

◆-----实用掌中宝-----◆

涵盖352个C语言常用函数，提供了352个典型示例

C语言

常用函数速查手册

陈超 等编著

- ◎ 涵盖了C语言常用的21个函数库的最常用函数
- ◎ 从函数的功能、参数和返回值三方面讲解函数的使用
- ◎ 所有函数都给出了针对性的示例，加深读者的理解
- ◎ 所有的示例代码都给出了详细的注释和完整的讲解



化学工业出版社

◆-----实用掌中宝-----◆

C语言

常用函数速查手册



化学工业出版社

· 北京 ·

本书全面、系统地讲解了C语言相关的21个函数库,所涉及的函数多达352个。为了方便读者学习,每一个函数都依次对其作用、语法形式、参数、返回值进行了讲解。同时,每个函数都配有专门的例子,供读者参考学习。最后给出了本书所涉及C语言函数的索引,便于读者检索。

为了方便读者查找,所有函数都按照所在库进行分章讲解。这样既方便读者系统学习,也方便同类函数的对比和查找。本书所涉及的函数全面,适合所有想学习C语言的开发人员、爱好者和大中专院校学生使用。对于经常采用C语言进行开发的开发人员,更是一本不可多得的案头必备工具参考书。

图书在版编目(CIP)数据

C语言常用函数速查手册 / 陈超等编著. —北京:
化学工业出版社, 2010.6
(实用掌中宝)
ISBN 978-7-122-08136-0

I. C… II. 陈… III. C语言-程序设计-技术
手册 IV. TP312-62

中国版本图书馆CIP数据核字(2010)第056114号

责任编辑: 陈 静

装帧设计: 蓝色印象

责任校对: 宋 玮

出版发行: 化学工业出版社(北京市东城区青年湖南街13号 邮政编码100011)

印 装: 北京市彩桥印刷有限责任公司

880mm×1230mm 1/32 印张11¹/₂ 字数325千字

2010年6月北京第1版第1次印刷

购书咨询: 010-64518888(传真: 010-64519686) 售后服务: 010-64518899

网 址: <http://www.cip.com.cn>

凡购买本书,如有缺损质量问题,本社销售中心负责调换。

定价: 28.00元

版权所有 违者必究

前言

C语言作为一门基础语言，深受广大编程爱好者的喜爱。由于其语法简单，功能强大，很多高校都把C语言作为必修课程，而初学者往往选择它作为编程的入门语言。同时，各类计算机考试的考察编程语言也为C语言，如全国计算机等级考试、程序员考试等。

C语言语法简单，通过各类库函数，可以实现所有功能。由于C语言诞生较早，现在很少有厂商提供完善的IDE支持，造成学习C语言的障碍，尤其是了解C语言函数。编者在日常开发中，对此深有感触。为了方便大家学习，编者花费了半年时间整理C语言各类函数，完成此书。

一、本书特色

1. 精选 21 个函数库，352 个函数

虽然C语言语法简单，但包含众多的函数库和函数。其数量众多，但是常用的比较有限。笔者根据多年C语言使用经验，精选了21个函数库，352个函数进行讲解。这些函数覆盖C语言各个常见领域，基本满足读者的各项需要。

2. 讲解细致，重点明确

为了方便读者学习，本书对每个函数都进行详细讲解，依次对函数的作用、语法、参数、返回值进行仔细描述，让读者可以快速掌握每个函数的使用。同时，书上出现的代码进行相应的压缩，避免过多的代码对读者的理解和使用造成影响。

3. 示例讲解，便于实践

为了方便读者更深入理解，本书每个函数都配以一个示例，这些示例具有很强的代表性。读者通过这些示例，不仅可以更快了解函数的使用，还可以参照示例进行动手实践。

第 1 章 输入输出函数库: stdio.h

在程序设计中,常常需要进行输入输出等相应的操作。C 语言专门提供了一个针对输入输出操作的函数库 stdio.h,该文件库中存在的常用函数主要包括字符、字符串的读取和输出等输入输出操作函数。

1.1 从流中取字符函数: getc()

