yday;    /*自 1 月 1 日起的天数,0-365*/
  int tm_isdst; /*是否采用夏时制,采用为正数*/
}
```

日期贮存结构 date

```
struct date
{
  int da_year; /*自 1900 的年数*/
  char da_day; /*天数*/
  char da_mon; /*月数  1=Jan*/
}
```

时间贮存结构 time

```
struct time
{
  unsigned char ti_min;    /*分钟*/
  unsigned char ti_hour; /*小时*/
  unsigned char ti_hund;
  unsigned char ti_sec;    /*秒*/
```

char *ctime(long *clock)
本函数把 clock 所指的时间(如由函数 time 返回的时间)转换成下列格式的字符串:Mon Nov 21 11:31:54 1983\n\0

char *asctime(struct tm *tm)
本函数把指定的 tm 结构类的时间转换成下列格式的字符串:
Mon Nov 21 11:31:54 1983\n\0

double difftime(time_t time2,time_t time1)
计算结构 time2 和 time1 之间的时间差距(以秒为单位)

struct tm *gmtime(long *clock)本函数把 clock 所指的时间(如由函数 time 返回的时间)转换成格林威治时间,并以 tm 结构形式返回

struct tm *localtime(long *clock)本函数把 clock 所指的时间(如函数 time 返回的时间)转换成当地标准时间,并以 tm 结构形式返回

void tzset()本函数提供了对 UNIX 操作系统的兼容性

long dostounix(struct date *dateptr,struct time *timeptr)
本函数将 dateptr 所指的日期,timeptr 所指的时间转换成 UNIX 格式,并返回自格林威治时间 1970 年 1 月 1 日凌晨起到现在的秒数

void unixtodos(long utime,struct date *dateptr,struct time *timeptr)
本函数将自格林威治时间 1970 年 1 月 1 日凌晨起到现在的秒数 utime 转换成 DOS 格式并保存于用户所指的结构 dateptr 和 timeptr 中

void getdate(struct date *dateblk)本函数将计算机内的日期写入结构 dateblk 中以供用户使用

void setdate(struct date *dateblk)本函数将计算机内的日期改成由结构 dateblk 所指定的日期

void gettime(struct time *timep)本函数将计算机内的时间写入结构 timep 中,以供用户使用

void settime(struct time *timep)本函数将计算机内的时间改为由结构 timep 所指的时间

long time(long *tloc)本函数给出自格林威治时间 1970 年 1 月 1 日凌晨至现在所经过的秒数,并将该值存于 tloc 所指的单元中.

int stime(long *tp)本函数将 tp 所指的时间(例如由 time 所返回的时间)写入计算机中.

```
//根据半径计算圆的周长和面积
#include <iostream.h>
const float PI=3.1416;              //声明常量(只读变量)PI 为 3.1416
float fCir_L(float);                //声明自定义函数 fCir_L()的原型
float fCir_S(float);                //声明自定义函数 fCir_S()的原型

//以下是 main()函数
main()
{
      float r,l,s;                  //声明 3 个变量

      cout<<"R=";                   //显示字符串
      cin>>r;                       //键盘输入
      l=fCir_L(r);                  //计算圆的周长，赋值给变量 l
      s=fCir_S(r);                  //计算圆的面积，赋值给变量 s
      cout<<"l="<<l;                //显示计算结果
      cout<<"\ns="<<s;
}

//定义计算圆的周长的函数 fCir_L()
float fCir_L(float x)
{
      float z=-1.0;                 //声明局部变量
      if (x>=0.0)                   //如果参数大于 0，则计算圆的周长
            z=2*PI*x;
      return(z);                    //返回函数值
}

//定义计算圆的面积的函数 fCir_S()
float fCir_S(float x)
{
      float z=-1.0;                 //声明局部变量
      if (x>=0.0)                   //如果参数大于 0，则计算圆的面积
            z=PI*x*x;
      return(z);                    //返回函数值
}
/* Program: P1-2.CPP
  Written by: Hap
  Date written: 02:11:10
*/


#include <iostream.h>
void main(void)
{
      double s1,s2,s3;
      s1=1.5;       /*  对变量 s1 赋值*/
      cout<<"s1="<<s1<<endl;
      /*  对变量 s2 赋值*/       s2=2.5;
      cout<<"s2="<<s2<<endl;
      s3=    /*  对变量 s3 赋值*/    3.5;
      cout<<"s3="<<s3<<endl;

      cout<<"s1+s2+s3="<<s1+s2+s3<<endl;              //计算并显示
      //计算并显示  cout<<"s1+s2+s3="<<s1+s2+s3<<endl;
}
#include <iostream.h>
main()
{
      double    r=1.0;
```

```
    cout<<"r="<<r<<endl;
    double l;
    l=2*3.1416*r;                //计算圆的周长，赋值给变量 l
    cout<<"l="<<l<<endl;        //显示圆的周长
    double s=3.1416*r*r;         //计算圆的面积，赋值给变量 s
    cout<<"s="<<s<<endl;         //显示圆的面积

    cout<<"R=";                  //显示提示输入的信息
    cin>>r;                       //键盘输入
    l=2*3.1416*r;                //计算圆的周长，赋值给变量 l
    cout<<"l="<<l<<endl;        //显示圆的周长
    s=3.1416*r*r;
    cout<<"s="<<s<<endl;         //显示圆的面积
}
#include <iostream.h>        //包含 iostream.h 头文件
void main()
{
    //输出字符常量、变量和字符串
    char c1='A';
    cout<<'W';
    cout<<c1<<endl;
    cout<<"This is a test."<<endl;
    cout<<"-------------------"<<endl;

    //输出整型常量、变量和表达式
    int n=100;
    cout<<10;
    cout<<n;
    cout<<2*n<<endl;     //输出整型表达式
    cout<<"-------------------"<<endl;

    //输出浮点型常量、变量和表达式
    double pi=3.1415926,r=10.0,s=pi*r*r;
    cout<<pi<<endl;
    cout<<r;
    cout<<s;
    cout<<2*r*pi<<endl;          //输出浮点型表达式
    cout<<"-------------------"<<endl;

    //一个 cout 可以输出多项数据
    cout<<'W'<<" "<<c1<<endl;
    cout<<"This is a test."<<endl;
    cout<<"pi="<<pi<<" r="<<r<<" s="<<s<<endl;
}
#include <iostream.h>    //包含 iostream.h 头文件
main()
{
    //输入输出字符
    char c;
    cin>>c;
    cout<<"c="<<c<<endl;

    //输入输出整型数据
    int n;
    cin>>n;
    cout<<"n="<<n<<endl;

    //输入输出浮点型数据
    double x;
    cin>>x;
```

```
    cout<<"x="<<x<<endl;

    //输入提示
    cout<<"n=";
    cin>>n;
    cout<<"n="<<n<<endl;

    //多项输入
    cout<<"c n x"<<endl;
    cin>>c>>n>>x;
    cout<<"c="<<c<<" n="<<n<<" x="<<x<<endl;
}
#include <iostream.h>   //包含 iostream.h 头文件
main()
{
    //声明整型变量
    int a,b;

    //从键盘上为整型变量赋值
    cout<<"a=";
    cin>>a;
    cout<<"b=";
    cin>>b;

    //整型数的算术运算
    cout<<a<<"+"<<b<<"="<<a+b<<endl;
    cout<<a<<"-"<<b<<"="<<a-b<<endl;
    cout<<a<<"*"<<b<<"="<<a*b<<endl;
    cout<<a<<"/"<<b<<"="<<a/b<<endl;
    cout<<a<<"%"<<b<<"="<<a%b<<endl;

    //测试溢出
    short n=32767,m;      //n 取 short 类型的最大值
    cout<<"n="<<n<<endl;
    m=n+1;          //引起溢出
    cout<<"n+1="<<m<<endl;
}
#include <iostream.h>   //包含 iostream.h 头文件
main()
{
    //声明变量，并初始化
    int a=010,b=10,c=0X10;

    //以十进制形式显示数据
    cout<<"DEC:";
    cout<<" a="<<a;
    cout<<" b="<<b;
    cout<<" c="<<c<<endl;

    //以八进制形式显示数据
    cout<<"OCT:";
    cout<<oct;                  //指定八进制输出
    cout<<" a="<<a;
    cout<<" b="<<b;
    cout<<" c="<<c<<endl;

    //以十六进制形式显示数据
    cout<<"HEX:";
    cout<<hex;                  //指定十六进制输出
    cout<<" a="<<a;
```

```
    cout<<" b="<<b;
    cout<<" c="<<c<<endl;

    //八、十和十六进制数混合运算并输出
    cout<<"a+b+c=";
    cout<<dec;                  //恢复十进制输出
    cout<<a+b+c<<endl;

    //测试八、十和十六进制输入
    cout<<"DEC:a="; cin>>a;
    cout<<"OCT:b="; cin>>b;
    cout<<"HEX:a="; cin>>c;
    cout<<"DEC:"<<dec<<endl;                //指定十进制输出
    cout<<"a="<<a<<endl;
    cout<<"b="<<b<<endl;
    cout<<"c="<<c<<endl;
}
#include <iostream.h>   //包含 iostream.h 头文件
#include<iomanip.h>     // iomanip.h 头文件包含 setprecision()的定义
main()
{
    //float 型变量的声明、输入、计算和输出
    float fx,fy;
    cout<<"fx=";
    cin>>fx;
    cout<<"fy=";
    cin>>fy;
    cout<<fx<<"+"<<fy<<"="<<fx+fy<<endl;
    cout<<fx<<"-"<<fy<<"="<<fx-fy<<endl;
    cout<<fx<<"*"<<fy<<"="<<fx*fy<<endl;
    cout<<fx<<"/"<<fy<<"="<<fx/fy<<endl<<endl;
    //cout<<fx<<"%"<<fy<<"="<<fx%fy<<endl;    Error!

    //double 型变量的声明、输入、计算和输出
    float dx,dy;
    cout<<"dx=";
    cin>>dx;
    cout<<"dy=";
    cin>>dy;
    cout<<dx<<"+"<<dy<<"="<<dx+dy<<endl;
    cout<<dx<<"-"<<dy<<"="<<dx-dy<<endl;
    cout<<dx<<"*"<<dy<<"="<<dx*dy<<endl;
    cout<<dx<<"/"<<dy<<"="<<dx/dy<<endl<<endl;
    //cout<<fx<<"%"<<fy<<"="<<fx%fy<<endl;    Error!

    //测试 float 和 double 类型数据的有效位
    fx=10.0;fy=6.0;
    float fz=fx/fy;
    dx=10.0;dy=6.0;
    double dz=dx/dy;
    cout<<"fz=";
    cout<<setprecision(20)<<fx<<"/"<<fy<<"="<<fz<<endl;
    cout<<"dz=";
    cout<<setprecision(20)<<dx<<"/"<<dy<<"="<<dz<<endl<<endl;;

    //float 型溢出
    float x=3.5e14;
    cout<<"x="<<x<<endl;
    cout<<"x*x="<<x*x<<endl;
    cout<<"x*x*x="<<x*x*x<<endl;
```

```
}
#include <iostream.h>   //包含 iostream.h 头文件
main()
{
    //字符类型变量的声明
    char c1='A';
    char c2;

    //字符数据的运算及输出
    c2=c1+32;
    cout<<"c1="<<c1<<endl;
    cout<<"c2="<<c2<<endl;

    //输出字符及 ASCII 码
    cout<<c1<<" : "<<int(c1)<<endl;
    cout<<c2<<" : "<<int(c2)<<endl;
    cout<<'$'<<" : "<<int('$')<<endl;

    //输入字符
    cout<<"c1 c2"<<endl;
    cin>>c1>>c2;
    cout<<"c1="<<c1<<"    c2="<<c2<<endl;
}
#include <iostream.h>   //包含 iostream.h 头文件
main()
{
    char c1='\a',TAB='\t';

    //阵铃一声
    cout<<c1<<endl;

    //使用水平制表符
    cout<<1<<TAB<<2<<TAB<<3<<TAB<<4<<endl;

    //使用双引号
    cout<<"He said \"Thank you\"."<<endl;

    //使用回车换行
    cout<<"abc\n"<<"def"<<'\n';
}

#include <iostream.h>   //包含 iostream.h 头文件
main()
{
    //声明 bool 变量，并初始化
    bool flag1=false,flag2=true;

    //输出布尔常量和变量
    cout<<"false:"<<false<<endl;
    cout<<"true: "<<true<<endl;
    cout<<"flag1="<<flag1<<endl;
    cout<<"flag2="<<flag2<<endl;

    //布尔变量的赋值和输出
    int x=1;
    flag1=x>0;          //存放关系运算结果
    cout<<"flag1="<<flag1<<endl;
    flag2=flag1;       //bool 类型变量相互赋值
    cout<<"flag2="<<flag2<<endl;
```

```
    //布尔变量超界处理
    flag1=100;
    cout<<"flag1="<<flag1<<endl;
    flag2=-100;
    cout<<"flag2="<<flag2<<endl;
}
#include <iostream.h>
const double PI=3.1416;        //声明常量(const 变量)PI 为 3.1416
main()
{
    //声明 3 个变量
    double r,l,s;

    //输入圆的半径
    cout<<"r=";
    cin>>r;

    //计算圆的周长
    l=2*PI*r;
    cout<<"l="<<l<<endl;

    //计算圆的面积
    s=PI*r*r;
    cout<<"s="<<s<<endl;
}

#include<iostream.h>
main()
{
    //定义枚举类型，并指定其枚举元素的值
    enum color {
        RED=3,
        YELLOW=6,
        BLUE=9
    };

    //声明枚举变量 a 和 b,并为枚举变量 a 赋初值
    enum color a=RED;
    color b;            //合法，与 C 语言不同

    // 输出枚举常量
    cout<<"RED="<<RED<<endl;
    cout<<"YELLOW="<<YELLOW<<endl;
    cout<<"BLUE="<<BLUE<<endl;

    //枚举变量的赋值和输出
    b=a;
    a=BLUE;
    cout<<"a="<<a<<endl;
    cout<<"b="<<b<<endl;
    //a=100;    错误!
    //a=6        也错误!

    //枚举变量的关系运算
    b=BLUE;                                    // 枚举变量的赋值运算
    cout<<"a<b="<<(a<b)<<endl;
}
#include <iostream.h>
const double PI=3.1416;        //声明常量(const 变量)PI 为 3.1416
main()
```

```
{
    //声明 3 个变量
    double r=3,l,s;

    //计算圆的周长
    l=2*PI*r;
    cout<<"l="<<l<<endl;

    //计算圆的面积
    s=PI*r*r;
    cout<<"s="<<s<<endl;

    //验证赋值误差
    int il,is;
    il=l;
    is=s;
    cout<<"il="<<il<<endl;
    cout<<"is="<<is<<endl;
}
#include <iostream.h>
main()
{
    //变量声明
    char c;
    double x,y;

    //测试自增
    cout<<"++E and E++ :"<<endl;
    c='B';
    cout<<"c="<<++c<<endl;     //输出 c=C
    c='B';
    cout<<"c="<<c++<<endl;     //输出 c=B
    x=1.5;
    y=5+ ++x;                  //加号后的空格不能少
    cout<<"y="<<y<<endl;       //输出 y=7.5
    x=1.5;
    y=5+x++;
    cout<<"y="<<y<<endl;       //输出 y=6.5
    cout<<"--------------------"<<endl;

    //测试自减
    cout<<"--E and E-- :"<<endl;
    c='B';
    cout<<"c="<<--c<<endl;     //输出 c=A
    c='B';
    cout<<"c="<<c--<<endl;     //输出 c=B
    x=1.5;
    y=5+--x;
    cout<<"y="<<y<<endl;       //输出 y=5.5
    x=1.5;
    y=5+x--;
    cout<<"y="<<y<<endl;       //输出 y=6.5
}
#include <iostream.h>
main()
{
    int a=3, b=2;

    //输出关系表达式
    cout<<a<b<<endl;
```

```
	cout<<(a<b)<<(a>b)<<(a>=b)<<(a==b)<<(a!=b)<<endl;

	bool flag=2*a<b+10;
	cout<<"flag="<<flag;
}
#include <iostream.h>
main()
{
	float a=3.5,b=2.1,c=0;
	cout<<"a="<<a<<"    b="<<b<<"    c="<<c<<endl;

	//与运算
	cout<<"a&&b="<<(a&&b)<<endl;//输出 1
	cout<<"a&&c="<<(a&&c)<<endl;//输出 0

	//或运算
	cout<<"a||b="<<(a||b)<<endl;//输出 1
	cout<<"a||c="<<(a||c)<<endl;//输出 1

	//非运算
	cout<<"!a="<<!a<<endl<<"!c="<<!c<<endl;//输出 0    1

	//关系运算和逻辑运算
	bool flag=a>=0 && a<=5;    //变量 a 在[0,5]区间内
	cout<<"a=>0 && a<=5="<<flag<<endl;//输出 1

	//算术运算、关系运算和逻辑运算
	cout<<"a+5>2*b+2||a<b+3="<<(a+5>2*b+2||a<b+3)<<endl;//输出 1
}
#include <iostream.h>
main()
{
	//按位与运算
	cout<<"24&12="<<(24&12)<<endl;
	//按位异或运算
	cout<<"24^12="<<(24^12)<<endl;
	//按位或运算
	cout<<"24|12="<<(24|12)<<endl;
	//按位取反运算
	cout<<"~24="<<(~24)<<endl;

	//左移位运算
	cout<<"5<<3="<<(5<<3)<<endl;
	cout<<"-5<<3="<<(-5<<3)<<endl;

	//右移位运算
	cout<<"5>>3="<<(5>>3)<<endl;
	cout<<"-5>>3="<<(-5>>3)<<endl;
}
#include <iostream.h>
main()
{
	int a=1,b=1,c=3;
	//显示 a,b,c 的值
	cout<<"a="<<a<<" b="<<b<<" c="<<c<<endl;

	//计算显示(1) b+=a+2*c%5;  的结果
	b+=a+2*c%5;        //相当于表达式语句  b=b+(a+2*c%5);
	cout<<"(1) b="<<b<<endl;
```

```
	//计算显示(2) a<<=c-2*b;  的结果
	a=1,b=1,c=3;
	a<<=c-2*b;		//  相当于表达式语句  a=a<<(c-2*b);
	cout<<"(2) a="<<a<<endl;

	//计算显示(3) a*=b=c=3;的结果
	a=1,b=1,c=3;
	a*=b=c=3;		//相当于语句组  c=3;b=c;a=a*b;
	cout<<"(3) a="<<a<<"    b="<<b<<"    c="<<c<<endl;

	//计算显示(4) a+=b+=c;的结果
	a=1,b=1,c=3;
	a+=b+=c;		//相当于语句组  b=b+c; a=a+b;
	cout<<"(4) a="<<a<<"    b="<<b<<"    c="<<c<<endl;

	//计算显示(5) a-=b=++c+2;的结果
	a=1,b=1,c=3;
	a-=b=++c+2;		//相当于语句组  ++c;b=b+c+2;a=a-b;
	cout<<"(5) a="<<a<<"    b="<<b<<"    c="<<c<<endl;
}
#include <iostream.h>
main()
{
	//用  sizeof  计算各类种常量的字节长度
	cout<<"sizeof('$')="<<sizeof('$')<<endl;
	cout<<"sizeof(1)="<<sizeof(1)<<endl;
	cout<<"sizeof(1.5)="<<sizeof(1.5)<<endl;
	cout<<"sizeof(\"Good!\")="<<sizeof("Good!")<<endl;

	//用 sizeof  计算各类型变量的字节长度
	int i=100;
	char c='A';
	float x=3.1416;
	double p=0.1;
	cout<<"sizeof(i)="<<sizeof(i)<<endl;
	cout<<"sizeof(c)="<<sizeof(c)<<endl;
	cout<<"sizeof(x)="<<sizeof(x)<<endl;
	cout<<"sizeof(p)="<<sizeof(p)<<endl;

	//用 sizeof  计算表达式的字节长度
	cout<<"sizeof(x+1.732)="<<sizeof(x+1.732)<<endl;

	//用  sizeof  计算各类型的字节长度
	cout<<"sizeof(char)="<<sizeof(char)<<endl;
	cout<<"sizeof(int)="<<sizeof(int)<<endl;
	cout<<"sizeof(float)="<<sizeof(float)<<endl;
	cout<<"sizeof(double)="<<sizeof(double)<<endl;

	//用 sizeof  计算数组的字节长度
	char str[]="This is a test.";
	int a[10];
	double xy[10];
	cout<<"sizeof(str)="<<sizeof(str)<<endl;
	cout<<"sizeof(a)="<<sizeof(a)<<endl;
	cout<<"sizeof(xy)="<<sizeof(xy)<<endl;

	//用 sizeof  计算自定义类型的长度
	struct st {
		short num;
		float math_grade;
```

```
        float Chinese_grade;
        float sum_grade;
    };
    st student1;
    cout<<"sizeof(st)="<<sizeof(st)<<endl;
    cout<<"sizeof(student1)="<<sizeof(student1)<<endl;
}
#include <iostream.h>
main()
{
    //声明变量语句中使用顺序运算
    int x, y;

    //计算中使用顺序运算
    x=50;
    y=(x=x-5, x/5);
    cout<<"x="<<x<<endl;
    cout<<"y="<<y<<endl;
}
#include <iostream.h>
main()
{
    //测试表达式类型的转换
    int n=100,m;
    double x=3.791,y;
    cout<<"n*x="<<n*x<<endl;

    //赋值类型转换
    m=x;
    y=n;
    cout<<"m="<<m<<endl;
    cout<<"y="<<y<<endl;

    //强制类型转换
    cout<<"int(x)="<<int(x)<<endl;
    cout<<"(int)x="<<(int)x<<endl;
    cout<<"int(1.732+x)="<<int(1.732+x)<<endl;
    cout<<"(int)1.732+x="<<(int)1.723+x<<endl;
    cout<<"double(100)="<<double(100)<<endl;
}
#include <iostream.h>
main()
{
    float a,b,s;

    cout<<"a b"<<endl;
    cin>>a>>b;          //利用 cin 从键盘上为变量  a,b  赋值
    s=a;
    if (a<b) {
        s=b;             //if 语句中只有这一个语句，可省略花括号
    }
    s=s*s;               //变量 s 中保存 a,b 中较大的一个数的平方
    cout<<"s="<<s;
}

#include <iostream.h>
main()
{
    int x,y;
    cout<<"x=";
```

```
    cin>>x;
    if (x<=0) {                    //满足条件执行
        y=2*x;
        cout<<"y="<<y;          //输出结果
    }
    else   {                      //不满足条件执行
        y=x*x;
        cout<<"y="<<y;        //输出结果
    }
}
#include <iostream.h>
main()
{
    int a,b,c;
    int smallest;
    cout<<"a   b   c"<<endl;
    cin>>a>>b>>c;
    if (a<=b)       //外层条件语句
    {
        if (a<=c)       //内层条件语句
            smallest=a;
        else
            smallest=c;
    }
    else
    {
        if (b<=c)       //内层条件语句
            smallest=b;
        else
            smallest=c;
    }
    cout<<"Smallest="<<smallest<<endl;
}

#include <iostream.h>
main()
{
    int score;

    //从键盘上输入分数
    cout<<"score=";
    cin>>score;

    //用带 else if 的条件语句判断处理
    if (score<0 || score>100)
    {
        cout<<"The score is out of range!"<<endl;
    }
    else if (score>=90)
        cout<<"Your grade is a A."<<endl;
    else if (score>=80)
        cout<<"Your grade is a B."<<endl;
    else if (score>=70)
        cout<<"Your grade is a C."<<endl;
    else if (score>=60)
        cout<<"Your grade is a D."<<endl;
    else
        cout<<"Your grade is a E."<<endl;
}
#include <iostream.h>
```

```
main()
{
	int n;
	cout<<"n=";
	cin>>n;
	if (n>=0 && n<=100 &&n%2==0)
	    cout<<"n="<<n<<endl;
	else
	    cout<<"The "<<n<<" is out of range!"<<endl;
}

#include <iostream.h>
main()
{
	int a,b,Max;
	//输入数据
	cout<<"a=";
	cin>>a;
	cout<<"b=";
	cin>>b;

	//找出较大值
	Max=a>b?a:b;
	cout<<"Max="<<Max<<endl;
}

#include <iostream.h>
main()
{
	int a,b;
	//输入数据
	cout<<"a=";
	cin>>a;
	cout<<"b=";
	cin>>b;

	//除法判断
	if (b!=0 && a%b==0) {
		cout<<b<<" divides "<<a<<endl;
		cout<<"a/b="<<a/b<<endl;
	}
	else
		cout<<b<<" does not divide "<<a<<endl;
}


#include <iostream.h>
main()
{
	//x,y 为操作数，c 为运算符
	int x,y,z;
	char c1;
	cin>>x>>c1>>y;	//c1

	//多路选择语句选择不同表达式计算语句
	switch(c1) {
		case '+':cout<<x<<"+"<<y<<"="<<x+y<<endl;
			break;
		case '-':cout<<x<<"-"<<y<<"="<<x-y<<endl;
			break;
```

```
            case '*':cout<<x<<"*"<<y<<"="<<x*y<<endl;
                    break;
            case '/':cout<<x<<"/"<<y<<"="<<x/y<<endl;
                    break;
            case '%':cout<<x<<"%"<<y<<"="<<x%y<<endl;
                    break;
            default :cout<<"Wrong !"<<endl; //当不符合上述情况时执行本子句
    }
}



#include<iostream.h>
float x=365.5;   //声明全局变量
main() {
    int x=1,y=2;
    double w=x+y;
    {
        double x=1.414,y=1.732,z=3.14;
        cout<<"inner:x="<<x<<endl;
        cout<<"inner:y="<<y<<endl;
        cout<<"inner:z="<<z<<endl;
        cout<<"outer:w="<<w<<endl;
        cout<<"::x="<<::x<<endl;       //访问重名的全局变量
    }
    cout<<"outer:x="<<x<<endl;
    cout<<"outer:y="<<y<<endl;
    cout<<"outer:w="<<w<<endl;

    //cout<<"inner:z="<<z<<endl;无效
    cout<<"::x="<<::x<<endl;       //访问重名的全局变量
}
#include<iostream.h>
main() {
    //显示 1,2,3...10
    for(int i=1;i<=10;i++)
        cout<<i<<" ";
    cout<<endl;

    //显示 10,9,8...1
    for(int j=10;j>=1;j--)
        cout<<j<<" ";
    cout<<endl;

    //显示 1,3,5...9
    for(int k=1;k<=10;k=k+2)
        cout<<k<<" ";
    cout<<endl;

    //显示 ABC...Z
    for(char c='A';c<='Z';c++)
        cout<<c;
    cout<<endl;

    //显示 0,0.1,0.2...1.0
    for(float x=0;x<=1.0;x=x+0.1)
        cout<<x<<" ";
    cout<<endl;

    //显示 0,0.1,0.2...1.0
```

```
    for(float x1=0;x1<=1.0+0.1/2;x1=x1+0.1)
        cout<<x1<<" ";
    cout<<endl;

    //计算 s=1+2+3...+100
    int s=0;
    for(int n=1;n<=100;n++)
        s=s+n;
    cout<<"s="<<s<<endl;
}
#include<iostream.h>
main()
{
    //计算 s=1+2+3...+100
    int s=0,n=1;
    while(n<=100) {
        s=s+n;
        n++;
    }
    cout<<"s="<<s<<endl;

    //累加键盘输入的数据
    double x,sum=0.0;
    cout<<"x=";
    cin>>x;
    while(x!=0) {
        sum+=x;
        cout<<"x=";
        cin>>x;
    }
    cout<<"sum="<<sum<<endl;
}
#include<iostream.h>
main()
{
    //计算 s=1+2+3...+100
    int s=0,n=0;
    do  {
        n++;
        s+=n;
    }while(n<100);
    cout<<"s="<<s<<endl;

    //累加键盘输入的数据
    double x,sum=0.0;
    do {
        cout<<"x=";
        cin>>x;
        sum+=x;
    } while(x!=0);
    cout<<"sum="<<sum<<endl;
}

#include<iostream.h>
main()
{
    //计算和打印打印乘法九九表
    for (int i=1;i<=9;i++) {
        cout<<i;
        for (int j=1;j<=9;j++)
```

```
                cout<<'\t'<<i<<"*"<<j<<"="<<i*j;
            cout<<endl;
        }
}

#include<iostream.h>
main()
{
    int x,sum=0;
    //定义标号 L1
L1: cout<<"x=";
    cin>>x;
    if (x==-1)
        goto L2;            //无条件转移语句，转到 L2 语句处
    else
        sum+=x;
    goto L1;                //无条件转移语句，转到 L1 语句处
    //定义标号 L2
L2: cout<<"sum="<<sum<<endl;
}


#include<iostream.h>
main()
{
    //累加键盘输入的数据
    double x,sum=0.0;
    while(1) {
        cout<<"x=";
        cin>>x;
        if (x<=0) break;
        sum+=x;
    }
    cout<<"sum="<<sum<<endl;
}

#include<iostream.h>
main()
{
    int i;
    for (i=1;i<=20;i++)
   {
        if (i%3==0)     //能被 3 整除的整数，返回进行下次循环
            continue;
        cout<<i<<" ";
    }
    cout<<endl;
}
#include<iostream.h>
main()
{
    //声明数组和变量
    int a[5],i,sum;
    double avg;

    //从键盘上循环为数组赋值
    for (i=0;i<5;i++) {
        cout<<"a["<<i<<"]=";
        cin>>a[i];
    }
```

```
    //直接显示数组元素
    cout<<a[0]<<a[1]<<a[2]<<a[3]<<a[4]<<endl;

    //利用 for 循环显示数组各元素的值
    for (i=0;i<5;i++)
        cout<<a[i]<<"    ";
    cout<<endl;

    //计算数组元素之和,并显示计算结果
    sum=a[0]+a[1]+a[2]+a[3]+a[4];
    cout<<"sum="<<sum<<endl;

    //利用循环计算数组的累加和
    for (sum=0,i=0;i<5;i++)
        sum+=a[i];

    //显示累加和及平均值
    cout<<"sum="<<sum<<endl;
    avg=sum/5.0;
    cout<<"avg="<<avg<<endl;
}
#include<iostream.h>
main()
{
     int i,max,index,a[5];

    //从键盘上为数组赋值
     for (i=0;i<=4;i++)
     {
        cout<<"a["<<i<<"]=";
        cin>>a[i];
     }

    //  利用循环遍历数组，找出最大值的元素及其下标
    max=a[0];
    for (i=0;i<=4;i++)
    {
            if (max<a[i])
            {
                max=a[i];
                index=i;
            }
        }
    cout<<"\nMax="<<max<<"    index="<<index;
}
#include<iostream.h>
#define size 5
main()
{
    //声明变量
    int i,j;
    float t,a[size];

    //从键盘上为数组赋值
    for (i=0;i<size;i++)
    {
        cout<<"a["<<i<<"]=";
        cin>>a[i];
    }
```

```
    //对数组按从小到大顺序排序
    for (i=0;i<size-1;i++)
        for (j=i+1;j<size;j++)
            if (a[i]>a[j])
            {
                t=a[i];
                a[i]=a[j];
                a[j]=t;
            }

    //显示排序结果
    for (i=0;i<size;i++)
        cout<<a[i]<<" ";
    cout<<endl;

    //输入要查找的数据
    int value;
    int found;     //找到为 1，否则为 0
    int   low,high,mid;
    for (i=1;i<=3;i++) {
        cout<<"value=";
        cin>>value;

        //二分法查找数组 a
        found=0;
        low=0;
        high=size-1;
        while(low<=high)
        {
            mid=(high+low)/2;
            if (a[mid]==value)
            {
            found=1;
            break;
            }
            if (a[mid]<value)
                low=mid+1;
            else
                high=mid-1;
        }
        if (found)
            cout<<"The valu found at:a["<<mid<<"]="<<a[mid]<<endl;
        else
            cout<<"The "<<value<<" is not found!"<<endl;
    }
}
#include<iostream.h>
main()
{
    //声明变量
    int i,j;
    float t,a[5];

    //从键盘上为数组赋值
    for (i=0;i<=4;i++)
    {
        cout<<"a["<<i<<"]=";
        cin>>a[i];
    }
```

```
    //对数组按从大到小顺序排序
    for (i=0;i<=3;i++)
        for (j=i+1;j<=4;j++)
            if (a[i]<=a[j])
            {
                t=a[i];
                a[i]=a[j];
                a[j]=t;
            }

    //显示排序结果
    for (i=0;i<=4;i++)
        cout<<a[i]<<" ";
}
#include<iostream.h>
main()
{
    //声明二维数组及变量
    int a[2][3],i,j;

    //从键盘上为数组 a 赋值
     for (i=0;i<2;i++)
        for (j=0;j<3;j++)
        {
            cout<<"a["<<i<<"]["<<j<<"]=";
            cin>>a[i][j];
         }

    //显示数组 a
     for (i=0;i<2;i++) {
        for (j=0;j<3;j++)
        {
            cout<<a[i][j]<<"    ";
        }
       cout<<endl;
    }

    //找出该数组的最大元素及其下标
    int h,l,Max=a[0][0];
     for (i=0;i<2;i++) {
        for (j=0;j<3;j++)
        {
            if (Max<a[i][j]) {
                Max=a[i][j];
                h=i;
                l=j;
            }
        }
    }
     cout<<"Max:"<<"a["<<h<<"]["<<l<<"]="<<a[h][l]<<endl;
}
#include<iostream.h>
main()
{
    //声明字符数组和变量
    char str[6];
    int i;

    //从键盘上输入字符串
```

```
    cout<<"str=";
    cin>>str;
    cout<<str<<endl;

    //按数组和下标变量两种方式显示字符数组
    cout<<str<<endl;
    for (i=0;i<6;i++)
        cout<<str[i];
    cout<<endl;

    //字符串反向输出
    for (i=5;i>=0;i--)
        cout<<str[i];
    cout<<endl;

    //将字符数组变成大写字母后输出
    for (i=0;i<=5;i++)
      str[i]-=32;              //小写字母转换成大写字母
    cout<<str<<endl;           //显示字符串
}
#include<iostream.h>
main()
{
    //声明变量和指针变量
    int a,b,c,*ip;

    //指针变量 ip 指向变量 a
    a=100;
    ip=&a;              //使指针变量  ip  指向变量 a
    cout<<"a="<<a<<endl;
    cout<<"*ip="<<*ip<<endl;
    cout<<"ip="<<ip<<endl;

    //指针变量 ip 指向变量 b
    ip=&b;              //使指针变量  ip  指向变量 b
    b=200;
    cout<<"b="<<b<<endl;
    cout<<"*ip="<<*ip<<endl;
    cout<<"ip="<<ip<<endl;

    //指针变量 ip 指向变量 c
    ip=&c;              //使指针变量  ip  指向变量 b
    *ip=a+b;
    cout<<"c="<<c<<endl;
    cout<<"*ip="<<*ip<<endl;
    cout<<"ip="<<ip<<endl;
}
#include<iostream.h>
main()
{
    //声明数组、变量和指针变量
    int a[2][3],i,j;
    int* ip;

    //从键盘上为数组 a 赋值
    for (i=0;i<2;i++)   //为数组 a 赋值
        for (j=0;j<3;j++)
        {
            cout<<"a["<<i<<"]["<<j<<"]=";
            cin>>a[i][j];
```

```
            }

        //利用下标变量显示数组 a
        for (i=0;i<2;i++) {
            for (j=0;j<3;j++)
            {
                cout<<a[i][j]<<"    ";
            }
            cout<<endl;
        }

        //利用指针变量显示数组 a
        ip=&a[0][0];
        for (i=0;i<2;i++) {
             for (j=0;j<3;j++)
             {
                 cout<<"a["<<i<<"]["<<j<<"]=";
                 cout<<ip<<"    ";
                 cout<<*ip<<endl;
                 ip++;
             }
        }
}
#include<iostream.h>
main()
{
        //声明数组、变量和指针变量
        int a[]={1,2,3,4,5,6};
        int *ip1,*ip2;

        //测试指针的赋值运算
        ip1=a;
        ip2=ip1;
        cout<<"*ip1="<<(*ip1)<<endl;
        cout<<"*ip2="<<(*ip2)<<endl;

        //测试指针的自增自减运算和组合运算
        ip1++;
        ip2+=4;
        cout<<"*ip1="<<(*ip1)<<endl;
        cout<<"*ip2="<<(*ip2)<<endl;

        //测试指针变量之间的关系运算
        int n=ip2>ip1;
        cout<<"ip2>ip1="<<n<<endl;
        cout<<"ip2!=NULL="<<(ip2!=NULL)<<endl;

        //指针变量之间的减法
        n=ip2-ip1;
        cout<<"ip2-ip1="<<n<<endl;
}
#include<iostream.h>
main()
{
        //声明字符型数组和指针变量
        char str[10];
        char *strip=str;

        //输入输出
        cout<<"str=";
```

```
    cin>>str;          //用字符数组输入字符串
    cout<<"str="<<str<<endl;
    cout<<"strip="<<strip<<endl;
    cout<<"strip=";
    cin>>strip;        //用字符指针变量输入字符串
    cout<<"str="<<str<<endl;
    cout<<"strip="<<strip<<endl;

    //利用指针变量改变其指向字符串的内容
    *(strip+2)='l';
    cout<<"str="<<str<<endl;
    cout<<"strip="<<strip<<endl;

    //动态为字符型指针变量分配内存
    strip=new char(100);
    cout<<"strip=";
    cin>>strip; //用字符指针变量输入字符串
    cout<<"str="<<str<<endl;
    cout<<"strip="<<strip<<endl;
}
#include<iostream.h>
main()
{
    // 声明用于存放运动员号码的数组
    int h[]={1001,1002,1003,1004};
    // 声明用于存放运动员成绩的数组
    float x[]={12.3,13.1,11.9,12.1};
    //声明用于存放运动姓名的字符型指针数组
    char *p[]={"Wang hua","Zhang jian","Li wei","Hua ming"};
    //i,j,it 是用做循环控制变量和临时变量
    int i,j,it;
    //ft 用做暂存变量
    float ft;
    //pt 为字符型指针变量用做暂存指针变量
    char *pt;

    //用选择法对数组 x 进行排序，并相应调整数组 h 和 p 中的数据
    for (i=0;i<=3;i++)
        for (j=i+1;j<=3;j++)
            if (x[i]>=x[j]) {
                ft=x[i],x[i]=x[j],x[j]=ft;
                it=h[i],h[i]=h[j],h[j]=it;
                pt=p[i],p[i]=p[j],p[j]=pt;
            }

    //以下打印排序结果
    for (i=0;i<=3;i++)
        cout<<h[i]<<" ,"<<p[i]<<" ,"<<x[i]<<endl;
}
#include<iostream.h>
main()
{
    //声明指针数组
    char *colors[]={"Red","Blue","Yellow","Green"};
    //指向指针的指针变量
    char **pt;

    //通过指向指针的变量访问其指向的内容
    pt=colors;
    for (int i=0;i<=3;i++) {
```

```
            cout<<"pt="<<pt<<endl;
            cout<<"*pt="<<*pt<<endl;
            cout<<"**pt="<<**pt<<endl;
            pt++;
        }
}
#include<iostream.h>
main()
{
    //定义结构类型
    struct      books
    {
    char     title[20];
    char     author[15];
    int      pages;
    float    price;
    } ;

    //声明结构变量
    struct books Zbk={"VC++ ","Zhang",295,35.5};
    books Wbk;

    //对结构变量的输出
    cout<<"Zbk:"<<endl;
    cout<<Zbk.title <<endl;
    cout<<Zbk.author<<endl;
    cout<<Zbk.pages<<endl;
    cout<<Zbk.price<<endl;
    cout<<"--------------------"<<endl;

    //对结构成员的运算
    Zbk.pages+=10;
    Zbk.price+=0.5;
    cout<<"Zbk.pages="<<Zbk.pages<<endl;
    cout<<"Zbk.price="<<Zbk.price<<endl;
    cout<<"--------------------"<<endl;

    //对结构变量的输入输出
    cout<<"Wbk.title =";
    cin>>Wbk.title;
    cout<<"Wbk.author=";
    cin>>Wbk.author;
    cout<<"Wbk.pages=";
    cin>>Wbk.pages;
    cout<<"Wbk.price=";
    cin>>Wbk.price;
    cout<<"Wbk:"<<endl;
    cout<<Wbk.title <<endl;
    cout<<Wbk.author<<endl;
    cout<<Wbk.pages<<endl;
    cout<<Wbk.price<<endl;
    cout<<"--------------------"<<endl;

    //结构变量之间的相互赋值
    books temp;
    temp=Wbk;
    cout<<"temp:"<<endl;
    cout<<temp.title<<endl;
    cout<<temp.author<<endl;
    cout<<temp.pages<<endl;
```

```
	cout<<temp.price<<endl;
}

#include<iostream.h>
main()
{
	int i;
	//定义结构类型
	struct student {
			int    num;
			char    name[10];
			float maths;
			float physics;
			float chemistry;
			double    total;
	};

	 //声明结构数组 st
	 student st[3];

	 //从键盘上为结构数组输入值
	 cout<<"        num    name          maths physics chemistry "<<endl;
	 for (i=0;i<3;i++)
	 {
		cout<<i+1<<"      ";
		cin>>st[i].num;
		cin>>st[i].name;
		cin>>st[i].maths;
		cin>>st[i].physics;
		cin>>st[i].chemistry;
	 }

	//计算每个学生的总成绩
	for (i=0;i<3;i++)
		st[i].total=st[i].maths+st[i].physics+st[i].chemistry;

	//输出结构数组各元素的值
	for (i=0;i<3;i++)
	{
		cout<<"st["<<i<<"]:      ";
		cout<<st[i].num<<'\t';
		cout<<st[i].name<<'\t';
		cout<<st[i].maths<<'\t';
		cout<<st[i].physics<<'\t';
		cout<<st[i].chemistry<<'\t';
		cout<<st[i].total<<endl;
	 }
}
#include<iostream.h>
main()
{
	//定义结构类型
	struct human {
		char name[10];
		int sex;
		int age;
	};

	//声明结构变量和结构指针变量,并初始化
	struct human x={"WangPing",1,30},*p=NULL;
```

```
    //结构指针变量指向对象
    p=&x;

    //显示结构变量的值
    cout<<"x.name="<<x.name<<endl;
    cout<<"x.sex="<<x.sex<<endl;
    cout<<"x.age="<<x.age<<endl;

    //利用结构指针显示结构对象中的数据
    cout<<"(*p).name="<<(*p).name<<endl;
    cout<<"(*p).sex="<<(*p).sex<<endl;
    cout<<"(*p).age="<<(*p).age<<endl;
    cout<<"p->name="<<p->name<<endl;
    cout<<"p->sex="<<p->sex<<endl;
    cout<<"p->age="<<p->age<<endl;

    //通过结构指针为结构对象输入数据
    cout<<"name:";
    cin>>(*p).name;
    cout<<"sex:";
    cin>>(*p).sex;
    cout<<"age:";
    cin>>(*p).age;

    //显示结构变量的值
    cout<<"x.name="<<x.name<<endl;
    cout<<"x.sex="<<x.sex<<endl;
    cout<<"x.age="<<x.age<<endl;
}
include<iostream.h>
main()
{
    //定义结构类型
    struct human {
        char name[10];
        int sex;
        int age;
        };

    //声明结构变量和结构指针,并初始化
    struct human x={"WangPing",1,30},*p=&x;

    //利用结构指针显示结构中的数据
    cout<<"(*p).name="<<(*p).name<<endl;
    cout<<"(*p).sex="<<(*p).sex<<endl;
    cout<<"(*p).age="<<(*p).age<<endl;
    cout<<"-------------------------"<<endl;

    //利用 new 运算符为 p 分配内存
    p=new human;

    //从键盘上为 p 指向的结构对象赋值
    cout<<"p->name=";
    cin>>p->name;
    cout<<"p->sex=";
    cin>>p->sex;
    cout<<"p->age=";
    cin>>p->age;
    cout<<"-------------------------"<<endl;
```

```
    //显示 p 所指结构对象的值
    cout<<"p->name="<<p->name<<endl;
    cout<<"p->sex="<<p->sex<<endl;
    cout<<"p->age="<<p->age<<endl;
    cout<<"-------------------------"<<endl;

    //显示结构变量的值
    cout<<"x.name="<<x.name<<endl;
    cout<<"x.sex="<<x.sex<<endl;
    cout<<"x.age="<<x.age<<endl;

    //释放 p 指向的内存
    delete p;
}
#include<iostream.h>
main()
{
    //定义结构类型
    struct human {
        char name[10];
        int sex;
        int age;
    };

    //声明结构数组和结构指针变量,并初始化
    human x[]={{"WeiPing",1,30},{"LiHua",1,25},{"LiuMin",0,23}},*p=NULL;

    //用下标变量的输出结构数组的元素
    for (int i=0;i<3;i++)
    {
        cout<<x[i].name<<'\t';
        cout<<x[i].sex<<'\t';
        cout<<x[i].age<<endl;
    }
    cout<<"-----------------"<<endl;

    //用结构指针输出结构数组的元素
    for (p=x;p<=&x[2];p++)
    {
        cout<<p->name<<'\t';
        cout<<p->sex<<'\t';
        cout<<p->age<<endl;
    }
}
#include<iostream.h>
main()
{
    //定义一个包含指针成员的结构类型
    struct test {
        char *str;
        int   *ip;
    } x;

    //使用结构变量 x 中的整型指针 ip
    x.ip=new int;       //分配 1 个单元
    *(x.ip)=100;
    cout<<"x.ip:"<<x.ip<<'\t'<<*(x.ip)<<endl;
    cout<<"---------------"<<endl;
    delete x.ip;
```

```
    x.ip=new int[5];       //分配 5 个单元
    for(int i=0;i<5;i++)
        *(x.ip+i)=100+i;
    cout<<"x.ip:"<<endl;
    for(i=0;i<5;i++)
        cout<<x.ip+i<<'\t'<<(*(x.ip+i))<<endl;
    delete x.ip;
    cout<<"----------"<<endl;

    //使用结构变量 x 中的字符型指针 str
    x.str=new char('A');       //分配 1 个单元
    cout<<"x.str:"<<(*x.str)<<endl;
    cout<<"----------"<<endl;
    delete x.str;
    x.str=new char[5];       //分配多个单元
    *x.str='G';
    *(x.str+1)='o';
    *(x.str+2)='o';
    *(x.str+3)='d';
    *(x.str+4)='\0';
    cout<<"x.str:"<<x.str<<endl;
    delete x.str;
    cout<<"----------"<<endl;

    //在声明结构变量时初始化
    test y={"Very Good!",NULL};
    cout<<"y.str:"<<y.str<<endl;
    cout<<"y.ip:"<<y.ip<<endl;
}
#include<iostream.h>
main()
{
    //定义 date 结构
    struct date
    {
        int year;
        int month;
        int day;
    };

    //定义 baby 结构
    struct baby {
        int       num;
        float     weight;
        date      birthday;     // date 为结构类型
    };

    //声明 baby 结构变量并初始化
    baby b1={10001,10,{2002,12,25}};

    //下列是 baby 结构变量 b1 的引用。
    cout<<"b1.num="<<b1.num<<endl;
    cout<<"b1.weight="<<b1.weight<<endl;
    cout<<"b1.birthday.year="<<b1.birthday.year<<endl;
    cout<<"b1.birthday.month="<<b1.birthday.month<<endl;
    cout<<"b1.birthday.day="<<b1.birthday.day<<endl;
    cout<<"--------------------------"<<endl;

    //声明 baby 结构变量 temp,并进行赋值运算
    baby temp;
```

```
    temp=b1;
    cout<<"temp.num="<<temp.num<<endl;
    cout<<"temp.weight="<<temp.weight<<endl;
    cout<<"temp.birthday.year="<<temp.birthday.year<<endl;
    cout<<"temp.birthday.month="<<temp.birthday.month<<endl;
    cout<<"temp.birthday.day="<<temp.birthday.day<<endl;
}
#include<iostream.h>
main()
{
    //定义名为 list 的递归结构
    struct list {
            char        name[10];
            int         sex;
            int         age;
            list        *next;    //成员 next 为指向其自身结构的指针
    };

    //使用递归结构变量
    list L1={"WeiPing",1,35.5,NULL};
    cout<<"L1:"<<endl;
    cout<<"name\t"<<L1.name<<endl;
    cout<<"sex\t"<<L1.sex<<endl;
    cout<<"age\t"<<L1.age<<endl;
    cout<<"next\t"<<L1.next<<endl;
}
#include<iostream.h>
main()
{
    int i;
    //定义名为 student 的递归结构
    struct  student {
            char name[10];
            int  math;
            int  computer;
            float sum;
            student *next;      //next 成员是指向自身的结构指针
    };

    //用 student 声明 3 个结构指针变量
    struct student *head,*tail,*temp;

    //申请第 1 块数据，并设置各结构指针的初值
    temp=new struct student;      //申请内存
    head=temp;    // 头指针
    tail=head;    // 尾指针

    //循环为链表输入数据
    cout<<"\tname      Math     Computer"<<endl;
    for (i=1;;i++) {
        cout<<i<<"\t";
        cin>>temp->name;
        if (temp->name[0]!='*')
        {
            cin>>temp->math>>temp->computer;
            temp->sum=temp->math+temp->computer;
            temp->next=NULL;
            tail=temp;          //设置链表尾指针
         }
         else
```

```
            {
              // 以下是输入结束处理
                delete temp;
                tail->next=NULL;
                break;
            }
          //为下一个学生申请内存
          temp->next=new struct student;
          temp=temp->next;       // 使处理指针 temp 指向新内存块
      }

      //将链表数据从头到尾打印出来
      cout<<"--------------------"<<endl;
      temp=head;
      while (temp!=NULL) {
                cout<<temp->name<<","<<temp->math<<",";
                cout<<temp->computer<<","<<temp->sum<<endl;
                temp=temp->next;
        }
}
#include<iostream.h>
main()
{
      int i;
      //定义名为 student 的递归结构
      struct    student {
                char name[10];
                int    math;
                int    computer;
                float sum;
                student *forw;       //forw 成员是前指针
                student *next;       //next 成员是后指针
      };

      //用 student 声明 3 个结构指针变量
      struct student *head,*tail,*temp;

      //申请第 1 块数据，并设置各结构指针的初值
      temp=new struct student;       //申请内存
      head=temp;          // 头指针
      tail=head;          // 尾指针
      head->forw=NULL;

      //循环为链表记录输入数据
      cout<<"\tname       Math      Computer"<<endl;
      for (i=1;;i++) {
            cout<<i<<"\t";
            cin>>temp->name;
            if (temp->name[0]!='*')
            {
                  cin>>temp->math>>temp->computer;
                  temp->sum=temp->math+temp->computer;
                  temp->next=NULL;
                  tail=temp;          //设置链表尾指针
             }
             else
             {
              // 以下是输入结束处理
                delete temp;
                tail->next=NULL;
```

```
                break;
            }
        //为下一个学生申请内存
        temp->next=new struct student;
        temp->next->forw=temp;     //设置前指针
        temp=temp->next;               //使处理指针 temp 指向新内存块
    }

    // 将链表数据从头到尾打印出来
    cout<<"head------>tail:"<<endl;
    temp=head;
    while (temp!=NULL) {
            cout<<temp->name<<","<<temp->math<<",";
            cout<<temp->computer<<","<<temp->sum<<endl;
            temp=temp->next;
     }

    // 将链表数据从尾到头打印出来
    cout<<"tail------>head:"<<endl;
    temp=tail;
    while (temp!=NULL) {
            cout<<temp->name<<","<<temp->math<<",";
            cout<<temp->computer<<","<<temp->sum<<endl;
            temp=temp->forw;
     }
}
#include<iostream.h>
main()
{
    int i;
    //定义联合类型
    union utag    {
            char       c;
            int        k;
            float      x;
    };

    //声明联合变量
    union utag u;

    // 使用联合变量中的字符型成员
    u.c='*';
    cout<<"u.c="<<u.c<<endl;

    // 使用联合变量中的整型成员
    u.k=1000;
    cout<<"u.k="<<u.k<<endl;

    // 使用联合变量中的浮点型成员
    u.x=3.1416;
    cout<<"u.x="<<u.x<<endl;

    //声明联合变量时初始化
    utag u1={'A'};

    //同时引用联合变量的各成员
    cout<<"u1.c="<<u1.c<<endl;
    cout<<"u1.k="<<u1.k<<endl;
    cout<<"u1.x="<<u1.x<<endl;
}
```

```
#include<iostream.h>
main()
{
    //定义结构类型，并为声明的结构变量赋初值
    struct s_tag {
            short       i;
            float x;
    } sx={100,3.1416};

    //定义联合类型，并为声明的联合变量赋初值
    union     u_tag    {
             short       i;
             float x;
    } ux={1000};

    //输出结构类型和结构变量的有关信息
    cout<<"sizeof(struct s_tag)="<<sizeof(struct s_tag)<<endl;
    cout<<"sx.i="<<sx.i<<endl;
    cout<<"sx.x="<<sx.x<<endl;
    cout<<"sizeof(sx)="<<sizeof(sx)<<endl;
    cout<<"-----------------------------"<<endl;

    //输出联合类型和联合变量的有关信息
    cout<<"sizeof(union u_tag)="<<sizeof(union u_tag)<<endl;
    ux.i=200;
    cout<<"ux.i="<<ux.i<<endl;   //输出联合变量 ux  的 i 成员
    ux.x=123.456;
    cout<<"ux.x="<<ux.x<<endl;   //输出联合变量 ux  的 x 成员
    cout<<"sizeof(ux)="<<sizeof(ux)<<endl;
}
#include<iostream.h>
main()
{
    //自定义类型
    typedef    int    ARRAY_INT[50];
    int i;
    ARRAY_INT a;       //用自定义类型声明数组变量 a

    //以下为数组 a 赋值，并打印
    for (i=0;i<50;i++) {
        if (i%10==0)            //每 10 个数换一次行
           cout<<endl;
        a[i]=i;
        cout<<a[i]<<"\t";
      }
    cout<<endl;
}
#include<iostream.h>
//定义结构类型
struct student
{
    int     num;
    char    name[20];
    float grade;
};
void main(void)
{
    //声明数组
    int i,size;
    char str[]="This is a string.";
```

```
    int int_values[] = {51, 23, 2, 44, 45,0,11};
    float float_values[] = {15.1, 13.3, 22.2, 10.4, 1.5};
    student st_arr[]={101,"WangLin",92,102,"LiPing",85,103,"ZhaoMin",88};

    //显示 char 类型数组元素及其大小
    size=sizeof(str) / sizeof(char);
    cout<<"Number of elements in str: ";
    cout<<size<<endl;
    for(i=0;i<size;i++) {
        cout<<str[i];
    }
    cout<<endl;

    //显示 int 类型数组元素及其大小
    size=sizeof(int_values) / sizeof(int);
    cout<<"Number of elements in int_values: ";
    cout<<size<<endl;
    for(i=0;i<size;i++) {
        cout<<int_values[i]<<" ";
    }
    cout<<endl;

    //显示 float 类型数组元素及其大小
    size=sizeof(float_values) / sizeof(float);
    cout<<"Number of elements in float_values: ";
    cout<<size<<endl;
    for(i=0;i<size;i++) {
        cout<<float_values[i]<<" ";
    }
    cout<<endl;

    //显示 student 类型数组元素及其大小
    size=sizeof(st_arr) / sizeof(student);
    cout<<"Number of elements in st_arr: ";
    cout<<size<<endl;
    for(i=0;i<size;i++) {
        cout<<st_arr[i].num<<" ";
        cout<<st_arr[i].name<<" ";
        cout<<st_arr[i].grade<<endl;
    }
}
#include<iostream.h>
//add()函数的定义，其有返回值
double add(double x,double y)
{
    double z;
    z=x+y;
    cout<<x<<"+"<<y<<"="<<z<<endl;
    return(z);
}

main()
{
    double a=0.5,b=1.0;

    //以不同参数形式调用函数 add()
    cout<<"add(1.5,2.5)="<<add(1.5,2.5)<<endl;
    cout<<"add(a,b)="<<add(a,b)<<endl;
    cout<<"add(2*a,a+b)="<<add(2*a,a+b)<<endl;
    cout<<"----------------------"<<endl;
```

```
	//以表达式方式调用函数 add()
	double c=2*add(a,b);
	cout<<"c="<<c<<endl;
	cout<<"----------------------"<<endl;

	//以语句式方式调用函数 add()
	add(2*a,b);
	cout<<"----------------------"<<endl;

	//用其他类型参数调用函数 add()
	int n=1,m=2;
	cout<<"add("<<n<<","<<m<<")="<<add(n,m)<<endl;
}
#include<iostream.h>
//定义符号函数 sgn(),其返回值为 int 类型
int sgn(double x)
{
	if (x>0) return(1);	//返回出口 1
	if (x<0) return(-1);	//返回出口 2
	return(0);		//返回出口 3
}
//main()函数定义
main()
{
	double x;
	int i;
	for (i=0;i<=2;i++) {
		cout<<"x=";
		cin>>x;
		cout<<"sgn("<<x<<")="<<sgn(x)<<endl;
	}
}
#include<iostream.h>
//函数原型语句可以在这里
//定义 main()函数
main()
{
	//max()函数原型声明语句
	float max(float,float);

	//变量声明语句
	float a,b,Max;

	//输入参数并计算
	cout<<"a=";
	cin>>a;
	cout<<"b=";
	cin>>b;
	Max=max(a,b);	//调用 max()函数
	cout<<"max("<<a<<","<<b<<")="<<Max<<endl;
}
//定义 max()函数
float max(float x,float y)	//max()返回值类型为浮点型
{
	float z;
	z=(x>y)?x:y;
	return(z);
}
```

```
#include<iostream.h>
//定义 f()函数
f(int x,int y)          //f()的参数以值方式传递
{
      ++x;
      --y;
      cout<<"x="<<x<<",y="<<y<<endl;
}
main()   {
      int a,b;

      //设置实际参数的值
      a=b=10;
      //以变量为参数调用 f()函数
      f(a,b);

      //验证实际参数的值
      cout<<"a="<<a<<",b="<<b<<endl;

      //以表达式参数形式调用 f()函数
      f(2*a,a+b);
}
#include<iostream.h>

//定义公共结构类型
struct student {
            int    num;
            char    name[10];
            float maths;
            float physics;
            float chemistry;
            double    total;
};

//定义结构输入函数
input_Rec(struct student *p)     //参数为 student 类型的结构指针变量
{
      cin>>p->num;
      cin>>p->name;
      cin>>p->maths;
      cin>>p->physics;
      cin>>p->chemistry;
}

//定义结构数据交换函数
swap_Rec(struct student *p1,struct student *p2)
{
      struct student x;

      //交换两个记录的数据
      x=*p1;
      *p1=*p2;
      *p2=x;
}

//输出结构的值
put_Rec(struct student *p)
{
      cout<<p->num<<'\t';
      cout<<p->name<<'\t';
```

```
        cout<<p->maths<<'\t';
        cout<<p->physics<<'\t';
        cout<<p->chemistry<<'\t';
        cout<<p->total<<endl;
}

//定义 main()函数
main()
{
        int i,j;
        //  声明结构指针变量和结构数组
        struct student *p1,a[3];

        //输入 3 个学生的数据并计算总成绩
        cout<<"num\tname\tmaths\tphysics\tchemistry"<<endl;
        for (p1=a;p1<=a+2;p1++)    {
                input_Rec(p1);
                p1->total=p1->maths+p1->physics+p1->chemistry;
        }

        //对 3 个学生的数据排序
        for (i=0;i<=2;i++)
                for (j=i+1;j<=2;j++)
                        if (a[i].total<a[j].total)
                                swap_Rec(&a[i],&a[j]);      //交换两个结构变量中的数据
         cout<<"--------------------"<<endl;     //输出一分界线

         //输出排序后的结构数组
        cout<<"num\tname\tmaths\tphysics\tchemistry\ttotal"<<endl;
        for (p1=a;p1<=a+2;p1++)
                 put_Rec(p1);
}
#include<iostream.h>
//定义结构
struct student {
        char    name[10];
        float      grade;
};

//交换 student 类型的数据
void swap(student &x,student &y)          //swap 的参数为引用传递方式
{
        student temp;
        temp=x;
        x=y;
        y=temp;
}

//返回 student 类型的引用，求优者
student& max(student &x,student &y)          //swap 的参数为引用传递方式
{
        return (x.grade>y.grade?x:y);
}

//显示 student 类型的数据
void show(student &x)          //show 的参数为引用传递方式
{
    cout<<x.name<<"    "<<x.grade<<endl;
}
void main()
```

```
{
	student a={"ZhangHua",351.5},b={"WangJun",385};

	//显示 a 和 b 的数据
	cout<<"a:";
	show(a);
	cout<<"b:";
	show(b);
	cout<<"------------------"<<endl;

	//交换 a 和 b 的数据,并显示
	swap(a,b);
	cout<<"a:";
show(a);
	cout<<"b:";
show(b);
	cout<<"------------------"<<endl;

	//计算和显示成绩高者
	student t=max(a,b);
	cout<<"Max:";
	show(t);
}
#include <iostream.h>
//参数带有默认值的函数
disp(int x=1,int y=1,int z=1)
{
	cout<<"参数 1: "<<x<<endl;
	cout<<"参数 2: "<<y<<endl;
	cout<<"参数 3: "<<z<<endl;
	cout<<"------------------"<<endl;
}

//main()函数中测试参数带有默认值的函数 disp()
void main()
{
	disp();
	disp(10);
	disp(10,20);
	disp(10,20,30);
	int a=1,b=2,c=3;
	disp(a,b,c);
}
#include <iostream.h>
//计算字符串长度的函数
int str_len(const char *string)
{
	//char *temp=string;  编译报错!
	//*string='x';            编译报错!
	int i=0;
	while (*(string+i)!=NULL)
		i++;
	return i;
}

//main()函数中测试 str_len()
void main()
{
	char a[]="ABCDE";
	cout<<a<<"\t"<<str_len(a)<<endl;
```

```
    char *str="Hello!";
    cout<<str<<"\t"<<str_len(str)<<endl;
    cout<<"This is a test."<<"\t"<<str_len("This is a test.")<<endl;
}
#include<iostream.h>
void   disp(void);   //这个函数声明语句不能少

//定义 main()函数的参数和返回值类型是 void 类型
void   main(void)
{
    //调用 void 类型函数
    disp();
}
//以下定义 disp()函数
void disp(void)   {
      cout<<" You are welcome."<<endl;
}
#include<iostream.h>
//函数原型语句
int   abs(int x);
long abs(long x);
float abs(float x);

//main()函数的定义
void main(void)
{
    //声明变量
    int i1=32767,i2=-32767;
    long l1=456789,l2=-456789;
    float x1=1.1234,x2=-1.1234;

    //直接在 cout 输出中调用函数
    cout<<abs(i1)<<","<<abs(i2)<<endl;
    cout<<abs(l1)<<","<<abs(l2)<<endl;
    cout<<abs(x1)<<","<<abs(x2)<<endl;
}

//定义 int 型的 abs()函数
int abs(int x) {
    if (x<0)
        return(-x);
    else
        return(x);
}

//定义 long 型的 abs()函数
long abs(long x) {
    if (x<0)
        return(-x);
    else
         return(x);
}

//定义 float 型  abs 函数
float abs(float x) {
    if (x<0.0)
        return(-x);
    else
        return(x);
}
```

```
#include<iostream.h>
//max()为内联函数
inline int max(int x,int y)      //注意 inline 关键字
{
      return x>y?x:y;
}

//定义 main()函数
main()
{
      int a=3,b=5,c;
      c=max(a,b);
      cout<<"max("<<a<<","<<b<<")="<<c<<endl;
      cout<<"max("<<15<<","<<11<<")="<<max(15,11)<<endl;
}
#include<iostream.h>
main()
{
      //函数原型声明
      int fact(int x);
      int n,sn;

      //依次从键盘上输入 3 个正整型数据计算它们的阶乘
      for (int i=1;i<=3;i++)
      {
            cout<<i<<"      n=";
            cin>>n;
            sn=fact(n);
            cout<<n<<"!="<<sn<<endl;
      }
}

//以下是采用递归方法定义的 fact()函数
int fact(int x)
{
    if (x==0) return(1);
        return(x*fact(x-1));    //此处又调用了它自身
}
#include<iostream.h>
//带参数的 main()函数
int main(int argc,char *argv[])
{
      int i;
      for(i=0;i<argc;i++)
           cout<<i<<":"<<argv[i]<<endl;
      return 0;
}
#include<iostream.h>
//用函数原型声明要使用的函数
void show_array1(int*,int);
void show_array2(int a[],int);
void sort(int*,int);
main()
{
      //声明数组并初始化
      int a[]={2,4,6,1,3,5};
      int b[3][3]={{2,4,6},{1,3,5},{0,1,2}};

      //显示数组的值
      cout<<"show_array1(int*,int):"<<endl;
```

```
	show_array1(a,6);
	show_array1(&b[0][0],3*3);

	//用 sort1 排序并显示
	cout<<"sort(int*,int) and show_array1(int*,int): "<<endl;
	sort(a,6);
	show_array1(a,6);
	sort(&b[0][0],3*3);
	show_array1(&b[0][0],9);

	//显示数组的值
	cout<<"show_array2(int a[],int):"<<endl;
	show_array2(a,6);
	show_array2(&b[0][0],3*3);
}

//显示数组,用指针当参数
void show_array1(int *p,int size) {
	for(int i=0;i<size;i++)
		cout<<*(p+i)<<" ";
	cout<<endl;
}

//显示数组,用数组当参数
void show_array2(int a[],int size) {
	for(int i=0;i<size;i++)
		cout<<a[i]<<" ";
	cout<<endl;
}

//对数组按从大到小顺序排序
void sort(int *p,int size) {
	int t;
	for (int i=0;i<size-1;i++)
		for (int j=i+1;j<size;j++)
			if (*(p+i)<=*(p+j))
			{
				t=*(p+i);
				*(p+i)=*(p+j);
				*(p+j)=t;
			}
}
#include<iostream.h>
//定义结构
struct student {
	char	name[10];
	float	grade;
};

//更改 student 数据的 grade 成员,参数形式为引用
void change(student &x,float grade)
{
	x.grade=grade;
}

//更改 student 数据的 grade 成员,参数形式为指针
void change1(student *p,float grade)
{
	p->grade=grade;
}
```

```
//更改 student 类型的数据,普通参数形式
void change2(student x,float grade)
{
	x.grade=grade;
}

//显示 student 类型的数据,参数形式为引用
void show(student &x)
{
	cout<<x.name<<"    "<<x.grade<<endl;
}

//在 main()函数中，测试对结构的处理函数
void main()
{
	student a={"ZhangHua",351.5};

	//显示 a 的数据
	show(a);

	//用 change 修改分数,并显示
	cout<<"change(student &x,float grade):"<<endl;
	change(a,360);
	show(a);

	//用 change1 修改分数,并显示
	cout<<"change1(student *p,float grade):"<<endl;
	change1(&a,375);
	show(a);

	//用 change2 修改分数,并显示
	cout<<"change2(student x,float grade):"<<endl;
	change2(a,380.5);
	show(a);
}
#include<iostream.h>
//定义函数计算数组的和和平均值
void calculate(int a[],int size,int& sum,float& average)
{
	sum=0;
	for (int i=0;i<size;i++) {
		sum+=a[i];
	}
	average=sum/size;
}
//定义显示数组的函数
void put_arr(int a[],int size)
{
	for(int i=0;i<size;i++)
		cout<<a[i]<<" ";
	cout<<endl;
}
main()
{
	//声明数组并初始化
	int asize,bsize;
	int a[]={2,4,6,1,3,5};
	int b[]={1,3,5,7,9,11,13,15};
```

```
	//显示数组的值
	asize=sizeof(a)/sizeof(int);
	cout<<"put_arr(a,asize):"<<endl;
	put_arr(a,asize);
	bsize=sizeof(b)/sizeof(int);
	cout<<"put_arr(b,bsize):"<<endl;
	put_arr(b,bsize);

	//计算数组的和和平均值
	float a_ave,b_ave;
	int a_sum,b_sum;
	cout<<"calculate(a,asize,a_sum,a_ave):"<<endl;
	calculate(a,asize,a_sum,a_ave);
	cout<<"a_sum="<<a_sum;
	cout<<" a_ave="<<a_ave<<endl;

	cout<<"calculate(b,bsize,b_sum,b_ave):"<<endl;
	calculate(b,bsize,b_sum,b_ave);
	cout<<"b_sum="<<b_sum;
	cout<<" b_ave="<<b_ave<<endl;
}
#include<iostream.h>

//参数为函数指针的函数
int get_result(int a, int b, int (*sub)(int,int))
{
	int r;
	r=sub(a,b);
	return r;
}

//计算最大值
int max(int a, int b)
{
	cout<<"In max"<<endl;
	return((a > b) ? a: b);
}

//计算最小值
int min(int a, int b)
{
	cout<<"In min"<<endl;
	return((a < b) ? a: b);
}

//求和
int sum(int a, int b)
{
	cout<<"In sum"<<endl;
	return(a+b);
}

//测试指向函数的指针
void main(void)
{
	int a,b,result;

	//测试 3 次
	for (int i=1;i<=3;i++) {
		cout<<"Input a and b :";
```

```
            cin>>a>>b;

            cout<<i<<"\tget_result("<<a<<","<<b<<", &max):"<<endl;
            result =get_result(a, b, &max);
            cout<<"Max of "<<a<<" and "<<b<<" is "<<result<<endl;

            result = get_result(a, b, &min);
            cout<<"Min of "<<a<<" and "<<b<<" is "<<result<<endl;

            result = get_result(a, b, &sum);
            cout<<"Sum of "<<a<<" and "<<b<<" is "<<result<<endl;
      }
}
#include<iostream.h>
#include<stdio.h>
#define size 3
//定义 book 结构类型
struct book
{
      char     title[20];
      char     author[15];
      int      pages;
      float    price;
};
//book 结构的输入函数
input_book(book& bk,char *name)
{
      cout<<name<<":"<<endl;
      cout<<"title:";
      cin>>bk.title;
      cout<<"author:";
      cin>>bk.author;
      cout<<"pages:";
      cin>>bk.pages;
      cout<<"price:";
      cin>>bk.price;
}
//book 结构的输出函数
output_book(book& bk,char *name)
{
      cout<<name<<":    ";
      cout<<bk.title<<" ";
      cout<<bk.author<<" ";
      cout<<bk.pages<<" ";
      cout<<bk.price<<endl;
}
void main(void)
{
      //声明变量和结构数组
      int i;
      char str[20];
      book bk[size];

      //输入结构数组
      for(i=0;i<size;i++) {
            sprintf(str,"bk[%d]",i+1);
            input_book(bk[i],str);
      }

      //显示结构数组
```

```
        for(i=0;i<size;i++) {
                sprintf(str,"bk[%d]",i+1);
                output_book(bk[i],str);
        }
}
#include<iostream.h>
//声明全局变量并初始化
extern int a[]={1,2,3};
extern float p=3.14;

//在 show()函数中使用外部变量
show()   {
        int i;
        cout<<"In show():"<<endl;
        cout<<"p="<<p<<endl;
        cout<<"a[]: ";
        for (i=0;i<=2;i++)
                cout<<a[i]<<"    ";
        cout<<endl;
        //cout<<"y="<<y<<endl;  编译出错!
}

//声明外部变量并初始化
int y=5678;

//在 main()函数中使用外部变量
main()
{
        //声明局部变量
        int i,p=100;

        //显示重名变量
        cout<<"In main():"<<endl;
        cout<<"p="<<p<<endl;

        //显示全局变量
        cout<<"::p="<<::p<<endl;
        cout<<"a[]: ";
        for (i=0;i<=2;i++)
                cout<<a[i]<<"    ";
        cout<<endl;
        cout<<"y="<<y<<endl;     //编译正确!

        show();   //调用函数
}
#include <iostream.h>
//使用静态变量的计数器函数
count1()
{
        //声明静态变量 i，并置初值为 0。i 在 count()中局部可见
        static int i=0;
        return(++i);
}
//使用局部变量的计数器函数
count2()
{
        int i=0;
        return(++i);
}
//在 main()函数中调用 count()函数
```

```
main()
{
	int i;

	//调用 count1()10 次
	cout<<"count1():"<<endl;
	for (i=1;i<=12;i++)
		cout<<count1()<<"    ";
	cout<<endl;

	//调用 count2()10 次
	cout<<"count2():"<<endl;
	for (i=1;i<=12;i++)
		cout<<count2()<<"    ";
	cout<<endl;
}
// p1-851.cpp  为 main()函数文件
#include<iostream.h>
main()
{
	int i,s=0;
	extern int fact(int x);
	for (i=2;i<=6;i=i+2)
		s+=fact(i);
	cout<<"s="<<s<<endl;
}
// p1-852.cpp 为计算阶乘函数文件
//定义 fact()函数为外部(extern)函数
extern int fact(int x)
{
	int i,t=1;
	if(x==0) return(1);
	for(i=1;i<=x;i++)
		t*=i;
	return(t);
}
#include<iostream.h>
#include<stdio.h>
#include<string.h>
#include<process.h>
main()  {
	//声明变量
	FILE *fp1;
	char str[80];

	//从键盘上任意输入一个字符串
	cout<<"Inupt a string:";
	cin.getline(str,80);

	//以写入方式打开 d.dat 文件
	if ((fp1=fopen("d.dat","w"))==NULL)
	{
	    cout<<"\nCould not open the file."<<endl;
	    cout<<"Exiting program."<<endl;
	    exit(1);     //结束程序执行
	}

	//  写"流"文件
	fputs(str,fp1);
	fputs("\n",fp1);
```

```
    fclose(fp1);                        //关闭文件

    // 以读方式打开 d.dat 文件
    if ((fp1=fopen("d.dat","r"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);    //结束程序执行
    }

    // 循环从"流"文件读取字符,并显示
    char ch;
    while ((ch=fgetc(fp1))!=EOF)
            cout<<ch;
    cout<<endl;
    fclose(fp1); //关闭文件

}
#include<iostream.h>
#include <process.h>
#include<stdio.h>
#include<conio.h>
void main(void)   {
    //变量声明
    char ch;
    FILE *fp1;

    //以写入方式打开 d.dat 文件
    if ((fp1=fopen("d.dat","w"))==NULL)      {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);    //结束程序执行
    }

    //循环从键盘上读取字符,写入"流"文件
    cout<<"char:"<<endl;
    cin>>ch;
    while (ch!='*') {
        fputc(ch,fp1);    //将字符写到 fp1 指向的"流"文件中
        cin>>ch;
    }
    fclose(fp1);   //关闭文件

    // 以读方式打开 d.dat 文件
    if ((fp1=fopen("d.dat","r"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);    //结束程序执行
    }

    // 循环从"流"文件读取字符,并显示
    while ((ch=fgetc(fp1))!=EOF)
            cout<<ch<<"   ";
    cout<<endl;
    fclose(fp1); //关闭文件
}
#include<iostream.h>
#include<stdio.h>
```

```
#include<string.h>
#include<process.h>
main()  {
    //声明变量
    int i=0;
    char p[100];                    // 声明输入缓冲区
    FILE *fp1;                      // 声明文件指针变量

    //以写入方式打开 d.dat 文件
    if ((fp1=fopen("d.dat","w"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);    //结束程序执行
    }

    // 写文件操作
    for (i=1;;i++)  {               //无条件循环
        cout<<i<<" string:";
        cin>>p;                     //从键盘上输入数据
        if (stricmp(p,"end")) {     //如果输入的字符串为 end，则结束循环
            fputs(p,fp1);           //写入文件操作
            fputs("\n",fp1);
        }
        else
            break;                  //退出循环
    }

    fclose(fp1);                    //关闭文件

    // 以读方式打开 d.dat 文件
    if ((fp1=fopen("d.dat","r"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);    //结束程序执行
    }

    // 循环从文件读取字符,并显示
    while (fgets(p,100,fp1)!=NULL)
            cout<<p;
     fclose(fp1); //关闭文件
}
#include<iostream.h>
#include<stdio.h>
#include<string.h>
#include<process.h>
#include<stdlib.h>
#define MAX     10
main()  {
    //声明变量
    int i,n;
    FILE *fp1;                      // 声明文件指针变量

    //以写入方式打开 d.dat 文件
    if ((fp1=fopen("d.dat","w"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);    //结束程序执行
```

```
    }

    // 写文件操作
    for (i=1;i<=MAX;i++)  {
        n=rand();              //产生 1 个整数随机数
        putw(n,fp1);
        cout<<n<<" ";
    }
    cout<<endl<<"--------------------"<<endl;

    fclose(fp1);                          //关闭文件

    // 以读方式打开 d.dat 文件
    if ((fp1=fopen("d.dat","r"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);    //结束程序执行
    }

    // 循环从"流"文件读取字符,并显示
    while ((n=getw(fp1))!=EOF)
            cout<<n<<" ";

    fclose(fp1); //关闭文件
}
#include<iostream.h>
#include<stdio.h>
#include<string.h>
#include<process.h>
#include<stdlib.h>
#define MAX       3
main()  {
    //定义结构类型
    struct student {
        int num;
        char name[10];
        float grade;
    };

    //声明数组和变量
    student st[3];
    int i;
    FILE *fp1;                          // 声明文件指针变量

    //以写入方式打开 d.dat 文件
    if ((fp1=fopen("d.dat","w"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);    //结束程序执行
    }

    //从键盘上读数据,写入文件
    cout<<"      num      name      grade"<<endl;
    for (i=0;i<MAX;i++)  {
        cout<<i+1<<" ";
        cin>>st[i].num;
        cin>>st[i].name;
        cin>>st[i].grade;
```

```
        fprintf(fp1,"%d %s %f\n",st[i].num,st[i].name,st[i].grade);
    }

    fclose(fp1);                            //关闭文件

    // 以读方式打开 d.dat 文件
    if ((fp1=fopen("d.dat","r"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);     //结束程序执行
    }

    // 循环从"流"文件读取字符,并显示
    student t;
    while ((fscanf(fp1, "%d %s %f",&t.num,t.name,&t.grade))!=EOF)    {
            cout<<t.num<<" ";
            cout<<t.name<<" ";
            cout<<t.grade<<endl;
    }

    fclose(fp1); //关闭文件
}
#include<iostream.h>
#include <process.h>
#include <stdlib.h>
#include <stdio.h>
int main(void)
{
    FILE *fpd,*fpw;       // 声明 FILE 结构指针变量
    unsigned char dw;
    int i=0;

    //以二进制读方式打开 Calc.exe 文件
    if((fpd=fopen("C:\WINDOWS\Calc.exe", "rb"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);     //结束程序执行
    }

    // 以二进制写方式打开 test.exe 文件
    if((fpw=fopen("test.exe", "wb+"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);     //结束程序执行
    }

    // 二进制文件读写操作，每次指定读写 1 个字节
    while(!feof(fpd)) {      //使用 feof()判断文件尾
          fread(&dw, 1, 1, fpd);
          fwrite(&dw, 1, 1, fpw);
    }
    // 关闭文件
    fclose(fpd);
    fclose(fpw);

    //执行 Calc.exe 和 Calc.exe 文件
    cout<<"1 Run C:\WINDOWS\Calc.exe"<<endl;
```

```
    system("C:\WINDOWS\Calc.exe");
    cout<<"--------------------"<<endl;
    cout<<"2 Run test.exe!"<<endl;
    system("test.exe");
}
#include<iostream.h>
#include <process.h>
#include<stdio.h>
#include<conio.h>
void main(void) {
    //声明变量
    int i;
    char ch;
    FILE *fp1;

    //以写入方式打开 d.dat 文件
    if ((fp1=fopen("d.dat","w"))==NULL)       {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);     //结束程序执行
    }

    //循环从键盘上读取字符,写入文件
    cout<<"char:";
    cin>>ch;
    while (ch!='*') {
        fputc(ch,fp1);     //将字符写到 fp1 指向的"流"文件中
        cin>>ch;
    }
    cout<<"---------------------"<<endl;
    fclose(fp1);   //关闭文件

    //以读方式打开 d.dat 文件
    if ((fp1=fopen("d.dat","r"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);     //结束程序执行
    }

    //循环从文件读取字符,并显示
    while ((ch=fgetc(fp1))!=EOF)
            cout<<ch;
    cout<<endl<<"---------------------"<<endl;

    //以下按倒序方式读取文件中的字符，并显示
    for (i=-1;;i--) {
        fseek(fp1,i,2);                    //设置文件指针，偏移量为 i,相对文件尾
        if ((ch=fgetc(fp1))!=EOF)
            cout<<ch;
        else
            break;
    }
    cout<<endl<<"---------------------"<<endl;

    //以下读取"流"文件中偶数位置上的字符，并打印
    long position;
    for (i=0;;i=i+2) {
        fseek(fp1,i,0);              //设置文件指针，偏移量为 i,相对文件头
        position=ftell(fp1);
```

```
            if ((ch=fgetc(fp1))==EOF)      //遇到文件尾，则退出，否则打印读取的字符
                  break;
            else {
                cout<<position<<" :"<<ch<<endl;
            }
      }
      cout<<endl;

      fclose(fp1); //关闭文件
}
#include<iostream.h>
#include<stdio.h>
#include<process.h>
#include<stdlib.h>
#define MAX 5

//显示数组的数据
void show_array(double x[],int size) {
      for(int i=0;i<size;i++)
            cout<<x[i]<<" ";
      cout<<endl;
}

//main 函数测试数组数据的文件读写
int main(void)
{
      //声明变量
      FILE *fp;       //  声明 FILE 结构指针变量
      int i;
      double a[MAX]={1.0,1.2,1.4,1.6,1.8};

      //显示数组 a 的数据
      cout<<"a:";
      show_array(a,MAX);

      //打开 d.dat 文件
      if ((fp=fopen("d.dat","wb+"))==NULL)
      {
          cout<<"\nCould not open the file."<<endl;
          cout<<"Exiting program."<<endl;
          exit(1);      //结束程序执行
      }

      //以单个元素对数组进行文件读操作
      for(i=0;i<MAX;i++) {
            fwrite(&a[i], sizeof(double), 1, fp);
      }

      rewind(fp);     //恢复读写指针的位置

      //以单个元素对数组进行文件读操作
      double b[MAX];
      for(i=0;i<MAX;i++) {
          if (!feof(fp))       //使用 feof()判断文件尾
             fread(&b[i], sizeof(double), 1, fp);
          else
                break;
      }
      cout<<"b:";
      show_array(b,MAX);//显示数组 b 的数据
```

```
    fclose(fp); //  关闭文件

    //打开 d1.dat 文件
    if ((fp=fopen("d1.dat","wb+"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);     //结束程序执行
    }

    //将数组当成数据块写入文件
    fwrite(&a, sizeof(double), MAX, fp);

    rewind(fp);     //恢复读写指针的位置

    //将数组当成数据块从文件中读取
    double c[MAX];
    if (!feof(fp))       //使用 feof()判断文件尾
          fread(&c, sizeof(double),MAX,fp);
    cout<<"c:";
    show_array(c,MAX);   //显示数组 c 的数据

    fclose(fp); //  关闭文件
}
#include<iostream.h>
#include<stdio.h>
#include<process.h>
#include<stdlib.h>
#define MAX 5
//定义结构类型
struct student {
          int    num;
          char name[20];
          float grade;
};

//显示 student 结构数据
void show_str(student a,char *name) {
    cout<<name<<":"<<endl;
    cout<<a.num<<" "<<a.name<<" "<<a.grade;
    cout<<endl;
}

//main 函数测试结构数据的文件读写
int main(void)
{
    //声明变量
    FILE *fp;
    //声明 FILE 结构指针变量
    student st={1001,"ZhangBin",85.5};

    //显示 st 结构数据
    show_str(st,"st");

    //打开 d.dat 文件
    if ((fp=fopen("d.dat","wb+"))==NULL)
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
```

```
        exit(1);     //结束程序执行
    }

    //用 fprintf()函数写结构数据到文件
    fprintf(fp,"%d %s %f",st.num,st.name,st.grade);

    rewind(fp);     //恢复读写指针的位置

    //用 fscanf()函数读文件中的数据赋值给结构并显示
    student temp;
    fscanf(fp, "%d %s %f",&temp.num,temp.name,&temp.grade);
    show_str(temp,"temp");
    cout<<"----------------------"<<endl;

    fclose(fp); //  关闭文件

    //将结构数据当成数据块进行读写
    if ((fp=fopen("d1.dat","wb+"))==NULL)    //打开 d1.dat 文件
    {
        cout<<"\nCould not open the file."<<endl;
        cout<<"Exiting program."<<endl;
        exit(1);     //结束程序执行
    }

    //声明结构数组并初始化
    int i;
    student starr[3]={{101,"WangPing",92},{102,"Li",85},{103,"LiuMin",97}};

    //显示结构数组
    for(i=0;i<3;i++)
          show_str(starr[i],"starr");

    //将结构数组当成数据块写入文件
    fwrite(starr, sizeof(student), 3, fp);

    rewind(fp);     //恢复读写指针的位置

    //按数据块从文件中读取数据赋值给结构数组
    student temp_arr[3];
    if (!feof(fp))       //使用 feof()判断文件尾
           fread(temp_arr, sizeof(student),3,fp);
    for(i=0;i<3;i++)
          show_str(temp_arr[i],"temp_arr");

    fclose(fp); //  关闭文件
}
#include<stdio.h>
#include<stdlib.h>
#include<iostream.h>
int main(void)
{
    //声明变量
    char ch;
    char str[20];
    int n;
    float x;

    //用 stdin 从键盘上输入数据
    fprintf(stdout,"ch str\n");
    fscanf(stdin,"%c %s",&ch,str);
```

```
        fprintf(stdout,"n        x \n");
        fscanf(stdin,"%d    %f",&n,&x);
        cout<<"----------------"<<endl;

        //输出显示
        fprintf(stdout,"ch=%c str=%s",ch,str);
        fprintf(stdout,"\nn=%d x=%f",n,x);
        cout<<endl;
}
#include <stdio.h>
void main( void )
{
        int c;
        /* Create an error by writing to standard input. */
        putc( 'A', stdin );
        if( ferror( stdin ) )
        {
           perror( "Write error" );
           clearerr( stdin );
        }

        /* See if read causes an error. */
        printf( "Will input cause an error? " );
        c = getc( stdin );
        if( ferror( stdin ) )
        {
             perror( "Read error" );
             clearerr( stdin );
        }
}
#include<iostream.h>
#include<math.h>      //此预处理指令不可少
const double HD=3.1415926/180;
main() {
        cout<<"x\tsin(x)"<<endl;
        for (int i=0;i<=180;i=i+30)
              cout<<i<<"\t"<<sin(i*HD)<<endl;
}
#include<iostream.h>
//以下是几个简单宏替换预处理指令
#define    YES              1
#define    PI               3.1415926
#define    RAD              PI/180
#define    MESG             "This is a string."

//以下是主程序
main()    {
        //以下各语句使用了宏替换
        cout<<"YES="<<YES<<endl;
        if (YES)
             cout<<"PI="<<PI<<endl;
        cout<<"RAD="<<RAD<<endl;
        cout<<MESG<<endl;
}
#include<iostream.h>
//以下为带参数宏替换的预处理指令
#define    PRINT(k)      cout<<(k)<<endl;
#define    MAX(a,b)      ((a)>(b) ? (a):(b))
main()
{
```

```
    int i=3,j=2;

    //MAX(a,b)宏替换的使用
    cout<<"MAX(10,12)="<<MAX(10,12)<<endl;
    cout<<"MAX(i,j)="<<MAX(i,j)<<endl;
    cout<<"MAX(2*i,j+3)="<<MAX(2*i,j+3)<<endl;

    //PRINT(k)宏替换的使用
    PRINT(5);
    PRINT(MAX(7,i*j));
}
#include<iostream.h>
#define   PI   3.1416
main()  {
    int i=100;
#if 1
    cout<<"i="<<i<<endl;
#endif

#ifdef PI
    cout<<"1  PI="<<PI<<endl;
#endif

#ifndef PI
    cout<<"2  PI="<<PI<<endl;   //此语句不被编译执行
#endif
}

#include<iostream.h>
const int MAX=5;        //假定栈中最多保存 5 个数据

//定义名为 stack 的类，其具有栈功能
class  stack {
    //数据成员
    float  num[MAX];  //存放栈数据的数组
    int  top;              //指示栈顶位置的变量
public:
    //成员函数
    void init(void) { top=0; }      //初始化函数
    void push(float x)              //入栈函数
    {
        if (top==MAX){
            cout<<"Stack is full !"<<endl;
            return;
        };
        num[top]=x;
        top++;
    }
    float pop(void)                 //出栈函数
    {
        top--;
        if (top<0){
        cout<<"Stack is underflow !"<<endl;
        return 0;
        };
        return num[top];
    }
}

//以下是 main()函数，其用 stack 类创建栈对象，并使用了这些对象
```

```
main(void)
{
    //声明变量和对象
    int i;
    float x;
    stack a,b;      //声明(创建)栈对象

    //以下对栈对象初始化
    a.init();
    b.init();

    //以下利用循环和 push()成员函数将 2,4,6,8,10 依次入 a 栈对象
    for (i=1; i<=MAX; i++)
        a.push(2*i);

    //以下利用循环和 pop()成员函数依次弹出 a 栈中的数据并显示
    for (i=1; i<=MAX; i++)
        cout<<a.pop()<<"    ";
    cout<<endl;

    //以下利用循环和 push()成员函数将键盘输入的数据依次入 b 栈
    cout<<"Please input five numbers."<<endl;
    for (i=1; i<=MAX; i++) {
        cin>>x;
        b.push(x);
    }

    //以下利用循环和 pop()成员函数依次弹出 b 栈中的数据并显示
    for (i=1; i<=MAX; i++)
        cout<<b.pop()<<"    ";
}
#include<iostream.h>
const int MAX=5;      //假定栈中最多保存 5 个数据

//定义名为 stack 的具有栈功能的类
class   stack {
    //数据成员
    float   num[MAX];    //存放栈数据的数组
    int   top;                  //指示栈顶位置的变量
public:
    //成员函数
    stack(void)   //初始化函数
    {
        top=0;
        cout<<"Stack initialized."<<endl;
    }
    void push(float x)        //入栈函数
    {
        if (top==MAX){
            cout<<"Stack is full !"<<endl;
            return;
        };
        num[top]=x;
        top++;
    }
    float pop(void)          //出栈函数
    {
        top--;
        if (top<0){
        cout<<"Stack is underflow !"<<endl;
```

```
            return 0;
            };
            return num[top];
        }
}

//以下是 main()函数，其用 stack 类创建栈对象，并使用了这些对象
main(void)
{
        //声明变量和对象
        int i;
        float x;
        stack a,b;        //声明(创建)栈对象并初始化

        //以下利用循环和 push()成员函数将 2,4,6,8,10 依次入 a 栈
        for (i=1; i<=MAX; i++)
              a.push(2.0*i);

        //以下利用循环和 pop()成员函数依次弹出 a 栈中的数据并显示
        for (i=1; i<=MAX; i++)
            cout<<a.pop()<<"    ";
        cout<<endl;

        //以下利用循环和 push()成员函数将键盘输入的数据依次入 b 栈
        cout<<"Please input five numbers."<<endl;
        for (i=1; i<=MAX; i++) {
                cin>>x;
                b.push(x);
        }

        //以下利用循环和 pop()成员函数依次弹出 b 栈中的数据并显示
        for (i=1; i<=MAX; i++)
            cout<<b.pop()<<"    ";
        cout<<endl;
}
#include<iostream.h>
const int MAX=5;          //假定栈中最多保存 5 个数据

//定义名为 stack 的具有栈功能的类
class    stack {
        //数据成员
        float    num[MAX];          //存放栈数据的数组
        int    top;                 //指示栈顶位置的变量
public:
        //成员函数
        stack(char c)               //初始化函数
        {
              top=0;
              cout<<"Stack "<<c<<" initialized."<<endl;
        }
        void push(float x)          //入栈函数
        {
              if (top==MAX){
                    cout<<"Stack is full !"<<endl;
                    return;
              };
              num[top]=x;
              top++;
        }
        float pop(void)                 //出栈函数
```

```
    {
        top--;
        if (top<0){
        cout<<"Stack is underflow !"<<endl;
        return 0;
        };
        return num[top];
    }
}

//以下是 main()函数，其用 stack 类创建栈对象，并使用了这些对象
main(void)
{
    //声明变量和对象
    int i;
    float x;
    stack a('a'),b('b');       //声明(创建)栈对象并初始化

    //以下利用循环和 push()成员函数将 2,4,6,8,10 依次入 a 栈
    for (i=1; i<=MAX; i++)
        a.push(2.0*i);

    //以下利用循环和 pop()成员函数依次弹出 a 栈中的数据并显示
    for (i=1; i<=MAX; i++)
      cout<<a.pop()<<"    ";
    cout<<endl;
}
#include<iostream.h>
main()
{
    //定义一个名为 student 的类
    class student {
            int num;
            char *name;
            float grade;
    public:
          //定义构造函数
        student(int n,char *p,float g): num(n),name(p),grade(g){}
        display(void) {
                cout<<num<<" ,"<<name<<","<<grade<<endl;
            }
    };

    student a(1001,"Liming",95),b(1002,"ZhangHua",96.5);     //创建对象，并初始化
    //student c;   错误，没提供参数

    a.display();                    //显示对象 a 中的数据
    b.display();                    //显示对象 b 中的数据
}
#include <iostream.h>
#include <stdlib.h>
//定义 timer 类
class timer{
    long minutes;
public:
    //无参数构造函数
    timer(void) {
        minutes =0;
    };
    //字符指针参数的构造函数
```

```
        timer(char *m) {
            minutes = atoi(m);
        };
        //整数类型的构造函数
        timer(int h, int m) {
            minutes = 60*h+m ;
        };
        //双精度浮点型构造函数
        timer(double h) {
            minutes = (int) 60*h ;
        };
        long getminutes(void) { return minutes ; };
};
//main()函数的定义
main(void)
{
        //使用 double 类型的构造函数创建对象
        timer start(8.30),finish(17.30);
        cout<<"finish(17.30)-start(8.30)=";
        cout<<finish.getminutes()-start.getminutes()<<endl;

        //使用 char 指针类型的构造函数创建对象
        timer start0("500"),finish0("800");     //创建对象
        cout<<"finish0(\"800\")-start0(\"500\")=";
        cout<<finish0.getminutes()-start0.getminutes()<<endl;

        //使用无参数构造函数和整型构造函数创建对象
        timer start1;
        timer finish1(3,30);
        cout<<"finish1(3,30)-start1=";
        cout<<finish1.getminutes()-start1.getminutes()<<endl;

        return 0;
}
#include <iostream.h>
//定义 rect 类
class rect {
        int length;
        int width;
        int area;
public:
        rect(int l=1,int w=1)
        {
            length=l;
            width=w;
            area=length*width;
        }
        void show_rect(char *name)
        {
            cout<<name<<":"<<endl;
            cout<<"length="<<length<<endl;
            cout<<"width="<<width<<endl;
            cout<<"area="<<area<<endl;
        }
};
//测试使用 rect 类
void main(void)
{
        //用 rect 类创建对象
        rect a;
```

```
    rect b(2);
    rect c(2,3);

    //调用对象的函数显示对象中的数据
    a.show_rect("a");
    b.show_rect("b(2)");
    c.show_rect("c(2,3)");
}
#include<iostream.h>
const int MAX=5;        //假定栈中最多保存 5 个数据

//定义名为 stack 的具有栈功能的类
class   stack {
    //数据成员
    double   num[MAX];    //存放栈数据的数组
    int   top;                    //指示栈顶位置的变量
public:
    //成员函数
    stack(char *name)   //构造函数
    {
        top=0;
        cout<<"Stack "<<name<<" initialized."<<endl;
    }
    ~stack(void)    //析构函数
    {
        cout << "Stack destroyed." << endl;   //显示信息
    }

    void push(double x)      //入栈函数
    {
        if (top==MAX){
            cout<<"Stack is full !"<<endl;
            return;
        };
        num[top]=x;
        top++;
    }
    double pop(void)   //出栈函数
    {
        top--;
        if (top<0){
        cout<<"Stack is underflow !"<<endl;
        return 0;
        };
        return num[top];
    }
}

//以下是 main()函数，其用 stack 类创建栈对象，并使用了这些对象
main(void)
{
    double x;
    //声明(创建)栈对象并初始化
    stack a("a"),b("b");

    //以下利用循环和 push()成员函数将 2,4,6,8,10 依次入 a 栈
    for (x=1; x<=MAX; x++)
        a.push(2.0*x);

    //以下利用循环和 pop()成员函数依次弹出 a 栈中的数据并显示
```

```
        cout<<"a: ";
        for (int i=1; i<=MAX; i++)
            cout<<a.pop()<<"    ";
        cout<<endl;

        //从键盘上为 b 栈输入数据,并显示
        for(i=1;i<=MAX;i++) {

              cout<<i<<" b:";
              cin>>x;
              b.push(x);
        }
        cout<<"b: ";
        for(i=1;i<=MAX;i++)
            cout<<b.pop()<<" ";
        cout<<endl;
}
#include<iostream.h>
#define MAX 5
//定义 stack 类接口
class stack{
        int num[MAX];
        int top;
public:
         stack(char *name);      //构造函数原型
        ~stack(void);                //析构函数原型
        void push(int n);
        int pop(void);
};
//main()函数测试 stack 类
main(void)
{
        int i,n;
        //声明对象
        stack a("a"),b("b");

        //以下利用循环和 push()成员函数将 2,4,6,8,10 依次入 a 栈
        for (i=1; i<=MAX; i++)
              a.push(2*i);

        //以下利用循环和 pop()成员函数依次弹出 a 栈中的数据，并显示
        cout<<"a: ";
        for (i=1; i<=MAX; i++)
            cout<<a.pop()<<"    ";
        cout<<endl;

        //从键盘上为 b 栈输入数据,并显示
        for(i=1;i<=MAX;i++) {
              cout<<i<<" b:";
              cin>>n;
              b.push(n);
        }
        cout<<"b: ";
        for(i=1;i<=MAX;i++)
            cout<<b.pop()<<" ";
        cout<<endl;

        return 0;
}
//-------------------------
```

```
//     stack 成员函数的定义
//-------------------------
//定义构造函数
stack::stack(char *name)
{
      top=0;
      cout << "Stack "<<name<<" initialized." << endl;
}
//定义析构函数
stack::~stack(void)
{
      cout << "stack destroyed." << endl;    //显示信息
}
//入栈成员函数
void stack::push(int n)
{
      if (top==MAX){
            cout<<"Stack is full !"<<endl;
            return;
      };
      num[top]=n;
      top++;
}
//出栈成员函数
int stack::pop(void)
{
      top--;
      if (top<0){
            cout<<"Stack is underflow !"<<endl;
            return 0;
      };
      return num[top];
}
#include<iostream.h>
//定义一个全部为 public:模式的类
class ex
{
public:
        int value;
        void set(int n) {
              value=n;
        }
        int get(void) {
            return value;
        }
};
//测试使用 ex 类
main()
{
      ex a;       //创建对象

      //以下通过成员函数访问对象数据
      a.set(100);
      cout<<"a.get()=";
      cout<<a.get()<<endl;

      //以下直接访问对象的数据成员
      a.value=200;
      cout<<"a.value=";
      cout<<a.value<<endl;
```

```
}
#include <iostream.h>
// ex_class 类接口定义
class ex_class
{
private:
	int iv;
	double dv;
public:
	ex_class(void);
	ex_class(int n,double x);
	void set_ex_class(int n,double x);
	void show_ex_class(char*);
};

//定义 ex_class 类的构造函数
ex_class::ex_class(void):iv(1), dv(1.0) { }
ex_class::ex_class(int n,double x):iv(n), dv(x) { }

//定义 ex_class 类的成员函数
void ex_class::set_ex_class(int n,double x)
{
	iv=n;
	dv=x;
}
void ex_class::show_ex_class(char *name)
{
	cout<<name<<": "<<endl;
	cout <<"iv=" <<iv<< endl;
	cout <<"dv=" <<dv<< endl;
}
//使用 ex_class 类
void main(void)
{
	ex_class obj1;
	obj1.show_ex_class("obj1");
	obj1.set_ex_class(5,5.5);
	obj1.show_ex_class("obj1");

	ex_class obj2(100,3.14);
	obj2.show_ex_class("obj2");
	obj2.set_ex_class(2000,1.732);
	obj2.show_ex_class("obj2");
}
#include<iostream.h>
//定义一个含有 static 数据成员的类
class ex
{
	static int num;		//static 数据成员
public:
	ex() {num++;}
	~ex() {num--;}
	disp_count() {
	    cout<<"The current instances count:";
	    cout<<num<<endl;
	}
};
int ex::num=0;	//设置 static 数据成员的初值
//main()函数测试 ex 类
main()
```

```
{
	ex a;
	a.disp_count();

	ex *p;
	p=new ex;
	p->disp_count();

	ex x[10];
	x[0].disp_count();

	delete p;
	a.disp_count();
}
#include<iostream.h>
//定义一个含有 static 数据成员的类
class ex
{
	static int num;		//static 数据成员
public:
	ex() {num++;}
	~ex() {num--;}
	static disp_count(void) //static 成员函数
	{
		cout<<"The current instances count:";
		cout<<num<<endl;
	}
};
int ex::num=0;	//设置 static 数据成员的初值
//main()函数测试 ex 类
main()
{
	ex a;
	a.disp_count();

	ex *p;
	p=new ex;
	p->disp_count();

	ex x[10];
	ex::disp_count();	//直接用类作用域符引用静态成员函数

	delete p;
	ex::disp_count();	//直接用类作用域符引用静态成员函数
}
#include <iostream.h>
class ex_class {
	int value;
public:
	ex_class(int n) {
		value=n;
		cout << "Stack initialized." << endl;
	}
	~ex_class() {
		cout << "The Object destroyed." <<endl;
	}
	void set_value(int n);
	void show_val(char *name);
} ;
```

```
//在类外定义内联成员函数
inline void ex_class::set_value(int n) {
	value=n;
}
//在类外定义非内联成员函数
void ex_class::show_val(char *name) {
	cout<<name<<": ";
	cout<<value<<endl;
}
//在 main()函数中测试 ex_class 类
main(void)
{
	//创建对象 x 和 y
	ex_class x(100),y(200);

	//显示对象的数据
	x.show_val("x");
	y.show_val("y");

	//设置新值给对象
	x.set_value(1);
	y.set_value(2);

	//显示对象的数据
	x.show_val("x");
	y.show_val("y");

	return 0;
}
#include <iostream.h>
//定义空类 empty
class empty
{
};
//在 main()函数中用空类创建对象
main()
{
	empty a,*p;   //编译通过
	cout<<"Test a empty class."<<endl;
}
#include<iostream.h>
//用 struct 关键字定义 ex_class 类
struct ex_class {
	ex_class(int n=1): value(n) {}
	void set_value(int n) {
		value=n;
	}
	show_obj(char *name) {
		cout<<name<<": "<<value<<endl;
	}
private:
	int value;
}
//测试  ex_class 类
main()
{
	//用 ex_class 创建对象
	ex_class a,b(3);

	a.show_obj("a");
```

```
	b.show_obj("b");

	a.set_value(100);
	b.set_value(200);

	a.show_obj("a");
	b.show_obj("b");
}
#include <iostream.h>
#include<string.h>
//定义双亲（parent）类
class parent {
	char	f_name[20];
	char	m_name[20];
	char	tel[10];
public:
	// parent 类的构造函数，其带有缺省值
	parent(char *p1="",char *p2="",char *p3="") {
		strcpy(f_name,p1);
		strcpy(m_name,p2);
		strcpy(tel,p3);
	}
	//显示 parent 对象的数据
	show_parent(void) {
		cout<<"The parent:"<<endl;
		cout<<"		father's name:"<<f_name<<endl;
		cout<<"		mother's name:"<<m_name<<endl;
		cout<<"		tel:"<<tel<<endl;
	}
};
//定义 student 类
class student {
	int		num;
	char		name[20];
	float		grade;
	parent		pt;
public:
	// student 类的构造函数
	student(int n,char *str,float g,class parent t) {
		num=n;
		strcpy(name,str);
		grade=g;
		pt=t;
	}
	//显示 student 对象的数据
	show_student(void) {
		cout<<"num:"<<num<<endl;
		cout<<"name:"<<name<<endl;
		cout<<"grade:"<<grade<<endl;
		pt.show_parent();
	}
};
//main()函数测试 student 类的对象
main(void)
{
	//创建双亲对象
	parent p1("ZhangHua","LiLan","83665215");

	//创建学生对象
	student st(10001,"ZhangHui",91.5,p1);
```

```
        //显示学生信息
        cout<<"p1:"<<endl;
        p1.show_parent();

        //显示学生信息
        cout<<"st:"<<endl;
        st.show_student();
}
#include <iostream.h>
#include <stdlib.h>
//定义 timer 类
class timer{
        long minutes;
public:
        //定义重载成员函数
        settimer(char *m) {
                minutes = atoi(m);
        };
        //定义重载成员函数
        settimer(int h, int m) {
                minutes = 60*h+m ;
        };
        //定义重载成员函数
        settimer(double h) {
                minutes = (int) 60*h ;
        };
        long getminutes(void) { return minutes; };
};
//main()函数的定义
main(void){
        timer start,finish;     //创建对象

        //使用重载成员函数
        start.settimer(8,30);
        finish.settimer(9,40);
        cout<<"finish.settimer(9,40)-start.settimer(8,30):";
        cout<<finish.getminutes()-start.getminutes()<<endl;

        //使用重载成员函数
        start.settimer(2.0);
        finish.settimer("180");
        cout<<"finish.settimer(\"180\")-start.settimer(2.0):";
        cout<<finish.getminutes()-start.getminutes()<<endl;

    return 0;
}
#include <iostream.h>
//定义复数类
class complex{
        float    real;              //实部
        float    image;             //虚部
public:
        //重载的运算符"+"的原型
        complex operator+ (complex right);
        //重载赋值运算符"="的定义
        complex operator= (complex right);
        void set_complex(float re, float im);
        void put_complex(char *name);
};
```

```
//重载加法运算符"+"的定义
complex complex::operator+ (complex right) {
	complex temp;
	temp.real = this->real + right.real;
	temp.image = this->image + right.image;
	return temp;
}
//重载加赋值运算符"="的定义
complex complex::operator= (complex right) {
		this->real = right.real;
		this->image = right.image;
		return *this;
}
//定义 set_complex()成员函数
void complex::set_complex(float re, float im) {
		real = re;
		image = im;
}
//定义 put_complex()成员函数
void complex::put_complex(char *name) {
		cout<<name<<": ";
		cout << real << ' ';
		if (image >= 0.0 ) cout << '+';
		cout << image << "i\n";
}
//在 main()函数中使用 complex 类的对象
main(void)
{
	complex A, B, C;   //创建复数对象

	//设置复数变量的值
	A.set_complex(1.2, 0.3);
	B.set_complex(-0.5, -0.8);

	//显示复数数据
	A.put_complex("A");
	B.put_complex("B");

	//赋值运算，显示结果
	C = A;
	C.put_complex("C=A");

	//加法及赋值运算，显示结果
	C = A + B;
	C.put_complex("C=A+B");
	return 0;
}
// Example of the friend class
#include <iostream.h>
//定义 YourClass 类，
class YourClass
{
//指定 YourOtherClass 是它的友元类
friend class YourOtherClass;
private:
	int num;
public:
	YourClass(int n){num=n;}
	display(char *YCname){
		cout<<YCname<<".num :";
```

```
            cout<<num<<endl;
        }
};
//定义 YourOtherClass，它是 YourClass 类的友元类
class YourOtherClass
{
public:
        //使用 YourClass 类的私有成员
        void disp1(YourClass yc,char *YCname){
                cout<<YCname<<".num :";
            cout<<yc.num<<endl;
        }
        //使用 YourClass 类的公共成员
        void disp2(YourClass yc,char* YCname){
            yc.display(YCname);
        }
};
//在 main()函数中创建和使用 YourClass 和 YourOtherClass 类对象
main(void)
{
        //声明 YourClass 类对象
        YourClass a(10),b(100);

        //显示 a 和 b 对象的值
        cout<<"YourClass:"<<endl;
        a.display("a");
        b.display("b");

        //声明 YourOtherClass 类对象
        YourOtherClass temp;

        //通过 temp 显示 a 和 b 对象的值
        cout<<"YourOtherClass:"<<endl;
        temp.disp1(a,"a");
        temp.disp2(b,"b");
}
#include<iostream.h>
//Y 类的不完全定义
class Y;

//X 类的定义
class X {
public:
        void disp(Y py,char *name);      //成员函数原型
};

//定义 Y 类
class Y {
        //声明本类的友元函数
        //X 类的 disp()为本例的友元函数
        friend void X::disp(Y py,char *name);
          //普通函数 putY()  为本例的友元函数
        friend void putY(Y& yc,char *name);
private: //私有成员
        int num;
        dispY(char *name){
            cout<<name<<".num="<<num<<endl;
        }
public: //公共成员函数
        Y(int n){
```

```
        num=n;
    }
};

//X 类成员函数的实现部分
void X::disp(Y py,char *name){
    cout<<"In X::disp():"<<endl;
    py.dispY(name);     //访问 Y 类的私有函数
}

//普通函数 putY()的定义
void putY(Y& yc,char *name){
    cout<<"In getY:"<<endl;
    yc.dispY(name);
    cout<<name<<".num=";
    cout<<yc.num<<endl;
}

//在 main()函数测试 X 和 Y 类的功能
main()
{
    //创建 Y 和 X 类的对象
    Y y1(100),y2(200);
    X x;

    //不可用 Y 类对象的私有成员函数显示
    //y1.dispY("y1");
    //y2.dispY("y2");

    //调用 X 类对象的友元函数显示
    x.disp(y1,"y1");
    x.disp(y2,"y2");

    //用 getY 函数显示 Y 类的对象显示
    putY(y1,"y1");
    putY(y2,"y2");
}
#include <iostream.h>
//定义日期类
class Date
{
    //定义友元重载输入运算符函数
    friend istream& operator >> (istream& input,Date& dt );
    //定义友元重载输出运算符函数
    friend ostream& operator<< (ostream& output,Date& dt );
    int mo, da, yr;
public:
    Date(void){    //无参数构造函数
        yr = 0;
        mo = 0;
        da = 0;
    }
    Date( int y, int m, int d )     //带参数构造函数
    {
        yr = y;
        mo = m;
        da = d;
    }
};
//定义">>"运算符重载函数
```

```
istream& operator >> ( istream& input, Date& dt )
{
	cout<<"Year:";
	input>>dt.yr;
	cout<<"Month:";
	input>>dt.mo;
	cout<<"Day:";
	input>>dt.da;
	return input;
}

//定义"<<"运算符重载函数
ostream& operator<< ( ostream& output, Date& dt )
{
    output<< dt.yr << '/' << dt.mo << '/' << dt.da<<endl;
    return output;
}

//在 main()函数中测试 Date 类的插入（<<）和提取（>>）运算符
void main()
{
	//声明对象
	Date dt1(2002,5,1),dt2;

	//显示 dt1 对象
	cout<<dt1;

	//对 dt2 对象进行输入和输出
	cin>>dt2;
	cout<<dt2;
}
#include<iostream.h>
//定义 ex 类
class ex_class
{
	int a;
	double b;
public:
	ex_class(int n=1,double x=1.0):a(n),b(x) {}
	void show_value(char *name) {
		cout<<name<<" :"<<endl;
		cout<<"a="<<a<<endl;
		cout<<"b="<<b<<endl;
	}
};

//定义 main()函数
main()
{
	//创建 ex_class 的对象并显示
	ex_class obj1,obj2(100,3.5);
	obj1.show_value("obj1");
	obj2.show_value("obj2");

	//创建 ex_class 的指针变量
	ex_class *p;

	//p 指向 obj1 并显示
	p=&obj1;
	p->show_value("p->obj1");
```

```
	//p 指向 obj2 并显示
	p=&obj2;
	(*p).show_value("(*p)obj2");

	//p 指向动态创建的对象并显示
	p=new ex_class;
	p->show_value("p->new");

	delete p;	//删除对象

}
#include<iostream.h>
//基类 Box
class Box {
	int width,height;
public:
	void SetWidth(int w) {
		width=w;
	}
	void SetHeight(int h) {
		height=h;
	}
	int GetWidth() {return width;}
	int GetHeight() {return height;}
};
//派生类 ColoredBox
class ColoredBox:public Box
{
	int color;
public:
	void SetColor(int c){
		color=c;
	}
	int GetColor() {return color;}
};
//  在 main()中测试基类和派生类
main(void)
{
	//声明并使用 ColoredBox 类的对象
	ColoredBox cbox;
	cbox.SetColor(3);		//使用自己的成员函数
	cbox.SetWidth(150);		//使用基类的成员函数
	cbox.SetHeight(100);		//使用基类的成员函数

	cout<<"cbox:"<<endl;
	cout<<"Color:"<<cbox.GetColor()<<endl;		//使用自己的成员函数
	cout<<"Width:"<<cbox.GetWidth()<<endl;		//使用基类的成员函数
	cout<<"Height:"<<cbox.GetHeight()<<endl;	//使用基类的成员函数
	//cout<<cbox.width; Error!
}
#include<iostream.h>
//基类 First
class First {
	int val1;
public:
	SetVal1(int v) {
		val1=v;
	}
	void show_First(void) {
```

```
            cout<<"val1="<<val1<<endl;
        }
};
//派生类 Second
class Second:private First {     //默认为 private 模式
        int val2;
public:
        void SetVal2(int v1,int v2) {
            SetVal1(v1);        //可见，合法
            val2=v2;
        }
        void show_Second(void) {
        // cout<<"val1="<<val1<<endl;  不能访问 First 私有成员
            show_First();
            cout<<"val2="<<val2<<endl;
        }
};
main() {
        Second s1;
        //s1.SetVal1(1);       //不可见，非法
        s1.SetVal2(2,3);       //合法
        //s1.show_First();    //不可见，非法
        s1.show_Second();
}
#include<iostream.h>
//基类 First
class First {
        int val1;
public:
        SetVal1(int v) {
            val1=v;
        }
        void show_First(void) {
            cout<<"val1="<<val1<<endl;
        }
};
//派生类 Second
class Second:public First {     //默认为 private 模式
        int val2;
public:
        void SetVal2(int v1,int v2) {
            SetVal1(v1);        //可见，合法
            val2=v2;
        }
        void show_Second(void) {
        // cout<<"val1="<<val1<<endl;  不能访问 First 私有成员
            show_First();
            cout<<"val2="<<val2<<endl;
        }
};
main() {
        Second s1;
        //调用 Second 类定义的成员函数
        s1.SetVal2(2,3);
        cout<<"s1.show_Second():"<<endl;
        s1.show_Second();

        //调用 First 类定义的成员函数
        s1.SetVal1(10);
        cout<<"s1.show_First():"<<endl;
```

```
	s1.show_First();
}
#include<iostream.h>
//定义最低层基类，它作为其他类的基类
class First {
	int val1;
public:
	First(void) {
		cout<<"The First initialized"<<endl;
	}
};
//定义派生类，它作为其他类的基类
class Second :public First {
	int val2;
public:
	Second(void) {
		cout<<"The Second initialized"<<endl;
	}
};
//定义最上层派生类
class Three :public Second {
	int val3;
public:
	Three() {
		cout<<"The Three initialized"<<endl;
	}
};
//定义各基类的对象，测试构造函数的执行情况
//定义各基类的对象，测试构造函数的执行情况
main() {
	cout<<"First f1;"<<endl;
	First f1;
	cout<<"Second s1;"<<endl;
	Second s1;
	cout<<"Three t1;"<<endl;
	Three t1;
}
#include<iostream.h>
//定义基类 First
class First {
	int    num;
	float grade;
public:
	//构造函数带参数
	First(int n,float v ) : num(n),grade(v)
	{
		cout<<"The First initialized"<<endl;
	}
	DispFirst(void) {
		cout<<"num="<<num<<endl;
		cout<<"grade="<<grade<<endl;
	}
};

//定义派生类 Second
class Second :public First {
	double val;
public:
	//无参数构造函数，要为基类的构造函数设置参数
	Second(void):First(10000,0) {
```

```
        val=1.0;
        cout<<"The Second initialized"<<endl;
    }

    //带参数构造函数，为基类的构造函数设置参数
    Second(int n,float x,double dx):First(n,x) {
        val=dx;
        cout<<"The Second initialized"<<endl;
    }
    Disp(char *name){
        cout<<name<<".val="<<val<<endl;
        DispFirst();
    }
};

//main()函数中创建和使用派生类对象
main() {
    //调用派生类的无参数构造函数
    cout<<"Second s1;"<<endl;
    Second s1;
    cout<<"s1.Disp(\"s1\");"<<endl;
    s1.Disp("s1");

    //调用派生类的有参数构造函数
    cout<<"Second s2(10002,95.7,3.1415926); "<<endl;
    Second s2(10002,95.7,3.1415926);
    cout<<"s2.Disp(\"s2\");"<<endl;
    s2.Disp("s2");
}
#include<iostream.h>
//定义最低层基类 First，它作为其他类的基类
class First {
    int val1;
public:
    First() {
        cout<<"The First initialized"<<endl;
    }
    ~First() {
        cout<<"The First destroyed"<<endl;
    }
};
//定义派生类 Second，它作为其他类的基类
class Second :public First {     //默认为 private 模式
    int val2;
public:
    Second() {
        cout<<"The Second initialized"<<endl;
    }
    ~Second() {
        cout<<"The Second destroyed"<<endl;
    }
};
//定义最上层派生类 Three
class Three :public Second {
    int val3;
public:
    Three() {
        cout<<"The Three initialized"<<endl;
    }
    ~Three() {
```

```
        cout<<"The Three destroyed"<<endl;
    }
};
//main()函数中测试构造函数和析构函数的执行情况
main() {
    Three t1;
    cout<<"---- Use the t1----"<<endl;
}
#include<iostream.h>
//基类
class First {
    int val1;
protected:
    void SetVal1(int v) {
        val1=v;
    }
public:
    show_First(void) {
        cout<<"val1="<<val1<<endl;
    }
};
//派生类
class Second:public First {
    int val2;
protected:
    void SetVal2(int v) {
        SetVal1(v);   //使用 First  基类的保护成员
        val2=v;
    }
public:
    show_Second(void) {
        show_First();
        cout<<"val2="<<val2<<endl;
    }
};
//派生类
class Third:public Second {
    int val3;
public:
    void SetVal3(int n) {
         SetVal1(n);   //使用 First  基类的保护成员
         SetVal2(n);   //使用 Second 基类的保护成员
         val3=n;
    }
    show_Third(void) {
        show_Second();
        cout<<"val3="<<val3<<endl;
    }
};
//main()函数的定义
main(void)
{
    First f1;
    //f1.SetVal1(1);     不可访问

    Second s1;
    //s1.SetVal1(1);     不可访问
    //s1.SetVal2(2);     不可访问

    Third   t1;
```

```
    //t1.SetVal1(1);     不可访问
    //t1.SetVal2(2);     不可访问
    t1.SetVal3(10);

    //显示 t1 对象的数据
    cout<<"t1.show_Third();"<<endl;
    t1.show_Third();
    cout<<"t1.show_Second();"<<endl;
    t1.show_Second();
    cout<<"t1.show_First();"<<endl;
    t1.show_First();
}
#include <iostream.h>
enum Color {Red,Yellow,Green,White};
//圆类 Circle 的定义
class Circle {
    float radius;
public:
    Circle(float r) {radius=r;}
    float Area() {
        return 3.1416*radius*radius;
    }
};
//桌子类 Table 的定义
class Table {
    float height;
public:
    Table(float h) {height=h;}
    float Height() {
        return height;
    }
};
//圆桌类 RoundTable 的定义
class RoundTable:public Table,public Circle {
    Color color;
public:
    RoundTable(float h,float r,Color c); //构造函数
    int GetColor() {
        return color;
    }
};
//圆桌构造函数的定义
RoundTable::RoundTable(float h,float r,Color c):Table(h),Circle(r)
{
    color=c;
}
//main()函数的定义
main() {
    RoundTable cir_table(15.0,2.0,Yellow);

    cout<<"The table properties are:"<<endl;
    //调用 Height 类的成员函数
    cout<<"Height="<<cir_table.Height()<<endl;

    //调用 circle 类的成员函数
    cout<<"Area="<<cir_table.Area()<<endl;

    //调用 RoundTable 类的成员函数
    cout<<"Color="<<cir_table.GetColor()<<endl;
}
```

```
#include <iostream.h>
//定义一个枚举类型
enum Color {Red,Yellow,Green,White};
//圆类 Circle 的定义
class Circle {
	float radius;
public:
	Circle(float r) {
		radius=r;
		cout<<"Circle initialized!"<<endl;
	}
	~Circle() {    //析构函数
		  cout<<"Circle    destroyed!"<<endl;
	}
	float Area() {
		return 3.1416*radius*radius;
	}
};
//桌子类 Table 的定义
class Table {
	float height;
public:
	Table(float h) {
		height=h;
		cout<<"Table initialized!"<<endl;
	}
	~Table() {    //构造函数
		cout<<"Table destroyed!"<<endl;
	}
	float Height() {
		return height;
	}
};
//圆桌类 RoundTable 的定义
class RoundTable:public Table,public Circle {
	Color color;
public:
	RoundTable(float h,float r,Color c); //构造函数
	int GetColor() {
	    return color;
	}
	~RoundTable() {    //构造函数
		cout<<"RoundTable destroyed!"<<endl;
	}
};
//圆桌构造函数的定义
RoundTable::RoundTable(float h,float r,Color c):Table(h),Circle(r)
{
	color=c;
	cout<<"RoundTable initialized!"<<endl;
}
//测试多继承中构造函数和析构函数的执行方式
main() {
	RoundTable cir_table(15.0,2.0,Yellow);

	cout<<"The table properties are:"<<endl;
	//调用 Height 类的成员函数
	cout<<"Height="<<cir_table.Height()<<endl;

	//调用 circle 类的成员函数
```

```
    cout<<"Area="<<cir_table.Area()<<endl;

    //调用 RoundTable 类的成员函数
    cout<<"Color="<<cir_table.GetColor()<<endl;
}
#include<iostream.h>
//定义有两个虚函数的基类
class Base {
public:
    //定义两个虚函数
    virtual void aFn1(void){
        cout<<"aFnl is in Base class."<<endl;
    }
    virtual void aFn2(void) {
        cout<<"aFn2 is in Base class."<<endl;
    }
    //定义非虚函数
    void aFn3(void) {
        cout<<"aFn3 is in Base class."<<endl;
    }
};

//派生类 Derived_1 中重新定义了基类中的虚函数 aFn1
class Derived_1:public Base
{
public:
    void aFn1(void) {     //覆盖 aFn1()函数
        cout<<"aFnl is in First derived class."<<endl;
    }
    // void aFn3(void) {     语法错误
    //     cout<<"aFn3 is in First derived class."<<endl;
    //}
};

//派生类 Derived_2 中重新定义了基类中的虚函数 aFn2
class Derived_2:public Base{
public:
    void aFn2(void){     //覆盖 aFn2()函数
        cout<<"aFn2 is in Second derived class."<<endl;
    }
    // void aFn3(void) {     语法错误
    //     cout<<"aFn3 is in Second derived class."<<endl;
    //}
};
//main()函数的定义
main(void)
{
    //创建和使用基类 Base 的对象
    Base b;
    cout<<"Base:"<<endl;
    b.aFn1();
    b.aFn2();
    b.aFn3();
    cout<<"----------------------"<<endl;

    //创建和使用派生类 Derived_1 的对象
    Derived_1 d1;
    cout<<"Derived_1:"<<endl;
    d1.aFn1();
    d1.aFn2();
```

```
    d1.aFn3();
    cout<<"----------------------"<<endl;

    //创建和使用派生类 Derived_2 的对象
    Derived_2 d2;
    cout<<"Derived_2:"<<endl;
    d2.aFn1();
    d2.aFn2();
    d2.aFn3();
}
#include<iostream.h>
//定义抽象类
class Base {
public:
    //定义两个纯虚函数
    virtual void aFn1(void)=0;
    virtual void aFn2(void)=0;
};

//派生类 Derived_1 中覆盖了基类中的纯虚函数
class Derived_1:public Base
{
public:
    void aFn1(void) {
        cout<<"aFnl is in First derived class."<<endl;
    }
    void aFn2(void) {
        cout<<"aFn2 is in First derived class."<<endl;
    }
};

//派生类 Derived_2 中覆盖了基类中的纯虚函数
class Derived_2:public Base{
public:
    virtual void aFn1(void){
        cout<<"aFn1 is in Second derived class."<<endl;
    }
    void aFn2(void){
        cout<<"aFn2 is in Second derived class."<<endl;
    }
};

//main()函数中测试抽象类及其派生类的对象
main(void)
{
    //用抽象类不能创建对象
    //    Base b;   语法错误
    //    b.aFn1();
    //    b.aFn2();

    //创建和使用 Derived_1 类的对象
    Derived_1 d1;
    cout<<"Derived_1 d1:"<<endl;
    d1.aFn1();
    d1.aFn2();
    cout<<"------------------"<<endl;

    //创建和使用 Derived_2 类的对象
    Derived_2 d2;
    cout<<"Derived_2 d2:"<<endl;
```

```
    d2.aFn1();
    d2.aFn2();
}
#include<iostream.h>
int extract_int()
{
    char ch;
    int n=0;
    while(ch=cin.get())
        if (ch>='0' && ch<='9')
        {
            cin.putback(ch);
            cin>>n;
            break;
        }
    return n;
}
//main()函数
main(void)
{
    //提取字符串中的数字
    int a=extract_int();
    int b=extract_int();
    int c=extract_int();

    //显示结果
    cout<<a<<"+"<<b<<"="<<c<<endl;
}
#include<iostream.h>
//定义节点（数据对象）的接口
class Node
{
    //声明 list 类为本类的友元类
    friend class list;
//私有成员
private:
    int Data;           //节点数据
    Node *previous; //前趋指针
    Node *next;         //后继指针
};

//定义双向链表 list 的接口声明
class list
{
//私有成员
private:
    Node *Head;       //链表头指针
    Node *Tail;       //链表尾指针
//定义接口函数
public:
    //构造函数
    list();
    //析构函数
    ~list();
    //从链表尾后添加数据
    void Build_HT(int Data);
    //从链表前头添加数据
    void Build_TH(int Data);
    //从头到尾显示数据
    void list::Display_HT();
```

```
	//从尾到头显示数据
	void list::Display_TH();
	//清除链表的全部数据
	void Clear();
};

//main()函数测试双向链表
int main(void)
{
	list list1;
	int i;

	//从尾添加数据
	cout<<"Add to the back of the list1:"<<endl;
	for (i=1;i<=20;i=i+2) {
		list1.Build_HT(i);
		cout<<i<<" ";
	}
	cout<<endl;

	//从头添加数据
	cout<<"Add to the front of the list1:"<<endl;
	for (i=0;i<=20;i=i+2) {
		list1.Build_TH(i);
		cout<<i<<" ";
	}
	cout<<endl;

	//显示链表
	list1.Display_HT();
	list1.Display_TH();

	return 0;
}
//list 类函数的定义
//构造函数的定义
list::list()
{
	 //初值
	 Head=0;
	 Tail=0;
}
//析构函数的定义
list::~list()
{
	Clear();
}
//从链表尾后添加数据
void list::Build_HT(int Data)
{
	Node *Buffer;

	Buffer=new Node;
	Buffer->Data=Data;
	if(Head==0)
	{
		Head=Buffer;
		Head->next=0;
		Head->previous=0;
	Tail=Head;
```

```
        }
        else
        {
                Tail->next=Buffer;
                Buffer->previous=Tail;
        Buffer->next=0;
                Tail=Buffer;
        }
}
//从链表前头添加数据
void list::Build_TH(int Data)
{
        Node *NewNode;
        NewNode=new Node;
        NewNode->Data=Data;

        if(Tail==0)
        {
                Tail=NewNode;
        Tail->next=0;
                Tail->previous=0;
                Head=Tail;
        }
        else
        {
                NewNode->previous=0;
                NewNode->next=Head;
                Head->previous=NewNode;
                Head=NewNode;
        }
}
//从头到尾显示数据
void list::Display_HT()
{
        Node *TEMP;
        TEMP=Head;
        cout<<"Display the list from Head to Tail:"<<endl;
        while(TEMP!=0)
        {
                cout<<TEMP->Data<<" ";
                TEMP=TEMP->next;
        }
        cout<<endl;
}
//从尾到头显示数据
void list::Display_TH()
{
        Node *TEMP;
        TEMP=Tail;
        cout<<"Display the list from Tail to Head:"<<endl;
        while(TEMP!=0)
        {
                cout<<TEMP->Data<<" ";
                TEMP=TEMP->previous;
        }
        cout<<endl;
}
//清除链表的全部数据
void list::Clear()
{
```

```
	Node *Temp_head=Head;

	if (Temp_head==0) return;
	do
	{
		Node *TEMP_NODE=Temp_head;
		Temp_head=Temp_head->next;
		delete TEMP_NODE;
	}
	while (Temp_head!=0);
}
#include <iostream>
#include <string>

using namespace std;

//测试字符串(string)对象
void main()
{
	//创建 string 对象,并显示
	string s1;
	string s2="ABCDEFGHIJK";
	string s3=s2;
	string s4(20,'A');
	string s5(s2,3,3);
	cout<<"s1="<<s1<<endl;
	cout<<"s2="<<s2<<endl;
	cout<<"s3="<<s3<<endl;
	cout<<"s4="<<s4<<endl;
	cout<<"s5="<<s5<<endl;

	//为 string 对象输入数据,并显示
	cout<<"s1=";
	cin>>s1;
	cout<<"s2=";
	cin>>s2;
	cout<<"s3=";
	cin>>s3;
	cout<<"s4=";
	cin>>s4;
	cout<<"s5=";
	cin>>s5;

	cout<<"s1="<<s1<<endl;
	cout<<"s2="<<s2<<endl;
	cout<<"s3="<<s3<<endl;
	cout<<"s4="<<s4<<endl;
	cout<<"s5="<<s5<<endl;
}
#include <iostream>
#include <string>

using namespace std;

//测试字符串(string)对象
void main()
{

	//创建 string 对象
	string s1,s2;
```

```
	//string 对象的赋值运算
	s1="One";
	s2="Two";
	cout<<"s1="<<s1<<endl;
	cout<<"s2="<<s2<<endl;

	//string 对象的连接运算
	string s3;
	s3=s1+" and "+s2;
	cout<<"s3="<<s3<<endl;

	//组合赋值连接运算
	s3+=" and Three";
	cout<<"s3="<<s3<<endl;

	//比较运算及其结果显示
	for (int i=1;i<=3;i++) {
		cout<<"----------------------"<<endl;
		cout<<"s1=";
		cin>>s1;
		cout<<"s2=";
		cin>>s2;
		if (s1<s2)     //小于
			cout<<s1<<" < "<<s2<<endl;
		if (s1<=s2)    //小于等于
		cout<<s1<<" <= "<<s2<<endl;
		if (s1==s2)    //等于
		cout<<s1<<" == "<<s2<<endl;
		if (s1>s2)     //大于
		cout<<s1<<" > "<<s2<<endl;
		if (s1>=s2)    //大于等于
		cout<<s1<<" >= "<<s2<<endl;
		if (s1!=s2)    //不等
			cout<<s1<<" != "<<s2<<endl;
	}
}
#include <iostream>
#include <string>

using namespace std;

//测试字符串(string)对象
void main()
{
	//创建 string 对象,并显示
	string s1="This";
	string s2="book.";
	cout<<"s1: "<<s1<<endl;
	cout<<"s2: "<<s2<<endl;

	//使用 length 成员函数
	cout<<"s1.length()="<<s1.length()<<endl;
	cout<<"s2.length()="<<s2.length()<<endl;

	//使用 append 成员函数
	s1.append(s2);
	cout<<"s1: "<<s1<<endl;

	//使用 find 成员函数和下标运算
```

```
    int pos=s1.find('b');
    cout<<"s1["<<pos<<"]="<<s1[pos]<<endl;

    //使用 insert 成员函数
    s1.insert(pos," is a ");
    cout<<s1<<endl;

    //使用 assign 成员函数
    s1.assign("Good");
    cout<<s1<<endl;

}
//根据半径计算圆的周长和面积
#include <iostream.h>
const float PI=3.1416;          //声明常量(只读变量)PI 为 3.1416
float fCir_L(float);            //声明自定义函数 fCir_L()的原型
float fCir_S(float);            //声明自定义函数 fCir_S()的原型

//以下是 main()函数
main()
{
    float r,l,s;                //声明 3 个变量

    cout<<"R=";                 //显示字符串
    cin>>r;                     //键盘输入
    l=fCir_L(r);                //计算圆的周长，赋值给变量 l
    s=fCir_S(r);                //计算圆的面积，赋值给变量 s
    cout<<"l="<<l;              //显示计算结果
    cout<<"\ns="<<s;
}

//定义计算圆的周长的函数 fCir_L()
float fCir_L(float x)
{
    float z=-1.0;               //声明局部变量
    if (x>=0.0)                 //如果参数大于 0，则计算圆的周长
        z=2*PI*x;
    return(z);                  //返回函数值
}

//定义计算圆的面积的函数 fCir_S()
float fCir_S(float x)
{
    float z=-1.0;               //声明局部变量
    if (x>=0.0)                 //如果参数大于 0，则计算圆的面积
        z=PI*x*x;
    return(z);                  //返回函数值
}
#include<iostream.h>
#include<stdlib.h>
#define MAX 30
//main()的定义
int main(void)
{
    char str[MAX],*p;

    //从键盘上输入 int 数
    cout<<"Please input a int:"<<endl;
    int n;
    cin>>n;
```

```
	//将整型数 n 按十进制转换为字符串并输出
	p=itoa(n,str,10);
	cout<<"str="<<str<<endl;
	cout<<"p="<<p<<endl;

	//将整型数 n 按十六进制转换为字符串并输出
	p=itoa(n,str,16);
	cout<<"str="<<str<<endl;
	cout<<"p="<<p<<endl;

	//从键盘上输入 double 类型的数据
	cout<<"Please input a double:"<<endl;
	double x;
	cout<<"x=";
	cin>>x;

	//将浮点数 x 转换为字符串后输出
	p=gcvt(x,10,str);
	cout<<"str="<<str<<endl;
	cout<<"p="<<p<<endl;

	return 0;
}
#include<iostream.h>
#include<stdlib.h>
#define MAX 30
//main()的定义
int main(void)
{
	char str[MAX];

	//字符串转换为 int 和 long 类型数据
	cout<<"Please input a string:"<<endl;
	cin>>str;
	int n=atoi(str);
	cout<<"n="<<n<<endl;
	long l=atol(str);
	cout<<"l="<<l<<endl;

	//字符串转换为 double 类型
	cout<<"Please input a string:"<<endl;
	cin>>str;
	double x=atof(str);
	cout<<"x="<<x<<endl;

	return 0;
}
#include<iostream.h>
#include <stdlib.h>
#include <time.h>

//定义产生[n1,n2]范围 int 随机数的函数
int rand(int n1,int n2) {
	if (n1>n2) return -1;
	if (n1==n2) return 0;
	int temp=n1+int((n2-n1)*double(rand())/RAND_MAX);
	return temp;
}
```

```
//main()函数的定义，加法练习程序
void main( void )
{
        int i;

      //使用当前的系统时间初始化随机数种子
      srand( (unsigned)time( NULL ) );

      //加法练习
      int a,b,c;
      do {
            a=rand(0,20);
            b=rand(0,20);
L1:         cout<<a<<"+"<<b<<"=";
            cin>>c;
            if (c==0) break;
            if (c!=a+b) {
                  cout<<"Error! Try again!"<<endl;
                  goto L1;
            }
            cout<<"OK!"<<endl;
      } while (1);

}
#include<iostream.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.1415926535

//main()函数的定义
void main( void )
{
      int i;
      double x=PI/180;
      cout<<"X\tSIN(X)\t\tCOS(X)"<<endl;
      cout<<"---------------------------------------"<<endl;
      for (i=0;i<=360;i=i+30) {
            cout<<i<<"\t";
            cout.precision(2);
            cout<<sin(i*x)<<"\t\t";
            cout<<cos(i*x)<<endl;
      }
}
#include<iostream.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.1415926535

//main()函数的定义
void main( void )
{
      int i;
      double d=180/PI;

      cout<<"X\tASIN(X)\t\tACOS(X)"<<endl;
      cout<<"---------------------------------------"<<endl;
      for (double x=0;x<=1.0+0.05;x=x+0.1) {
            cout<<x<<"\t";
            cout<<int(asin(x)*d)<<"\t\t";
            cout<<int(acos(x)*d)<<endl;
```

```
    }
}
#include<iostream.h>
#include <stdlib.h>
#include <math.h>

//main()函数的定义
void main( void )
{
	_complex a={3,4},b={3,-4};

	double d=cabs(a);
	cout<<"cabs("<<a.x<<","<<a.y<<")="<<d<<endl;
	cout<<"cabs("<<b.x<<","<<b.y<<")="<<cabs(b)<<endl;
}
##include<iostream.h>
#include <stdlib.h>
#include <math.h>

//main()函数的定义
void main( void )
{
	double x;

	//循环输入数据计算对数
	do {
		cout<<"x=";
		cin>>x;
		if (x<=0) break;
		cout<<"log("<<x<<")="<<log(x)<<endl;
		cout<<"log10("<<x<<")="<<log10(x)<<endl;
	} while(1);
}
#include<iostream.h>
#include <stdlib.h>
#include <math.h>

//main()函数的定义
void main( void )
{
	double y;
	for(double x=-5;x<=5;x++){
		y=exp(x);
		cout<<"exp("<<x<<")="<<y<<endl;
	}
}
#include<iostream.h>
#include <stdlib.h>
#include <math.h>

//main()函数的定义
void main( void )
{
	double y;
	int N;
	//输入一个大于等于 0 的数
	do {
		cout<<"N=";
		cin>>N;
		if (N>=0) break;
```

```
    } while (1);

    //计算并显示
    for(int i=0;i<=N;i++){
        y=pow(2,i);
        cout<<"pow("<<2<<","<<i<<")="<<y<<endl;
    }
}
#include<iostream.h>
#include <stdlib.h>
#include <math.h>

//main()函数的定义
void main( void )
{
    double y;
    for(int i=0;i<=10;i++){
        y=sqrt(i);
        cout<<"sqrt("<<i<<")="<<y<<endl;
    }
}
#include<iostream.h>
#include <time.h>

//时间延迟函数
void Dtime(int dt) {
    time_t current_time;
    time_t start_time;
    //  得到开始时间
    time(&start_time);
    do
    {
       time(&current_time);
    }
    while ((current_time - start_time) < dt);
}

//main()函数的定义
void main(void)
{
    cout<<"The First information!"<<endl;
    cout<<"About to delay 5 seconds"<<endl;
    Dtime(5);
    cout<<"The Second information!"<<endl;
}
#include<iostream.h>
#include <time.h>

//main()函数的定义
void main(void)
{
    //声明 time_t 类型的变量，其以秒为单位存放系统时间
    time_t current_time;

    //得到当前的系统时间（秒）
    time(&current_time);

    //转换系统时间为 tm 结构的时间信息
    tm    *ptime=gmtime(&current_time);
```

```
	//显示 time_t 结构的时间
	cout<<"current_time:"<<current_time<<endl;

	//显示 tm 结构的时间信息
	cout<<"seconds after the minute:"<<(ptime->tm_sec)<<endl;
	cout<<"minutes after the hour:"<<(ptime->tm_min)<<endl;
	cout<<"hours since midnight:"<<(ptime->tm_hour)<<endl;
	cout<<"day of the month:"<<(ptime->tm_mday)<<endl;
	cout<<"months since January:"<<(ptime->tm_mon)<<endl;
	cout<<"years since 1900:"<<(ptime->tm_year)<<endl;
	cout<<"days since Sunday:"<<(ptime->tm_wday)<<endl;
	cout<<"days since January 1:"<<(ptime->tm_yday)<<endl;
	cout<<"daylight savings time flag:"<<(ptime->tm_isdst)<<endl;
}
#include<iostream.h>
#include <time.h>

//main()函数的定义
void main(void)
{
	//声明变量
	time_t current_time;

	//得到当前系统时间
	time(&current_time);

	//转换系统时间为 tm 结构
	tm    *ptime=gmtime(&current_time);

	//转换 time_t 类型的时间字符串并显示
	char *timep=ctime(&current_time);
	cout<<"ctime(&current_time):"<<endl;
	cout<<timep;

	//转换 tm 类型的数据转换为时间字符串并显示
	char *tmp=asctime(ptime);
	cout<<"asctime(ptime):"<<endl;
	cout<<timep;
}
#include<iostream.h>
#include<conio.h>
#include <time.h>

//定义时间延迟函数
void Dtime(double dt) {
	time_t current_time;
	time_t start_time;

	//得到开始时间
	time(&start_time);
	//延迟处理
	do
	{
	   time(&current_time);
	}
	while (difftime(current_time,start_time)<dt);
}

//main()函数的定义
void main(void)
```

```
{
	//声明变量
	int i;
	time_t current_time;
	char *timep;
	//循环 10 次，每隔 2 秒显示一次时间
	for(i=0;i<10;i++) {
		time(&current_time);
		timep=ctime(&current_time);
		cputs(timep);
		Dtime(2);
	}
}
#include<iostream.h>
#include<stdlib.h>
#include<malloc.h>
int main(void)
{
	//定义结构类型
	struct student {
	    int num;
	    char name[20];
	    float grade;
	};

	//声明结构指针变量
	struct student *sp;
	//计算申请的内存量
	int size=sizeof(struct student);

    //申请需要的存储空间并强制类型转换
    sp=(struct student*)malloc(size);

	//为结构对象输入数据
	cout<<"nmu:";
	cin>>(sp->num);
	cout<<"name:";
	cin>>(sp->name);
	cout<<"grade:";
	cin>>(sp->grade);

	//输出结构对象的数据
	cout<<"num:"<<(sp->num)<<endl;
	cout<<"name:"<<(sp->name)<<endl;
	cout<<"grade:"<<(sp->grade);

	//释放内存
	free(sp);
}
#include<iostream.h>
#include<conio.h>
#include <time.h>

//定义时间延迟函数
void Dtime(double dt) {
	time_t current_time;
	time_t start_time;

	//  得到开始时间
	time(&start_time);
```

```
    //延迟处理
    do
    {
       time(&current_time);
    }
    while (difftime(current_time,start_time)<dt);
}

//控制台函数显示
void cputs_show(int n) {
    time_t current_time;
    char *timep;
    cputs("Show time with cputs\n");

    for(int i=0;i<5;i++) {
        time(&current_time);
        timep=ctime(&current_time);
        cputs(timep);
        Dtime(n);
    }
}

//cout 对象显示
void cout_show(int n) {
    time_t current_time;
    char *timep;
    cout<<"Show time with cout"<<endl;

    for(int i=0;i<5;i++) {
        time(&current_time);
        timep=ctime(&current_time);
        cout<<timep;
        Dtime(n);
    }
}

//main()函数的定义
void main(void)
{
    cputs_show(1);
    cout_show(1);
}
#include<stdio.h>
main()
{
    //输出字符串
    printf("He said \"Hello!\"");

    //输出各进制整数
    int i=64;
    printf("\ni=%d",i);                    //以十进制格式输出
    printf("\ni=%o",i);                    //以八进制格式输出
    printf("\ni=%x",i);                    //以十六进制格式输出
    printf("\ni=%d,%o,%x",i,i,i);          //各种格式混合输出

    //输出浮点数
    float x=3141.5926;
    printf("\nx=%f",x);      //指定输出浮点数的格式为十进制形式
    printf("\nx=%e",x);      //指定输出浮点数的格式为指数形式
```

```
    //控制输出项宽度
    int j=123;
    printf("\nj=%-10d",j);          //任选项"-"指定左对齐，W 指定宽度为 10
    printf("\nj=%10d\n",j);         //W 指定宽度为 10

    //控制输出精度
    float y=3.1415926;
    printf("y=%10.2f\n",y);    //W 指定宽度为 10，P 指定小数点后保留 2 位
    printf("y=%10.5f\n",y);    //W 指定宽度为 10，P 指定小数点后保留 5 位
}
#include<stdio.h>
main()
{
    //输入字符串
    char str[80];
    printf("str:");         //显示提示
    scanf("%s",str);
    printf("The string:%s",str);

    //输入各进制整数
    int a,b,c,sum;
    printf("\na\tb\tc\n");                    //显示提示
    scanf("%d %o %x",&a,&b,&c);   //以十进制、八进制、十六进制形式输入数据
    sum=a+b+c;
    printf("a=%d   b=%d   c=%d     sum=%d",a,b,c,sum);

    //输入浮点数并计算显示
    float x,y;                    //声明变量
    printf("\nx\ty\n");                //显示提示
    scanf("%f %f",&x,&y);              //对非空白字符"x= y="读入，不保存
    printf("sum=%f   product=%f\n",x+y, x*y);    //显示表达式的值
}
#include<iostream.h>
#include<direct.h>
#include<errno.h>
#define MAX_PATH 250
main()
{
    //声明变量
    char *p,str[MAX_PATH];

    //设置新目录
    if (mkdir("d:\\ABC")){
        cout<<"mkdir Error!"<<endl;
    }

    //更该工作目录
    if (chdir("d:\\ABC")){
        cout<<"chdir Error!"<<endl;
    }

    //读取当前目录
    if ((p=getcwd(str,MAX_PATH))==NULL) {
        cout<<"getcwd Error!"<<endl;
    }
    else
    {
        cout<<"p:"<<p<<endl;
        cout<<"str:"<<str<<endl;
    }
```

```
        //更该工作目录
        if (chdir("d:\\")){
                cout<<"chdir Error!"<<endl;
        }

        //删除指定目录
        if (rmdir("d:\\ABC")==-1)
                cout<<"rmdir Error!"<<endl;
}
#include<iostream.h>
#include <time.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>

void main( void )
{
        struct stat buf;
        int result;

        //获得 c:\Windows\Calc.exe 文件的状态信息
        result =stat( "c:\\windows\\Calc.exe", &buf );

        //显示 Calc.exe 文件的状态信息
       if( result != 0 )
              perror( "Problem getting information" );
        else
        {
                cout<<"Size of the file in bytes:"<<buf.st_size<<endl;
                cout<<"Drive number of the disk containing the file :";
                cout<<char(buf.st_dev + 'A')<<endl;
                cout<<"Time of creation of the file:"<<ctime(&buf.st_ctime);
                cout<<"Time of last access of the file:"<<ctime(&buf.st_atime);
                cout<<"Time of last modification of the file:"<<ctime(&buf.st_mtime);
       }
}
#include<iostream.h>
#include <string.h>

void main( void )
{
        //设置字符串
        char string[] = "Fill the string with something";
        cout<<"string:"<<string<<endl;
        char *p=strset(string,'*');
        cout<<"p         :"<<p<<endl;
        cout<<"string:"<<string<<endl;

        //按指定字符和指定数目设置字符数组
        char string1[] = "Fill the string with something";
        cout<<"string1:"<<string1<<endl;
        p=strnset(string1,'*',5);
        cout<<"p        :"<<p<<endl;
        cout<<"string1:"<<string1<<endl;
}
#include<iostream.h>
#include <string.h>

void main( void )
```

```
{
	//拷贝字符串常量到字符数组
	char string[80] = "Fill the string with something";
	cout<<"string:"<<string<<endl;
	cout<<"strcpy:"<<endl;
	char *p=strcpy(string,"abc");
	cout<<"p	:"<<p<<endl;
	cout<<"string:"<<string<<endl;
	char str[80];
	cout<<"str:";
	cin>>str;
	p=strcpy(string,str);
	cout<<"p	:"<<p<<endl;
	cout<<"string:"<<string<<endl;

	//拷贝前 5 个字符到 string 中
	cout<<"str:";
	cin>>str;
	cout<<"strncpy:"<<endl;
	p=strncpy(string,str,strlen(str));
	cout<<"p	:"<<p<<endl;
	cout<<"string:"<<string<<endl;
}
#include<iostream.h>
#include <string.h>

void main( void )
{
	//声明字符数组和字符型指针变量
	char string[80],*p;

	//拷贝字符串
	strcpy( string, "I'll see you");
	cout<<"string:"<<string<<endl;

	//追加字符串
	p=strcat( string, " in the morning.");
	cout<<"String: "<<string<<endl;
	cout<<"p	: "<<p<<endl;
}
#include<iostream.h>
#include <string.h>

//字符串输入函数
void str_input(char *p1,char *p2)
{
	cout<<"string1:";
	cin>>p1;
	cout<<"string2:";
	cin>>p2;
}

//显示 strcmp()函数的比较结果
void strcmp_put(char *p1,char *p2)
{
	cout<<"strcmp():"<<endl;
	int result=strcmp(p1,p2);
	if (result>0)
		cout<<p1<<" greater than "<<p2<<endl;
	if (result<0)
```

```
            cout<<p1<<" less than "<<p2<<endl;
        if (result==0)
            cout<<p1<<" identical to "<<p2<<endl;
}

//显示 stricmp()函数的比较结果
void stricmp_put(char *p1,char *p2)
{
        cout<<"stricmp():"<<endl;
        int result=stricmp(p1,p2);
        if (result>0)
            cout<<p1<<" greater than "<<p2<<endl;
        if (result<0)
            cout<<p1<<" less than "<<p2<<endl;
        if (result==0)
            cout<<p1<<" identical to "<<p2<<endl;
}

//显示 strncmp()函数的比较结果
void strncmp_put(char *p1,char *p2,size_t count )
{
        cout<<"strncmp():"<<endl;
        int result=strncmp(p1,p2,count);
        if (result>0)
            cout<<p1<<" greater than "<<p2<<endl;
        if (result<0)
            cout<<p1<<" less than "<<p2<<endl;
        if (result==0)
            cout<<p1<<" identical to "<<p2<<endl;
}

//main()函数
void main( void )
{
        //声明字符数组
        char str1[80],str2[80],p;
        int i;

        //测试测试各字符串比较函数
        for(i=1;i<=3;i++) {
            str_input(str1,str2);
            strcmp_put(str1,str2);
            stricmp_put(str1,str2);
            strncmp_put(str1,str2,3);
            cout<<"----------------------"<<endl;
        }
}
#include<iostream.h>
#include <string.h>

//main()函数
void main( void )
{
        //声明字符数组
        char string[80],*p;
        int i;

        //转换字符串中的小写字母为大写
        cout<<"Convert a string to uppercase:"<<endl;
        cout<<"string:";
```

```
        cin>>string;
        p=strupr(string);
        cout<<"p:"<<p<<endl;
        cout<<"string:"<<string<<endl;
        cout<<"----------------------"<<endl;

        //转换字符串中的大写字母为小写
        cout<<"Convert a string to lowercase:"<<endl;
        cout<<"string:";
        cin>>string;
        p=strlwr(string);
        cout<<"p:"<<p<<endl;
        cout<<"string:"<<string<<endl;
}
#include<iostream.h>
#include <string.h>

//main()函数
void main( void )
{
        //声明字符数组
        char string[]="This is a test.";
        int n;

        //获得字符串的长度
        cout<<"string:"<<string<<endl;
        n=strlen(string);
        cout<<"The length of "<<"\""<<string<<"\": "<<n<<endl;

        //输入字符并计算其长度
        cout<<"string:";
        cin>>string;
        n=strlen(string);
        cout<<"The length of "<<"\""<<string<<"\": "<<n<<endl;
}
#include<iostream.h>
#include <string.h>

//main()函数
void main( void )
{
        //声明字符数组
        char ch,string[80],*p;
        int n;

        //输入字符串和要查找的字符
        cout<<"Test strchr():"<<endl;
        cout<<"string:";
        cin>>string;
        cout<<"ch       :";
        cin>>ch;

        //在 string 中查找 ch 中的字符并显示
        p=strchr(string,ch);
        cout<<"p        :"<<p<<endl;

        //输入字符串和要查找的字符串并查找
        char substr[80];
        cout<<"Test strstr():"<<endl;
        cout<<"substr:";
```

```
    cin>>substr;

    //在 string 中查找 substr 中的字符串并显示
    p=strstr(string,substr);
    cout<<"p       :"<<p<<endl;
}
#include<iostream.h>
#include <string.h>

//main()函数
void main( void )
{
    //声明字符数组
    char string[80],*p;

    //输入字符串并将其反转
    cout<<"string:";
    cin>>string;
    p=strrev(string );
    cout<<"p         :"<<p<<endl;
    cout<<"string:"<<string<<endl;
}
#include<iostream.h>
#include <string.h>

char string[80];
char seps[]     = " ,\t\n";
char *token;

void main( void )
{
    //从键盘上输入两个语句
    for (int i=1;i<3;i++) {
        cout<<"Please input a sentence:"<<endl;
        //整行输入
        cin.getline(string,80);
        cout<<"Tokens:"<<endl;
        //首次分离字符串
        token = strtok( string, seps );
        while( token != NULL )                    //结束分离判断
        {
            cout<<token<<endl;
            //下次分离字符串
            token = strtok( NULL, seps );
        }
    }
}
#include<iostream.h>
#include<stdio.h>
#include <string.h>

//main()函数
void main( void )
{
    //声明变量和数组
    char    buffer[200], s[] = "computer", c = 'l';
    int     i = 35, j;
    float fp = 1.7320534f;

    //格式化输出到 buffer
```

```
    j    = sprintf( buffer,          "\tString:       %s\n", s );
    j += sprintf( buffer + j, "\tCharacter: %c\n", c );
    j += sprintf( buffer + j, "\tInteger:      %d\n", i );
    j += sprintf( buffer + j, "\tReal:           %f\n", fp );

    cout<<"Output:"<<endl;
    cout<<buffer;
    cout<<"character count ="<<j<<endl;
}
//根据半径计算圆的周长和面积
#include <iostream.h>
const float PI=3.1416;              //声明常量(只读变量)PI 为 3.1416
float fCir_L(float);                //声明自定义函数 fCir_L()的原型
float fCir_S(float);                //声明自定义函数 fCir_S()的原型

//以下是 main()函数
main()
{
    float r,l,s;                    //声明 3 个变量

    cout<<"R=";                     //显示字符串
    cin>>r;                         //键盘输入
    l=fCir_L(r);                    //计算圆的周长，赋值给变量 l
    s=fCir_S(r);                    //计算圆的面积，赋值给变量 s
    cout<<"l="<<l;                  //显示计算结果
    cout<<"\ns="<<s;
}

//定义计算圆的周长的函数 fCir_L()
float fCir_L(float x)
{
    float z=-1.0;                   //声明局部变量
    if (x>=0.0)                     //如果参数大于 0，则计算圆的周长
        z=2*PI*x;
    return(z);                      //返回函数值
}

//定义计算圆的面积的函数 fCir_S()
float fCir_S(float x)
{
    float z=-1.0;                   //声明局部变量
    if (x>=0.0)                     //如果参数大于 0，则计算圆的面积
        z=PI*x*x;
    return(z);                      //返回函数值
}
#include<iostream.h>

//定义名为 max_value 的函数模板
template <class T> T max_value (T a,T b)
{
    return ((a> b)? a: b);
}

//在 main()函数中测试 max_value 函数模板
void main(void)
{
    //double 类型数据使用 max_value 模板函数
    double x = 1.2, y = 2.1;
    cout<<"x="<<x<<"\t";
    cout<<"y="<<y<<endl;
```

```
	double result=max_value(x,y);
	cout<<"max_value(x,y)="<<result<<endl;
	cout<<"max_value(2*3.0,2+3.0)="<<max_value(2*3.0,2+3.0)<<endl;
	cout<<"------------------"<<endl;

	//int 类型数据使用 max_value 模板函数
	int n= 1, m= 6;
	cout<<"n="<<n<<"\t";
	cout<<"m="<<m<<endl;
	cout<<"max_value(n,m)="<<max_value(n,m)<<endl;
	cout<<"------------------"<<endl;

	//char 类型数据使用 max_value 模板函数
	char ch1='A',ch2='a';
	cout<<"ch1="<<ch1<<"\t";
	cout<<"ch2="<<ch2<<endl;
	cout<<"max_value(ch1,ch2)="<<max_value(ch1,ch2)<<endl;
	cout<<"------------------"<<endl;

	//字符串数据使用 max_value 模板函数
	char str1[]="abc",str2[]="ABC",*p;
	p=max_value(str1,str2);
	cout<<"max_value("<<str1<<","<<str2<<")="<<p<<endl;
}
#include<iostream.h>

//函数模板的原型
template <class T1, class T2> void display(T1 x, T2 y);

//在 main()函数中测试 display 函数模板
void main(void)
{
	//声明变量
	char c='A';
	char str[]="This is a test";
	int n=10;
	float x=1.5;
	double z=3.1415926;

	//两个参数类型相同
	display(c, char(c+2));
	display(str, str);
	display(n, 2*n);
	display(x,2*x);
	display(z, 2*z);
	cout<<"------------------"<<endl;

	//两个参数类型不同
	display(c, str);
	display(str, c);
	display(n, str);
	display(str,2*x);
	display(z, n);
}

//定义名为 display 的函数模板
template <class T1, class T2> void display(T1 x, T2 y)
{
	cout << x << " " << y << endl;
}
```

```
#include<iostream.h>

//声明引用参数的函数模板原型
template <class T> void swap(T &x, T &y);

//定义一个结构类型
struct student {
	int n;
	char name[20];
	float grade;
};

//在 main()函数中测试 swap()函数模板
void main(void)
{
	//交换两个 int 型变量中的数据
	int m=3,n=5;
	cout<<"m="<<m<<"    n="<<n<<endl;
	swap(m,n);
	cout<<"m="<<m<<"    n="<<n<<endl;
	cout<<"-------------------"<<endl;

	//交换两个 double 型变量中的数据
	double x=3.5,y=5.7;
	cout<<"x="<<x<<"    y="<<y<<endl;
	swap(x,y);
	cout<<"x="<<x<<"    y="<<y<<endl;
	cout<<"-------------------"<<endl;

	//交换两个 char 型变量中的数据
	char c1='A',c2='a';
	cout<<"c1="<<c1<<"    c2="<<c2<<endl;
	swap(c1,c2);
	cout<<"c1="<<c1<<"    c2="<<c2<<endl;
	cout<<"-------------------"<<endl;

	//交换两个结构变量中的数据
	student s1={1001,"ZhangHua",90};
	student s2={1011,"LiWei",95.5};
	cout<<"s1:    ";
	cout<<s1.n<<"    "<<s1.name<<"    "<<s1.grade<<endl;
	cout<<"s2:    ";
	cout<<s2.n<<"    "<<s2.name<<"    "<<s2.grade<<endl;
	swap(s1,s2);
	cout<<"swap(s1,s2):"<<endl;
	cout<<"s1:    ";
	cout<<s1.n<<"    "<<s1.name<<"    "<<s1.grade<<endl;
	cout<<"s2:    ";
	cout<<s2.n<<"    "<<s2.name<<"    "<<s2.grade<<endl;
}

//定义名为 swap 的函数模板用于交换两个变量中的数据
template <class T> void swap(T &x, T &y)
{
	T temp;
	temp=x;
	x=y;
	y=temp;
}
#include<iostream.h>
```

```
//声明函数模板的原型语句
template <class T> void swap(T *x, T *y);

//定义一个结构类型
struct student {
	int n;
	char name[20];
	float grade;
};

//在 main()函数中测试 swap()函数模板
void main(void)
{
	//交换两个 int 型变量中的数据
	int m=3,n=5;
	cout<<"m="<<m<<"    n="<<n<<endl;
	swap(&m,&n);
	cout<<"m="<<m<<"    n="<<n<<endl;
	cout<<"-------------------"<<endl;

	//交换两个 double 型变量中的数据
	double x=3.5,y=5.7;
	cout<<"x="<<x<<"    y="<<y<<endl;
	swap(&x,&y);
	cout<<"x="<<x<<"    y="<<y<<endl;
	cout<<"-------------------"<<endl;

	//交换两个 char 型变量中的数据
	char c1='A',c2='a';
	cout<<"c1="<<c1<<"    c2="<<c2<<endl;
	swap(&c1,&c2);
	cout<<"c1="<<c1<<"    c2="<<c2<<endl;
	cout<<"-------------------"<<endl;

	//交换两个结构变量中的数据
	student s1={1001,"ZhangHua",90};
	student s2={1011,"LiWei",95.5};
	cout<<"s1:    ";
	cout<<s1.n<<"    "<<s1.name<<"    "<<s1.grade<<endl;
	cout<<"s2:    ";
	cout<<s2.n<<"    "<<s2.name<<"    "<<s2.grade<<endl;

	swap(&s1,&s2);
	cout<<"swap(s1,s2):"<<endl;
	cout<<"s1:    ";
	cout<<s1.n<<"    "<<s1.name<<"    "<<s1.grade<<endl;
	cout<<"s2:    ";
	cout<<s2.n<<"    "<<s2.name<<"    "<<s2.grade<<endl;
}

//定义名为 swap 的函数模板用于交换两个变量中的数据
template <class T> void swap(T *x, T *y)
{
	T temp;
	temp=*x;
	*x=*y;
	*y=temp;
}
#include<iostream.h>
```

```
//定义输入函数模板
template <class T> void input(char *str,T &x) {
	cout<<str<<"=";
	cin>>x;
}

//定义输出函数模板
template <class T> void output(char *str,T x) {
	cout<<str<<"="<<x<<endl;
}

//在 main()函数中测试输入输出函数模板
void main(void)
{
	//输入输出 int 型数据
	int a,b;
	input("a",a);
	output("a",a);
	b=3*a;
	output("3*a",b);
	output("a+b",a+b);
	cout<<"--------------------"<<endl;

	//输入输出 double 型数据
	double x,y;
	input("x",x);
	output("x",x);
	y=2*x;
	output("y",y);
	cout<<"--------------------"<<endl;

	//输入输出 char 型数据
	char c1;
	input("c1",c1);
	output("c1+2",char(c1+2));
	cout<<"--------------------"<<endl;

	//输入输出字符串数据
	char string[80];
	input("string",string);
	output("string",string);
}
#include<iostream.h>
#include<string.h>

//显示数组的函数模板
template <class T> void arr_put(T arr[],int size) {
	for (int i=0 ;i<=size;i++)
	    cout<<arr[i]<<" ";
	cout<<endl;
}

//选择排序数组的函数模板
template <class T> void sort(T arr[],int size) {
	T temp;
	int i,j;
	for (i=0;i<size;i++)
	      for (j=i+1;j<=size;j++)
	            if (arr[i]<=arr[j])
```

```
                {
                    temp=arr[i];
                    arr[i]=arr[j];
                    arr[j]=temp;
                }
}

//在 main()函数中测试数组排序的函数模板
void main(void)
{
    //用排序函数模板处理 int 型数组
    cout<<"int:"<<endl;
    int a[]={1,5,2,7,9,0,10,-1};
    arr_put(a,7);
    sort(a,7);
    arr_put(a,7);

    //用排序函数模板处理 double 型数组
    cout<<"double:"<<endl;
    double x[]={1.2,2.1,1.414,1.732};
    arr_put(x,3);
    sort(x,3);
    arr_put(x,3);

    //用排序函数模板处理 char 类型数组
    cout<<"char:"<<endl;
    char str[80];
    cout<<"str:";
    cin>>str;
    int size=strlen(str);
    arr_put(str,size);
    sort(str,size);
    arr_put(str,size);
}
#include<iostream.h>
#include<string.h>

//显示数组的函数模板
template <class T> void arr_put(T arr[],int size) {
    for (int i=0 ;i<size;i++)
        cout<<arr[i]<<" ";
    cout<<endl;
}

//选择法对数组排序的函数模板
template <class T> void sort(T arr[],int size) {
    T temp;
    int i,j;
    for (i=0;i<size-1;i++)
        for (j=i+1;j<size;j++)
            if (arr[i]>arr[j])
            {
                temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;
            }
}

//二分查找法的函数模板
template <class T> int binary_search(T array[], T value, int size)
```

```
{
    int found = 0;
    int high = size, low = 0, mid;

    mid = (high + low) / 2;

    cout<<"Looking for "<<value<<endl;
    while ((! found) && (high >= low))
    {
       if (value == array[mid])
          found = 1;
       else if (value < array[mid])
          high = mid - 1;
       else
          low = mid + 1;
          mid = (high + low) / 2;
    }
    return((found) ? mid: -1);
}

//main()函数中使用处理数组的函数模板
void main(void)
{
    //处理 int 型数组
    int array[10]={1,3,5,7,9,2,4,6,8,10};

    //显示数组初值
    arr_put(array,10);

    //对数组排序并显示
    sort(array,10);
    arr_put(array,10);

    //查找数组
    cout<<"Result of search: "<<binary_search(array, 3, 10)<<endl;
    cout<<"Result of search: "<<binary_search(array, 2, 10)<<endl;
    cout<<"Result of search: "<<binary_search(array, 9, 10)<<endl;
    cout<<"Result of search: "<<binary_search(array, 5, 10)<<endl;
    cout<<"-----------------------------"<<endl;

    //处理字符串型数组
    char ch1,str[]="happy";
    int size=strlen(str);

    //显示数组初值
    arr_put(str,size);

    //对数组排序并显示
    sort(str,size);
    arr_put(str,size);

    //查找数组
    cout<<"Input a char:";
    cin>>ch1;
    cout<<"Result of search: "<<binary_search(str, ch1, size)<<endl;
}
#include<iostream.h>

//定义名为 ex_class 的类模板
template <class T>    class ex_class
```

```
{
	T value;
public:
	ex_class(T v) { value=v; }
	void set_value(T v) { value=v; }
	T get_value(void) {return value;}
};

//main()函数中测试 ex_class 类模板
void main(void)
{
	//测试 int 类型数据
	ex_class <int> a(5),b(10);
	cout<<"a.value:"<<a.get_value()<<endl;
	cout<<"b.value:"<<b.get_value()<<endl;

	//测试 char 类型数据
	ex_class <char> ch('A');
	cout<<"ch.value:"<<ch.get_value()<<endl;
	ch.set_value('a');
	cout<<"ch.value:"<<ch.get_value()<<endl;

	//测试 double 类型数据
	ex_class <double> x(5.5);
	cout<<"x.value:"<<x.get_value()<<endl;
	x.set_value(7.5);
	cout<<"x.value:"<<x.get_value()<<endl;
}
#include <iostream.h>
//定义栈的尺寸
const int SIZE = 100;

//定义处理栈的类模板接口
template <class T> class stack {
	T stck[SIZE];
	int tos;
public:
	stack(void) {
		tos = 0;
		cout << "Stack Initialized." << endl;
	}
	~stack(void) {
	    cout << "Stack Destroyed." << endl;
	}
	void push(T);
	T pop(void);
};

//定义栈的成员函数
template <class T> void stack<T>::push(T i)
{
	if(tos==SIZE)
	{
		cout << "Stack is full." << endl;
		return;
	}
	stck[tos++] = i;
}
template <class T> T stack<T>::pop(void)
{
```

```
    if(tos==0)
    {
        cout << "Stack underflow." << endl;
        return 0;
    }
    return stck[--tos];
}

//main()函数中测试 stack 类模板
void main(void)
{
    //处理 int 类型数据的栈
    cout<<"stack<int> a :"<<endl;
    stack<int> a;
    a.push(1);
    a.push(2);
    cout << a.pop() << " ";
    cout << a.pop() << endl;

    //处理 double 类型数据的栈
    cout<<"stack<double> b :"<<endl;
    stack<double> b;
    b.push(99.3);
    b.push(-12.23);
    cout << b.pop() << " ";
    cout << b.pop() <<endl;

    //处理 char 类型数据的栈
    cout<<"stack<char> c :"<<endl;
    stack<char> c;
    for(int i=0; i<10; i++)
       c.push((char) 'A' + i);
    for(i=0; i<10; i++)
       cout <<c.pop();
    cout << endl;
}
#include<iostream.h>

//定义名为 ex_class 的类模板
template <class T1,class T2>    class ex_class
{
    T1 value1;
    T2 value2;
public:
    ex_class(T1 v1,T2 v2) {
        value1=v1;
        value2=v2;
    }
    void set_value(T1 v1,T2 v2) {
        value1=v1;
        value2=v2;
    }
    void put_value(void) {
        cout<<"valu1="<<value1<<endl;
        cout<<"valu2="<<value2<<endl;
    }
};

//main()函数中测试 ex_class 类模板
void main(void)
```

```
{
	//测试 int 和 double 类型数据
	ex_class <int,double> a(5,1.5);
	cout<<"ex_class <int,double> a:"<<endl;
	a.put_value();
	a.set_value(100,3.14);
	a.put_value();

	//测试 double 和 int 类型数据
	ex_class <double,int> b(0.5,5);
	cout<<"ex_class <double,int> b:"<<endl;
	b.put_value();
	b.set_value(1.732,100);
	b.put_value();

	//测试 char 和 int 类型数据
	ex_class <char,int> c('a',5);
	cout<<"ex_class <char,int> c:"<<endl;
	c.put_value();
	c.set_value('B',100);
	c.put_value();

	//测试 int 和 int 类型数据
	ex_class <int,int> d(5,10);
	cout<<"ex_class <int,int> d:"<<endl;
	d.put_value();
	d.set_value(100,200);
	d.put_value();
}
#include <iostream>
#include <list>
#include <numeric>
#include <algorithm>

using namespace std;

//创建一个 list 容器的实例 LISTINT
typedef list<int> LISTINT;

//创建一个 list 容器的实例 LISTCHAR
typedef list<int> LISTCHAR;

void main(void)
{
	//--------------------------
	//用 list 容器处理整型数据
	//--------------------------
	//用 LISTINT 创建一个名为 listOne 的 list 对象
	LISTINT listOne;
	//声明 i 为迭代器
	LISTINT::iterator i;

	//从前面向 listOne 容器中添加数据
	listOne.push_front (2);
	listOne.push_front (1);

	//从后面向 listOne 容器中添加数据
	listOne.push_back (3);
	listOne.push_back (4);
```

```
	//从前向后显示 listOne 中的数据
	cout<<"listOne.begin()--- listOne.end():"<<endl;
	for (i = listOne.begin(); i != listOne.end(); ++i)
		cout << *i << " ";
	cout << endl;

	//从后向后显示 listOne 中的数据
	LISTINT::reverse_iterator ir;
	cout<<"listOne.rbegin()---listOne.rend():"<<endl;
	for (ir =listOne.rbegin(); ir!=listOne.rend();ir++) {
		cout << *ir << " ";
	}
	cout << endl;

	//使用 STL 的 accumulate(累加)算法
	int result = accumulate(listOne.begin(), listOne.end(),0);
	cout<<"Sum="<<result<<endl;
	cout<<"------------------"<<endl;

	//--------------------------
	//用 list 容器处理字符型数据
	//--------------------------

	//用 LISTCHAR 创建一个名为 listOne 的 list 对象
	LISTCHAR listTwo;
	//声明 i 为迭代器
	LISTCHAR::iterator j;

	//从前面向 listTwo 容器中添加数据
	listTwo.push_front ('A');
	listTwo.push_front ('B');

	//从后面向 listTwo 容器中添加数据
	listTwo.push_back ('x');
	listTwo.push_back ('y');

	//从前向后显示 listTwo 中的数据
	cout<<"listTwo.begin()---listTwo.end():"<<endl;
	for (j = listTwo.begin(); j != listTwo.end(); ++j)
		cout << char(*j) << " ";
	cout << endl;

	//使用 STL 的 max_element 算法求 listTwo 中的最大元素并显示
	j=max_element(listTwo.begin(),listTwo.end());
	cout << "The maximum element in listTwo is: "<<char(*j)<<endl;
}
#include <iostream>
#include <vector>

using namespace std;
typedef vector<int> INTVECTOR;

//测试 vector 容器的功能
void main(void)
{
	//vec1 对象初始为空
	INTVECTOR vec1;
	//vec2 对象最初有 10 个值为 6 的元素
	INTVECTOR vec2(10,6);
	//vec3 对象最初有 3 个值为 6 的元素
```

```
INTVECTOR vec3(vec2.begin(),vec2.begin()+3);

//声明一个名为 i 的双向迭代器
INTVECTOR::iterator i;

//从前向后显示 vec1 中的数据
cout<<"vec1.begin()--vec1.end():"<<endl;
for (i =vec1.begin(); i !=vec1.end(); ++i)
	cout << *i << " ";
cout << endl;

//从前向后显示 vec2 中的数据
cout<<"vec2.begin()--vec2.end():"<<endl;
for (i =vec2.begin(); i !=vec2.end(); ++i)
	cout << *i << " ";
cout << endl;

//从前向后显示 vec3 中的数据
cout<<"vec3.begin()--vec3.end():"<<endl;
for (i =vec3.begin(); i !=vec3.end(); ++i)
	cout << *i << " ";
cout << endl;

//测试添加和插入成员函数
vec1.push_back(2);
vec1.push_back(4);
vec1.insert(vec1.begin()+1,5);
vec1.insert(vec1.begin()+1,vec3.begin(),vec3.end());
cout<<"push() and insert():" <<endl;
for (i =vec1.begin(); i !=vec1.end(); ++i)
	cout << *i << " ";
cout << endl;

//测试赋值成员函数
vec2.assign(8,1);
cout<<"vec2.assign(8,1):" <<endl;
for (i =vec2.begin(); i !=vec2.end(); ++i)
	cout << *i << " ";
cout << endl;

//测试引用类函数
cout<<"vec1.front()="<<vec1.front()<<endl;
cout<<"vec1.back()="<<vec1.back()<<endl;
cout<<"vec1.at(4)="<<vec1.at(4)<<endl;
cout<<"vec1[4]="<<vec1[4]<<endl;

//测试移出和删除
vec1.pop_back();
vec1.erase(vec1.begin()+1,vec1.end()-2);
cout<<"vec1.pop_back() and vec1.erase():" <<endl;
for (i =vec1.begin(); i !=vec1.end(); ++i)
	cout << *i << " ";
cout << endl;

//显示序列的状态信息
cout<<"vec1.capacity(): "<<vec1.capacity()<<endl;
cout<<"vec1.max_size(): "<<vec1.max_size()<<endl;
cout<<"vec1.size(): "<<vec1.size()<<endl;
cout<<"vec1.empty(): "<<vec1.empty()<<endl;
```

```
	//vector 序列容器的运算
	cout<<"vec1==vec3: "<<(vec1==vec3)<<endl;
	cout<<"vec1<=vec3: "<<(vec1<=vec3)<<endl;
}
#include <iostream>
#include <deque>

using namespace std;
typedef deque<int> INTDEQUE;

//从前向后显示 deque 队列的全部元素
void put_deque(INTDEQUE deque, char *name)
{
	INTDEQUE::iterator pdeque;

	cout << "The contents of " << name << " : ";
	for(pdeque = deque.begin(); pdeque != deque.end(); pdeque++)
		cout << *pdeque << " ";
	cout<<endl;
}

//测试 deqtor 容器的功能
void main(void)
{
	//deq1 对象初始为空
	INTDEQUE deq1;
	//deq2 对象最初有 10 个值为 6 的元素
	INTDEQUE deq2(10,6);
	//deq3 对象最初有 3 个值为 6 的元素
	INTDEQUE deq3(deq2.begin(),deq2.begin()+3);

	//声明一个名为 i 的双向迭代器变量
	INTDEQUE::iterator i;

	//从前向后显示 deq1 中的数据
	put_deque(deq1,"deq1");

	//从前向后显示 deq2 中的数据
	put_deque(deq2,"deq2");

	//从前向后显示 deq3 中的数据
	put_deque(deq3,"deq3");

	//从 deq1 序列后面添加两个元素
	deq1.push_back(2);
	deq1.push_back(4);
	cout<<"deq1.push_back(2) and deq1.push_back(4):"<<endl;
	put_deque(deq1,"deq1");

	//从 deq1 序列前面添加两个元素
	deq1.push_front(5);
	deq1.push_front(7);
	cout<<"deq1.push_front(5) and deq1.push_front(7):"<<endl;
	put_deque(deq1,"deq1");

	//在 deq1 序列中间插入数据
	deq1.insert(deq1.begin()+1,3,9);
	cout<<"deq1.insert(deq1.begin()+1,3,9):"<<endl;
	put_deque(deq1,"deq1");
```

```
	//测试引用类函数
	cout<<"deq1.front()="<<deq1.front()<<endl;
	cout<<"deq1.back()="<<deq1.back()<<endl;
	cout<<"deq1.at(4)="<<deq1.at(4)<<endl;
	cout<<"deq1[4]="<<deq1[4]<<endl;
	deq1.at(1)=10;
	deq1[2]=12;
	cout<<"deq1.at(1)=10 and deq1[2]=12 :"<<endl;
	put_deque(deq1,"deq1");

	//从 deq1 序列的前后各移去一个元素
	deq1.pop_front();
	deq1.pop_back();
	cout<<"deq1.pop_front() and deq1.pop_back():"<<endl;
	put_deque(deq1,"deq1");

	//清除 deq1 中的第 2 个元素
	deq1.erase(deq1.begin()+1);
	cout<<"deq1.erase(deq1.begin()+1):"<<endl;
	put_deque(deq1,"deq1");

	//对 deq2 赋值并显示
	deq2.assign(8,1);
	cout<<"deq2.assign(8,1):"<<endl;
	put_deque(deq2,"deq2");

	//显示序列的状态信息
	cout<<"deq1.max_size(): "<<deq1.max_size()<<endl;
	cout<<"deq1.size(): "<<deq1.size()<<endl;
	cout<<"deq1.empty(): "<<deq1.empty()<<endl;

	//deqtor 序列容器的运算
	cout<<"deq1==deq3: "<<(deq1==deq3)<<endl;
	cout<<"deq1<=deq3: "<<(deq1<=deq3)<<endl;
}
#include <iostream>
#include <list>

using namespace std;
typedef list<int> INTLIST;

//从前向后显示 list 队列的全部元素
void put_list(INTLIST list, char *name)
{
	INTLIST::iterator plist;

	cout << "The contents of " << name << " : ";
	for(plist = list.begin(); plist != list.end(); plist++)
		cout << *plist << " ";
	cout<<endl;
}

//测试 list 容器的功能
void main(void)
{
	//list1 对象初始为空
	INTLIST list1;
	//list2 对象最初有 10 个值为 6 的元素
	INTLIST list2(10,6);
	//list3 对象最初有 3 个值为 6 的元素
```

```
INTLIST list3(list2.begin(),--list2.end());

//声明一个名为 i 的双向迭代器
INTLIST::iterator i;

//从前向后显示各 list 对象的元素
put_list(list1,"list1");
put_list(list2,"list2");
put_list(list3,"list3");

//从 list1 序列后面添加两个元素
list1.push_back(2);
list1.push_back(4);
cout<<"list1.push_back(2) and list1.push_back(4):"<<endl;
put_list(list1,"list1");

//从 list1 序列前面添加两个元素
list1.push_front(5);
list1.push_front(7);
cout<<"list1.push_front(5) and list1.push_front(7):"<<endl;
put_list(list1,"list1");

//在 list1 序列中间插入数据
list1.insert(++list1.begin(),3,9);
cout<<"list1.insert(list1.begin()+1,3,9):"<<endl;
put_list(list1,"list1");

//测试引用类函数
cout<<"list1.front()="<<list1.front()<<endl;
cout<<"list1.back()="<<list1.back()<<endl;

//从 list1 序列的前后各移去一个元素
list1.pop_front();
list1.pop_back();
cout<<"list1.pop_front() and list1.pop_back():"<<endl;
put_list(list1,"list1");

//清除 list1 中的第 2 个元素
list1.erase(++list1.begin());
cout<<"list1.erase(++list1.begin()):"<<endl;
put_list(list1,"list1");

//对 list2 赋值并显示
list2.assign(8,1);
cout<<"list2.assign(8,1):"<<endl;
put_list(list2,"list2");

//显示序列的状态信息
cout<<"list1.max_size(): "<<list1.max_size()<<endl;
cout<<"list1.size(): "<<list1.size()<<endl;
cout<<"list1.empty(): "<<list1.empty()<<endl;

//list 序列容器的运算
put_list(list1,"list1");
put_list(list3,"list3");
cout<<"list1>list3: "<<(list1>list3)<<endl;
cout<<"list1<list3: "<<(list1<list3)<<endl;

//对 list1 容器排序
list1.sort();
```

```
	put_list(list1,"list1");

	//结合处理
	list1.splice(++list1.begin(), list3);
	put_list(list1,"list1");
	put_list(list3,"list3");
}
#include <iostream.h>
#include <set>

using namespace std;

//创建 set 模板的实例
typedef set<int> SET_INT;

//put_HTset 函数，从头向尾显示 set 容器的所有元素
void put_HTset(SET_INT set1,char *name)
{
	SET_INT::iterator it;

	cout<<name<<": ";
	cout<<"Head to Tail=";
	for (it=set1.begin();it!=set1.end();++it)
		cout<<(*it)<<" ";
	cout<<endl;
}

//put_THset 函数，从尾向头显示 set 容器的所有元素
void put_THset(SET_INT s1,char *name)
{
	SET_INT::reverse_iterator i;

	cout<<name<<": ";
	cout<<"Tail to Head=";
	for (i=s1.rbegin(); i!=s1.rend();i++)
		cout <<(*i) <<" ";
	cout<<endl;
}

//测试 set 模板
void main(void)
{
	int i;
	//声明 set 的对象和迭代器
	SET_INT s1;		//容器初始尾空
	SET_INT::iterator it;

	//向 s1 对象中插入值
	for (i=1;i<20;i=i+2) {
		s1.insert(i);
	}

	//正向显示 s1 中的数据
	put_HTset(s1,"s1");

	//反向显示 s1 中的数据
	put_THset(s1,"s1");

	//构造含有元素的序列并显示
	SET_INT s2(s1);
```

```
	put_HTset(s2,"s2");

	//删除 s2 的第 2 个元素并显示
	s2.erase(++s2.begin());
	put_HTset(s2,"s2");

	//向 s2 插入 8 和 9 并显示
	s2.insert(8);
	s2.insert(9);
	put_HTset(s2,"s2");

	//清空 s2 的序列
	s2.clear();
	put_HTset(s2,"s2");

	//按关键给定的区间显示序列中的元素
	cout<<"[s1.lower_bound(5),s1.upper_bound(15)] :";
	for (it=s1.lower_bound(4);it!=s1.upper_bound(16);it++)
		cout<<(*it)<<" ";
	cout<<endl;

	//显示 s1 的状态信息
	cout<<"s1.size():"<<s1.size()<<endl;
	cout<<"s1.max_size():"<<s1.max_size()<<endl;
	cout<<"s1.count(15):"<<s1.count(15)<<endl;

	//交换两个 set 容器的元素并显示
	s1.swap(s2);
	put_HTset(s1,"s1");
	put_HTset(s2,"s2");

	//关系运算
	s1.insert(5);
	cout<<"s1>s2 = "<<(s1>s2)<<endl;
}
#include <iostream.h>
#include <set>

using namespace std;

//创建 multiset 模板的实例
typedef multiset<int> MULTISET_INT;

//put_HTset 函数，从头向尾显示 multiset 容器的所有元素
void put_HTset(MULTISET_INT set1,char *name)
{
	MULTISET_INT::iterator it;

	cout<<name<<": ";
	cout<<"Head to Tail=";
	for (it=set1.begin();it!=set1.end();++it)
		cout<<(*it)<<" ";
	cout<<endl;
}

//put_THset 函数，从尾向头显示 multiset 容器的所有元素
void put_THset(MULTISET_INT s1,char *name)
{
	MULTISET_INT::reverse_iterator i;
```

```
	cout<<name<<": ";
	cout<<"Tail to Head=";
	for (i=s1.rbegin(); i!=s1.rend();i++)
		cout <<(*i) <<" ";
	cout<<endl;
}

//测试 multiset 模板
void main(void)
{
	int i;
	//声明 multiset 的对象和迭代器
	MULTISET_INT s1;		//容器初始尾空
	MULTISET_INT::iterator it;

	//向 s1 对象中插入值
	for (i=1;i<20;i=i+2) {
		s1.insert(i);
	}

	//正向显示 s1 中的数据
	put_HTset(s1,"s1");

	//反向显示 s1 中的数据
	put_THset(s1,"s1");

	//构造含有元素的序列并显示
	MULTISET_INT s2(s1);
	put_HTset(s2,"s2");

	//删除 s2 的第 2 个元素并显示
	s2.erase(++s2.begin());
	put_HTset(s2,"s2");

	//向 s2 插入 8 和 9 并显示
	s2.insert(8);
	s2.insert(9);
	put_HTset(s2,"s2");

	//清空 s2 的序列
	s2.clear();
	put_HTset(s2,"s2");

	//按键给定的区间显示序列中的元素
	cout<<"[s1.lower_bound(5),s1.upper_bound(15)] :";
	for (it=s1.lower_bound(4);it!=s1.upper_bound(16);it++)
		cout<<(*it)<<" ";
	cout<<endl;

	//显示 s1 的状态信息
	cout<<"s1.size():"<<s1.size()<<endl;
	cout<<"s1.max_size():"<<s1.max_size()<<endl;
	cout<<"s1.count(15):"<<s1.count(15)<<endl;

	//交换两个 multiset 容器的元素并显示
	s1.swap(s2);
	put_HTset(s1,"s1");
	put_HTset(s2,"s2");

	//关系运算
```

```
	s1.insert(2);
	put_HTset(s1,"s1");
	put_HTset(s2,"s2");
	cout<<"s1>s2 = "<<(s1>s2)<<endl;
}
#include <iostream>
#include <string>
#include <map>

using namespace std;

//创建 map 的实例，整数(int)映射字符串(string)
typedef map<int, string> INT2STRING;

//测试 map 容器
void main()
{
	//创建 map 对象 theMap
	INT2STRING theMap;
	INT2STRING::iterator theIterator,it;

	//向 theMap 容器中添入数据，数字和字符串配对
	//每个元素是一个映射对
	theMap.insert(INT2STRING::value_type(0,"Zero"));
	theMap.insert(INT2STRING::value_type(2,"Two"));
	theMap.insert(INT2STRING::value_type(4,"Four"));
	theMap.insert(INT2STRING::value_type(6,"Six"));
	theMap.insert(INT2STRING::value_type(8,"Eight"));

	//显示 map 容器的所有对象
	cout<<"theMap.begin()--theMap.end():"<<endl;
	for (theIterator=theMap.begin();theIterator!=theMap.end();++theIterator){
		cout<<(*theIterator).first;
		cout<<","<<(*theIterator).second<<" ";
	}
	cout<<endl;

	//测试 map 容器 key 的惟一性
	theMap.insert(INT2STRING::value_type(0,"Zero"));
	theMap.insert(INT2STRING::value_type(1,"One"));
	theMap.insert(INT2STRING::value_type(2,"Two"));
	theMap.insert(INT2STRING::value_type(3,"Three"));
	theMap.insert(INT2STRING::value_type(4,"Four"));
	theMap.insert(INT2STRING::value_type(5,"Five"));
	theMap.insert(INT2STRING::value_type(6,"Six"));
	theMap.insert(INT2STRING::value_type(7,"Seven"));
	theMap.insert(INT2STRING::value_type(8,"Eight"));
	theMap.insert(INT2STRING::value_type(9,"Nine"));
	//下列语句将不能插入到 map 容器中
	theMap.insert(INT2STRING::value_type(5,"AAA"));

	//显示 map 容器的所有对象
	cout<<"theMap.begin()--theMap.end():"<<endl;
	for (theIterator=theMap.begin();theIterator!=theMap.end();++theIterator){
		cout<<(*theIterator).first;
		cout<<","<<(*theIterator).second<<" ";
	}
	cout<<endl;

	//按键给定的区间显示序列中的元素
```

```
	cout<<"[theMap.lower_bound(3),theMap.upper_bound(8)] :"<<endl;
	for (it=theMap.lower_bound(3);it!=theMap.upper_bound(8);it++) {
		cout<<(*it).first;
		cout<<","<<(*it).second<<" ";
	}
	cout<<endl;

	//显示 theMap 的状态信息
	cout<<"theMap.size():"<<theMap.size()<<endl;
	cout<<"theMap.max_size():"<<theMap.max_size()<<endl;
	cout<<"theMap.count(15):"<<theMap.count(15)<<endl;

	// 从键盘上输入数字，显示对应的字符串
	string theString = "";
	int index;
	for( ; ; )
	{
		cout << "Enter \"q\" to quit, or enter a Number: ";
		cin >> theString;
		if(theString == "q")
			break;

		for(index = 0; index < theString.length(); index++){
			theIterator = theMap.find(theString[index] - '0');
			if(theIterator != theMap.end() )
				cout << (*theIterator).second << " ";
			else
				cout << "[err] ";
		}
		cout << endl;
	}
}
#include <iostream>
#include <string>
#include <map>

using namespace std;

//创建 multimap 的实例，整数(int)映射字符串(string)
typedef multimap<int, string> INT2STRING;

//测试 multimap 容器
void main()
{
	//创建 multimap 对象 theMap
	INT2STRING theMap;
	INT2STRING::iterator theIterator,it;

	//向 theMap 容器中添入数据，数字和字符串配对
	//每个元素是一个映射对
	theMap.insert(INT2STRING::value_type(90,"张卫"));
	theMap.insert(INT2STRING::value_type(85,"李华"));
	theMap.insert(INT2STRING::value_type(73,"赵明"));
	theMap.insert(INT2STRING::value_type(96,"郝名"));

	//显示 multimap 容器的所有对象
	cout<<"theMap.begin()--theMap.end():"<<endl;
	for (theIterator=theMap.begin();theIterator!=theMap.end();++theIterator){
		cout<<(*theIterator).second;
		cout<<"\t"<<(*theIterator).first<<endl;
```

```
    }

    //测试 multimap 容器 key 的非惟一性
    theMap.insert(INT2STRING::value_type(90,"李朋"));
    theMap.insert(INT2STRING::value_type(85,"钱德"));
    theMap.insert(INT2STRING::value_type(93,"赵刚"));

    //按成绩高低输出 multimap 容器的所有对象
    INT2STRING::reverse_iterator i;
    cout<<"theMap.rbegin()--theMap.rend():"<<endl;
    for (i=theMap.rbegin();i!=theMap.rend();++i){
        cout<<(*i).second;
        cout<<"\t"<<(*i).first<<endl;
    }

    //按关键给定的区间显示序列中的元素
    cout<<"[theMap.lower_bound(80),theMap.upper_bound(90)] :"<<endl;
    for (it=theMap.lower_bound(80);it!=theMap.upper_bound(90);it++) {
        cout<<(*it).second;
        cout<<"\t"<<(*it).first<<endl;
    }

    //显示 theMap 的状态信息
    cout<<"theMap.size():"<<theMap.size()<<endl;
    cout<<"theMap.max_size():"<<theMap.max_size()<<endl;
    cout<<"theMap.count(90):"<<theMap.count(90)<<endl;

    //清除 90 分以下的数据,并显示结果
    theMap.erase(theMap.lower_bound(60),theMap.upper_bound(89));
    cout<<"theMap.rbegin()--theMap.rend():"<<endl;
    for (i=theMap.rbegin();i!=theMap.rend();++i){
        cout<<(*i).second;
        cout<<"\t"<<(*i).first<<endl;
    }
}
#include <iostream>
#include <valarray>
#include <math.h>

using namespace std;

#define ARRAY_SIZE    3                    //array size

//测试 valarray 容器
void main()
{
    //创建具有 3 个元素的数组 val_array
    valarray<double> val_array(ARRAY_SIZE);

    //设置数组的值为 1, 4, 9
    for (int i = 0; i < ARRAY_SIZE; i++)
        val_array[i] = (i+1) * (i+1);

    //显示 val_array 数组的大小
    cout << "Size of val_array = " << val_array.size() << endl;

    //  显示 val_array 数组的值
    cout << "The values in val_array before calling sqrt() and pow():" << endl;
    for (i = 0; i < ARRAY_SIZE; i++)
        cout << val_array[i] << "       ";
```

```
    cout << endl;

    //声明一个 rev_valarray 数组，其保存对数组 val_array 的取反
    valarray<double> rev_valarray(ARRAY_SIZE);
    for (i = 0; i < ARRAY_SIZE; i++)
        rev_valarray[i] = val_array[ARRAY_SIZE - i - 1];

    //显示 rev_valarray 数组的大小和元素
    cout << "Size of rev_valarray = " << rev_valarray.size() << endl;
    cout << "The values in rev_valarray:" << endl;
    for (i = 0; i < ARRAY_SIZE; i++)
        cout << rev_valarray[i] << "        ";
    cout <<endl;

    //  声明 rvalue_array 数组，其存放调用 sqrt()和 pow()函数的返回值
    valarray<double> rvalue_array;

    //调用 sqrt()函数并显示结果
    rvalue_array = sqrt(val_array);
    cout << "The result of rvalue_array after calling sqrt():" << endl;
    for (i = 0; i < ARRAY_SIZE; i++)
        cout << rvalue_array[i] << "          ";
    cout <<endl;

    //对 val_array 数组元素计算幂函数并显示
    rvalue_array = pow(val_array, rev_valarray);
    cout << "The result after calling pow(val_array, rev_valarray):"
         << endl;
    for (i = 0; i < ARRAY_SIZE; i++)
        cout << rvalue_array[i] << "          ";
    cout <<endl;

    //对 val_array 数组元素计算幂函数，指数均为 2.0，并显示
    rvalue_array = pow(val_array, 2.0);
    cout << "The result after calling pow(val_array, 2.0):" << endl;
    for (i = 0; i < ARRAY_SIZE; i++)
        cout << rvalue_array[i] << "          ";
    cout <<endl;

    //对 2.0 进行幂函数运算，指数均为数组 val_array 的各元素值
    rvalue_array = pow(2.0, val_array);
    cout << "The result after calling pow(2.0, val_array):" << endl;
    for (i = 0; i < ARRAY_SIZE; i++)
        cout << rvalue_array[i] << "          ";
    cout <<endl;

    //对 val_array 和 rvalue_array 求和
    cout<<"val_array.sum()="<<val_array.sum()<<endl;
    cout<<"rvalue_array.sum()="<<rvalue_array.sum()<<endl;

    //求最大值并显示
    cout<<"val_array.max()="<<val_array.max()<<endl;
    cout<<"rvalue_array.max()="<<rvalue_array.max()<<endl;
}
#include <stack>
#include <iostream>

using namespace std ;

typedef stack<int> STACK_INT;
```

```
void main()
{
	STACK_INT stack1;
	int i;

	//判断栈是否空
	cout << "stack1.empty() returned " <<
		(stack1.empty()? "true": "false") << endl;

	//0,2,4,6...入栈
	for (i=0;i<10;i=i+2)
		stack1.push(i);

	//top()函数
	if (!stack1.empty())
		cout << "stack1.top() returned " <<stack1.top() << endl;

	//计算栈的长度
	cout<<"stack1.size(): "<<stack1.size()<<endl;

	//改变栈顶的值  20.
	if (!stack1.empty()) {
		cout << "stack1.top()=20;" << endl;
		stack1.top()=20;
	}

	//弹出栈中所有的数据并显示
	cout<<"stack1: ";
	while (!stack1.empty()) {
		cout<<stack1.top()<<" ";
		stack1.pop();
	}
	cout<<endl;
}
#include <iostream>
#include <list>
#include <numeric>

using namespace std;
//创建一个 list 容器的实例 LISTINT，其存放 int 型数据
typedef list<int> LISTINT;

void main(void)
{
	//用 LISTINT 创建一个名为 listOne 的 list 对象
	LISTINT listOne;
	//指定 i 为迭代器变量
	LISTINT::iterator i;
	LISTINT::reverse_iterator ir;

	//从前面向 listOne 容器中添加数据
	listOne.push_front (2);
	listOne.push_front (1);

	//从后面向 listOne 容器中添加数据
	listOne.push_back (3);
	listOne.push_back (4);

	//从前向后显示 listOne 中的数据
```

```
	for (i = listOne.begin(); i != listOne.end(); ++i)
		cout << *i << " ";
	cout << endl;

	//从后向后显示 listOne 中的数据
	for (ir =listOne.rbegin();ir!=listOne.rend(); ++ir)
		cout << *ir << " ";
	cout << endl;

	//从键盘上输入数据
	for (i = listOne.begin(); i != listOne.end(); ++i) {
		cout<<"listOne    :";
		cin>>(*i);
	}

	//从前向后显示 listOne 中的数据
	for (i = listOne.begin(); i != listOne.end(); ++i)
		cout << *i << " ";
	cout << endl;

	//bidirectional 迭代器不允许加减运算
	// i=listOne.begin()+1;
}
#include <iostream>
#include <iostream>
#include <numeric>
#include <vector>
#include <list>
#include <set>

using namespace std;

//利用类模板生成类实例
typedef vector < int > IntArray;
typedef list <int> LISTINT;
typedef set<int> SET_INT;
int add(int a, int b) {
    return a+b;
}
//在 main()函数中测试 accumulate 算法
void main ()
{
//-------------------------------------------
//    accumulate 算法对于普通数组的计算
//-------------------------------------------
	int x[]={1,3,5,7,9};

	cout<<"x[]:";
	for (int i=0;i<5;i++)
		cout<<x[i]<<" ";
	cout<<endl;
	cout<<"accumulate(x,x+5,0)=";
	cout<<accumulate(x,x+5,0)<<endl;
	int val=100;
	cout<<"val="<<val<<endl;
	cout<<"accumulate(x,x+5,val)=";
	cout<<accumulate(x,x+5,val)<<endl;
//-------------------------------------------
//    accumulate 算法对于 vector 容器的计算
//-------------------------------------------
```

```
        //声明 intvector 容器和迭代器 ii
        IntArray intvector;
        IntArray::iterator ii;

        //向 intvector 容器中插入元素
        for (i=1; i<=5; i++) {
            intvector.push_back(i);
        };

        //显示 intvector 容器中的元素值和累加结果
        cout << "intvector: "<<endl;
        for (ii=intvector.begin();ii !=intvector.end();++ii)
            cout<<(*ii)<<" ";
        cout<<endl;
        cout<<"accumulate(intvector.begin(),intvector.end(),0)=";
        cout<<accumulate(intvector.begin(),intvector.end(),0)<<endl;
//-------------------------------------------
//      accumulate 算法对于 list 容器的计算
//-------------------------------------------
        //声明 list 容器对象和迭代器
        LISTINT::iterator iL;
        LISTINT list1;

        //向 list1 容器对象中插入元素并显示
        list1.push_front(1);
        list1.push_front(3);
        list1.push_front(5);
        list1.push_back(2);
        list1.push_back(6);

        //显示 list1 容器的元素值和累加结果
        cout << "list1: "<<endl;
        for (iL=list1.begin();iL !=list1.end();++iL)
            cout<<(*iL)<<" ";
        cout<<endl;
        cout<<"accumulate(list1.begin(),list1.end(),0)=";
        cout<<accumulate(list1.begin(),list1.end(),0)<<endl;
//-------------------------------------------
//      accumulate 算法对于 set 容器的计算
//-------------------------------------------
        //声明 set 容器对象和迭代器
        SET_INT set1;
        SET_INT::iterator si;

        //向 set1 容器中插入元素
        set1.insert(5);
        set1.insert(20);
        set1.insert(10);
        set1.insert(15);
        set1.insert(25);

        //显示 set1 容器的元素值和累加结果
        cout <<"set1: "<<endl;
        for (si=set1.begin();si !=set1.end();++si)
            cout<<(*si)<<" ";
        cout<<endl;
        cout<<"accumulate(set1.begin(),set1.end(),0)=";
        cout<<accumulate(set1.begin(),set1.end(),0)<<endl;
        cout<<"accumulate(set1.begin(),set1.end(),100)=";
        cout<<accumulate(set1.begin(),set1.end(),100)<<endl;
```

```
}
#include <iostream>
#include <algorithm>
#include <vector>
#include <list>
#include <set>
#define size 10
using namespace std;

//产生指定范围的整数随机数
int getrand(int min,int max) {
	int m;
	m=(max-min);
	m=min+double(rand())/RAND_MAX*m ;
	return m;
}

//利用类模板生成实例
typedef vector < int > IntArray;
typedef list <int> LISTINT;
typedef set<int> SET_INT;

//在 main()函数中测试 accumulate 算法
void main ()
{
//-----------------------------------------
//	count 算法对于普通数组的计算
//-----------------------------------------
	int x[size];

	cout<<"x[]:";
	for (int i=0;i<size;i++) {
		x[i]=getrand(1,3);
		cout<<x[i]<<" ";
	}
	cout<<endl;
	cout<<"count(x,x+size,2)=";
	cout<<count(x,x+size,2)<<endl;
	cout<<"count(x+2,x+8,2)=";
	cout<<count(x+2,x+8,2)<<endl;
//-----------------------------------------
//	count 算法对于 vector 容器的计算
//-----------------------------------------
	//声明 intvector 容器和迭代器 ii
	IntArray intvector;
	IntArray::iterator ii;

	//向 intvector 容器中插入元素
	for (i=1; i<size; i++) {
		intvector.push_back(getrand(2,6));
	};
	//显示 intvector 容器中的元素值和统计结果
	cout << "intvector: ";
	for (ii=intvector.begin();ii !=intvector.end();++ii)
		cout<<(*ii)<<" ";
	cout<<endl;
	cout<<"count(intvector.begin(),intvector.end(),4)=";
	cout<<count(intvector.begin(),intvector.end(),4)<<endl;
//-----------------------------------------
//	count 算法对于 list 容器的计算
```

```
//-------------------------------------------
	//声明 list 容器对象和迭代器
	LISTINT::iterator iL;
	LISTINT list1;

	//向 list1 容器对象中插入元素并显示
	for (i=1; i<size; i++) {
		list1.push_front(getrand(3,5));
	};

	//显示 list1 容器的元素值和统计结果
	cout << "list1: ";
	for (iL=list1.begin();iL !=list1.end();++iL)
		cout<<(*iL)<<" ";
	cout<<endl;
	cout<<"count(list1.begin(),list1.end(),3)=";
	cout<<count(list1.begin(),list1.end(),3)<<endl;
//-------------------------------------------
//	count 算法对于 set 容器的计算
//-------------------------------------------
	//声明 set 容器对象和迭代器
	SET_INT set1;
	SET_INT::iterator si;

	//向 set1 容器中插入元素
	for (i=1; i<size; i++) {
	   set1.insert(getrand(1,10));
	};

	//显示 set1 容器的元素值和统计结果
	cout <<"set1: ";
	for (si=set1.begin();si !=set1.end();++si)
		cout<<(*si)<<" ";
	cout<<endl;
	cout<<"count(set1.begin(),set1.end(),5)=";
	cout<<count(set1.begin(),set1.end(),5)<<endl;
}
#include <iostream>
#include <algorithm>
#include <string>
#include <vector>

using namespace std;

//如果字符串以'S'开头，则返回 true
int MatchFirstChar( const string& str)
{
	string s("S") ;
	return s == str.substr(0,1) ;
}

//测试 count_if 算法
void main()
{
	const int VECTOR_SIZE = 8 ;

	//生成成员类型为 strings 的 vector 容器类
	typedef vector<string > StringVector ;

	//定义迭代器类型
```

```
    typedef StringVector::iterator StringVectorIt ;

    //声明 vector 容器的对象
    StringVector NamesVect(VECTOR_SIZE) ;

    //声明迭代器
    StringVectorIt start, end, it ;

    int result = 0 ;     //  存放统计数据

    //初始化 vector 容器 NamesVect
    NamesVect[0] = "She" ;
    NamesVect[1] = "Sells" ;
    NamesVect[2] = "Sea" ;
    NamesVect[3] = "Shells" ;
    NamesVect[4] = "by" ;
    NamesVect[5] = "the" ;
    NamesVect[6] = "Sea" ;
    NamesVect[7] = "Shore" ;

    //设置容器的起始位置和终止位置
    start = NamesVect.begin() ;
    end = NamesVect.end() ;

    //显示 NamesVect 容器的元素
    cout << "NamesVect: " ;
    for(it = start; it != end; it++)
        cout << *it << " " ;
    cout <<endl ;

    //统计并显示 NamesVect 容器的所有元素中以'S'字符开头的字符串
    result = count_if(start, end, MatchFirstChar) ;
    cout << "Number of elements that start with letter \"S\" = "
        << result << endl    ;

    //显示 NamesVect 容器[1,6]之间的元素
    cout <<"NamesVect[1]--NamesVect[6]: " ;
    for(it =&NamesVect[1]; it != &NamesVect[7]; it++)
        cout << *it << " " ;
    cout <<endl ;

    //统计并显示 NamesVect 容器的所有元素中以'S'字符开头的字符串
    result = count_if(&NamesVect[1], &NamesVect[7], MatchFirstChar) ;
    cout << "Number of elements that start with letter \"S\" = "
        << result << endl    ;
}
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;
//利用类模板生成实例
typedef vector < int > IntArray;

//显示数组
void put_array(int x[],int size) {
      for(int i=0;i<size;i++)
        cout<<x[i]<<" ";
    cout<<endl;
}
```

```
//显示 vector 容器中的元素
void put_vector(IntArray v)
{
	IntArray::iterator theIterator;

	for (theIterator=v.begin();theIterator!=v.end();++theIterator){
		cout<<(*theIterator)<<" ";
	}
	cout<<endl;
}

//在 main()函数中测试 fill 和 fill_n 算法
void main ()
{
//-------------------------------------------
//	fill 和 fill_n 算法对普通数组的计算
//-------------------------------------------
	int x[]={1,3,5,7,9};
	cout << "x[]: ";
	put_array(x,5);
	//填数处理
	fill(x+1,x+3,2);
	cout << "fill(x+1,x+3,2): "<<endl;
	put_array(x,5);
	fill_n(x,3,8);
	cout << "fill_n(x,3,8): "<<endl;
	put_array(x,5);
//-------------------------------------------
//	fill 和 fill_n 算法对于 vector 容器的计算
//-------------------------------------------
	//声明 intvector 容器和迭代器 ii
	IntArray intvector;

	//向 intvector 容器中插入元素
	for (int i=1; i<=10; i++) {
		intvector.push_back(i);
	};
	//显示 intvector 容器中的元素值和统计结果
	cout << "intvector: "<<endl;
	put_vector(intvector);
	//填数处理
	fill(intvector.begin(),intvector.begin()+3,2);
	cout << "fill(intvector.begin(),intvector.begin()+3,2): "<<endl;
	put_vector(intvector);
	fill_n(&intvector[5],3,8);
	cout << "fill_n(&intvector[5],3,8): "<<endl;
	put_vector(intvector);
}
#include <iostream>
#include <algorithm>
#include <vector>
#define ARRAY_SIZE 10
using namespace std;

//利用类模板生成实例
typedef vector < int > IntArray;

//显示数组
void put_array(int x[],int size) {
	for(int i=0;i<size;i++)
```

```
        cout<<x[i]<<" ";
    cout<<endl;
}

//显示 vector 容器中的元素
void put_vector(IntArray v)
{
    IntArray::iterator theIterator;

    for (theIterator=v.begin();theIterator!=v.end();++theIterator){
        cout<<(*theIterator)<<" ";
    }
    cout<<endl;
}

//在 main()函数中测试 find()算法
void main ()
{
    int i,value,*p;
//-------------------------------------------
//    find()算法对于普通数组的处理
//-------------------------------------------
    int x[ARRAY_SIZE]={1,3,5,7,9,2,4,6,8,10};
    cout << "x[]: ";
    put_array(x,ARRAY_SIZE);

    //find()算法查找,并显示查找结果
    for(i=0;i<=2;i++) {
        cout<<"value=";
        cin>>value;
        p=find(x,x+ARRAY_SIZE,value);

        if (p != x + ARRAY_SIZE)   {   //查到
            cout << "First element that matches " << value;
            cout<< " is at location " << p - x<< endl;
        }
        else   {                    //未查到
           cout << "The sequence does not contain any elements";
           cout<< " with value " << value << endl ;
        }

    }
//-------------------------------------------
//    find()算法对于 vector 容器的处理
//-------------------------------------------
    //声明 intvector 容器对象
    IntArray intvector;

    //向 intvector 容器中插入元素
    for (i=1; i<=10; i++) {
        intvector.push_back(i);
    };

    //显示 intvector 容器中的元素值
    cout << "intvector: ";
    put_vector(intvector);

    //find()算法查找,并显示查找结果
    IntArray::iterator pos;
```

```
        for (i=0;i<=2;i++) {
            cout<<"value=";
            cin>>value;
            pos=find(intvector.begin(),intvector.end(),value);
            if (pos != intvector.end())    {    //查到
                cout << "First element that matches " << value;
                cout<< " is at location " <<pos - intvector.begin()<< endl;
            }
            else    {                    //未查到
                cout << "The sequence does not contain any elements";
                cout<< " with value " << value << endl ;
            }
        }
}
#include <iostream>
#include <algorithm>
#include <vector>
#define ARRAY_SIZE 10
using namespace std;

//利用类模板生成实例
typedef vector < int > IntArray;

//显示数组
void put_array(int x[],int size) {
        for(int i=0;i<size;i++)
            cout<<x[i]<<" ";
}

//显示 vector 容器中的元素
void put_vector(IntArray v)
{
        IntArray::iterator theIterator;

        for (theIterator=v.begin();theIterator!=v.end();++theIterator){
            cout<<(*theIterator)<<" ";
        }
}

//在 main()函数中测试 find()_end()算法
void main ()
{
//-------------------------------------------
//      find_end()算法对普通数组的处理
//-------------------------------------------
        int x[ARRAY_SIZE]={1,3,5,7,9,2,4,6,8,10};
        cout << "x[]: ";
        put_array(x,ARRAY_SIZE);
        cout<<endl;
        int y[]={5,7,9};
        cout << "y[]: ";
        put_array(y,3);
        cout<<endl;

        // find_end()算法查找,并显示查找结果
        int *p=find_end(x,x+ARRAY_SIZE,&y[0],&y[2]);
        if (p != x + ARRAY_SIZE)    {    //查到
            cout << "The first element that matches :" ;
            put_array(y,3);
            cout<< " is at location in x" << p - x<< endl;
```

```
        }
        else    {                   //未查到
                cout << "The sequence does not contain any elements";
                cout<< " with value " ;
                put_array(&x[3],3);
        }

//-------------------------------------------
//      find_end()算法对 vector 容器的处理
//-------------------------------------------
     //声明 intvector 容器对象
        IntArray intvector;

        //向 intvector 容器中插入元素
        for (int i=1; i<=10; i++) {
              intvector.push_back(i);
        };

        //显示 intvector 容器中的元素值
        cout << "intvector: ";
        put_vector(intvector);
        cout<<endl;

        IntArray temp;
        temp.push_back(5);
        temp.push_back(6);
        temp.push_back(7);
        cout << "temp: ";
        put_vector(temp);
        cout<<endl;

        // find_end()算法查找,并显示查找结果
        IntArray::iterator pos;
        pos=find_end(intvector.begin(),intvector.end(),temp.begin(),temp.end());

        if (pos != intvector.end())    {    //查到
              cout << "The first element that matches ";
              put_vector(temp);
              cout<< " is at location in intvector " <<pos - intvector.begin()<< endl;
        }
        else    {                   //未查到
                cout << "The sequence does not contain any elements";
                cout<< " with value ";
              put_vector(temp);
              cout<< endl ;
        }
}
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

//返回一个 Fibonacci 数，其由 generate_n()算法调用
int Fibonacci1(void)
{
        static int r;
        static int f1 = 0;
        static int f2 = 1;
        r = f1 + f2 ;
```

```
        f1 = f2 ;
        f2 = r ;
        return f1 ;
}
//返回一个 Fibonacci 数，其由 generate()算法调用
int Fibonacci2(void)
{
        static int r;
        static int f1 = 0;
        static int f2 = 1;
        r = f1 + f2 ;
        f1 = f2 ;
        f2 = r ;
        return f1 ;
}
//定义整型数的 vector 容器类
        typedef vector<int > IntVector ;

//显示 vector 容器中的元素
void put_vector(IntVector v,char *name)
{
        IntVector::iterator theIterator;
        cout<<name<<":"<<endl;
        for (theIterator=v.begin();theIterator!=v.end();++theIterator){
                cout<<(*theIterator)<<" ";
        }
        cout<<endl;
}

//测试 generate()和 generate_n()算法
void main()
{
        const int VECTOR_SIZE = 15 ;

        //定义迭代器类
        typedef IntVector::iterator IntVectorIt ;

        //声明 vector 容器对象
        IntVector Numbers1(VECTOR_SIZE),Numbers2(VECTOR_SIZE);
        int i ;

        //初始化 vector 容器对象
        for(i = 0; i < VECTOR_SIZE; i++)
                Numbers1[i] = i ;

        //显示 vector 容器对象的元素
        cout << "Before calling generate_n:" << endl ;
        put_vector(Numbers1,"Numbers1");

        //利用 generate_n 算法用 Fibonacci  数填充 vector 容器
        generate_n(Numbers1.begin(), VECTOR_SIZE, Fibonacci1) ;

        //显示 vector 容器对象的元素
        cout << "After calling generate_n:" << endl ;
        put_vector(Numbers1,"Numbers1");

        //利用 generate 算法用 Fibonacci  数填充 vector 容器
        generate(Numbers2.begin(),Numbers2.end(), Fibonacci2) ;

        //显示 vector 容器对象的元素
```

```
	cout << "After calling generate:" << endl ;
	put_vector(Numbers2,"Numbers2");
}
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

//利用类模板生成实例
typedef vector < int > IntArray;

//显示数组
void put_array(int x[],int size) {
	for(int i=0;i<size;i++)
		cout<<x[i]<<" ";
	cout<<endl;
}

//显示 vector 容器中的元素
void put_vector(IntArray v)
{
	IntArray::iterator theIterator;

	for (theIterator=v.begin();theIterator!=v.end();++theIterator){
		cout<<(*theIterator)<<" ";
	}
	cout<<endl;
}

//在 main()函数中测试 reverse()和 reverse_copy()算法
void main ()
{
//---------------------------------------------
//	reverse()和 reverse_copy()算法对普通数组处理
//---------------------------------------------
	int x[]={1,3,5,7,9};
	cout<<"x[]:";
	put_array(x,5);

	//reverse()反转 x 数组并显示
	reverse(x,x+5);
	cout<<"x[]:";
	put_array(x,5);

	int y[]={2,4,6,8,10};
	cout<<"y[]:";
	put_array(y,5);

	//reverse_copy()反转 y 数组的部分元素并拷贝到 x 数组第 2 个元素位置
	reverse_copy(y+1,y+3,x+1);
	cout<<"x[]:";
	put_array(x,5);
	cout<<"y[]:";
	put_array(y,5);
//---------------------------------------------
//	reverse()和 reverse_copy()算法对 vector 容器的处理
//---------------------------------------------
	//声明 intvector 容器和迭代器 ii
	IntArray intvector;
```

```
	//向 intvector 容器中插入元素
	for (int i=1; i<=10; i++) {
		intvector.push_back(i);
	};

	//显示 intvector 容器中的元素值
	cout << "intvector: "<<endl;
	put_vector(intvector);

	//reverse()对于 vector 容器的处理
	reverse(intvector.begin(),intvector.end());
	cout << "intvector: "<<endl;
	put_vector(intvector);

	// reverse_copy 对于 vector 容器的处理
	IntArray temp(5);
	reverse_copy(intvector.begin()+2,intvector.begin()+7,temp.begin());
	cout << "temp: "<<endl;
	put_vector(temp);
}
#include <iostream>
#include <algorithm>
#include <vector>
#include <stdlib.h>
#define ARRAY_SIZE 15
using namespace std;

//定义整型数的 vector 容器类
typedef vector<int > IntVector ;

//显示数组
void put_array(int x[],int size) {
	for(int i=0;i<size;i++)
		cout<<x[i]<<" ";
	cout<<endl;
}

//显示 vector 容器中的元素
void put_vector(IntVector v,char *name)
{
	IntVector::iterator theIterator;
	cout<<name<<": ";
	for (theIterator=v.begin();theIterator!=v.end();++theIterator){
		cout<<(*theIterator)<<" ";
	}
	cout<<endl;
}

//产生指定范围的整数随机数
int getrand(int min,int max) {
	int m;
	m=(max-min);
	m=min+double(rand())/RAND_MAX*m ;
	return m;
}

//在 main()函数中测试 sort()和 partial_sort()算法
void main ()
{
```

```
	int i;
//-------------------------------------------
//	sort()和 partial_sort()算法对普通数组处理
//-------------------------------------------
	//sort()算法处理数组，并显示
	int x[ARRAY_SIZE];
	for (i=0;i<ARRAY_SIZE;i++) {
		x[i]=getrand(1,20);
	}
	cout<<"x[]:";
	put_array(x,ARRAY_SIZE);
	sort(x,x+ARRAY_SIZE);
	cout<<"sort(x,x+ARRAY_SIZE):"<<endl;
	put_array(x,ARRAY_SIZE);

	//partial_sort()算法对于数组进行处理
	int y[ARRAY_SIZE];
	for (i=0;i<ARRAY_SIZE;i++) {
		y[i]=getrand(1,30) ;
	}
	cout<<"y[]:";
	put_array(y,ARRAY_SIZE);
	partial_sort(y+2,y+7,y+ARRAY_SIZE);
	cout<<"partial_sort(y+2,y+7,y+ARRAY_SIZE):"<<endl;
	put_array(y,ARRAY_SIZE);
//-------------------------------------------
//	sort()和 partial_sort()算法对 vector 容器的处理
//-------------------------------------------
	IntVector Numbers1,Numbers2;
	for(i=0;i<15;i++) {
		Numbers1.push_back(getrand(1,30));
		Numbers2.push_back(getrand(1,30));
	}
	put_vector(Numbers1,"Numbers1");
	put_vector(Numbers2,"Numbers2");

	//sort()算法处理并显示
	sort(Numbers1.begin(),Numbers1.end());
	cout<<"After call sort():"<<endl;
	put_vector(Numbers1,"Numbers1");

	//partial_sort()算法处理并显示
	partial_sort(Numbers2.begin()+2,Numbers2.begin()+7,Numbers2.end());
	cout<<"After call partial_sort():"<<endl;
	put_vector(Numbers2,"Numbers2");
}
#include <iostream>
#include <algorithm>
#include <stdlib.h>
#include <time.h>

#define ARRAY_SIZE 15
using namespace std;

//显示数组
void put_array(int x[],int size) {
	for(int i=0;i<size;i++)
		cout<<x[i]<<" ";
	cout<<endl;
}
```

```
//产生指定范围的整数随机数
int getrand(int min,int max) {
	int m;
	m=(max-min);
	m=min+double(rand())/RAND_MAX*m ;
	return m;
}
//在 main()函数中测试 max_element()和  min_element()算法
void main ()
{
	//声明变量和数组
	int i;
	int x[ARRAY_SIZE];

	//用 1 到 100 的随机数初始化数组，并显示
	srand( (unsigned)time( NULL ) );
	for (i=0;i<ARRAY_SIZE;i++) {
		x[i]=getrand(1,100);
	}
	cout<<"x[]:";
	put_array(x,ARRAY_SIZE);

	//对数组 x 使用 max_element()算法，并显示
	int *pMax=max_element(x,x+ARRAY_SIZE);
	cout<<"pMax	="<<pMax<<endl;
	cout<<"Location="<<(pMax-x)<<endl;
	cout<<"*pMax	="<<(*pMax)<<endl;

	//对数组 x 使用 min_element()算法，并显示
	int *pMin=min_element(x,x+ARRAY_SIZE);
	cout<<"pMin	="<<pMin<<endl;
	cout<<"Location="<<(pMin-x)<<endl;
	cout<<"*pMin	="<<(*pMin)<<endl;
}
```