函数 getc() 用于从流中取字符,其原型如下。

```
int getc(FILE *stream);
```

【参数】 参数 *stream 为要从中读取字符的文件流。

【返回值】 该函数执行成功后,将返回所读取的字符。

【例 1-1】 下面的示例演示了 getc() 函数的使用,在程序中采用该函数从标准输入控制台中读取字符,代码如下。

```
#include <stdio.h> //加入标准输入输出库
void main( ){
    char ch;
    printf("Input a character:"); //输出提示信息
    ch = getc(stdin); //从标准输入控制台中读取字符
    printf("The character input was: '%c'\n",ch); //输出字符
}
```

运行上述程序,首先声明一个用于保存所取字符的变量;然后输出提示信息,接收从标准输入控制台按下的任意键,并将该字符输出到控制台。

1.2 从 stdin 流中读字符函数: getchar()

函数 `getchar()` 用于从标准输入控制台读取字符, `stdin` 为标准输入控制台。函数 `getchar()` 的原型如下。

```
int getchar(void);
```

【参数】 该函数没有参数。

【返回值】 该函数执行后, 将返回从控制台所读取的字符。

【例 1-2】 下面的示例演示了 `getchar()` 函数的使用, 使用该函数获取标准输入控制台中的字符, 直到按回车键结束, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
void main( ){
    int c;
    while ((c = getchar( )) != '\n')                //从控制台流中读取字符, 直
到按回车键结束
        printf("%c", c);                            //输出读取内容
}
```

运行上述程序后, 将采用 `getchar()` 函数获取当前控制台的输入, 直到按回车键结束, 并将其输出。

1.3 从控制台取字符(带回显)函数: getche()

函数 `getche()` 用于读取从控制台输入的字符, 并将输入的字符立即显示在控制台, 其原型如下。

```
int getche(void);
```

【参数】 该函数没有参数。

【返回值】 该函数从控制台读取字符, 并在控制台显示读取的字符。

【例 1-3】 下面的示例演示了 `getche()` 函数的使用, 从控制台读取并

回显字符, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <conio.h>           //加入控制台输入输出库
void main( ){
    char ch;
    printf("Input a character:"); //输出提示信息
    ch = getche( );           //读取字符
    printf("\nYou input a '%c'\n", ch); //输出字符
}
```

运行上述程序, 首先声明用于暂存从控制台所读取字符的变量 `ch`, 输出提示信息; 然后读取并显示从控制台所输入的字符到变量 `ch` 中; 最后再将其输出到控制台。

1.4 从流中取整数函数: `getw()`

函数 `getw()` 用于从流中取整数, 其原型如下。

```
int getw(FILE *stream);
```

【参数】 参数 `stream` 为需要取整数的流。

【返回值】 该函数返回从流中读取的整数。

【例 1-4】 下面的示例演示了 `getw()` 函数的使用, 采用该函数从流中取整数, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <stdlib.h>          //加入标准工具库
#define FNAME "test.$$$"     //声明文件名常量
void main( ){
    FILE *fp;
    int word;
    fp = fopen(FNAME, "wb");   //打开文件
    if (fp == NULL) {         //判断打开是否成功, 并输出提示信息
        printf("Error opening file %s\n", FNAME);
        exit(1);
    }
```

```

    }
    word = 94; //赋值
    putw(word, fp); //将值放进文件
    if (ferror(fp)) printf("Error writing to file\n");
    //判断是否出错
    else printf("Successful write\n");
    fclose(fp); //关闭文件
    fp = fopen(FNAME, "rb"); //打开文件
    if (fp == NULL) { //判断打开成功否
        printf("Error opening file %s\n", FNAME);
        exit(1);
    }
    word = getw(fp); //读取整数
    if (ferror(fp)) printf("Error reading file\n"); //判断读取成功否
    else printf("Successful read: word = %d\n", word);
    fclose(fp); //关闭文件
    unlink(FNAME); //删除文件
}

```

运行上述程序, 首先打开指定文件, 若打开失败则输出提示信息, 退出程序。若打开成功, 则将给定值写入到该文件中, 并根据写入结果判断写入是否成功, 关闭文件, 然后再次打开该文件, 若打开失败, 则输出提示信息, 退出程序; 若打开成功后, 则从该文件中读取整数, 并根据读取结果输出相应的提示信息, 最后关闭并删除该文件。

中端从读流中取出一个字符并返回 (int) 函数, 不返回则返回 0 (即 EOF)。

1.5 输出一个字符到指定流中函数: putc()

函数 putc() 用于输出一个字符到指定流中, 其原型如下。

```
int putc(int ch, FILE *stream);
```

【参数】 参数 ch 表示要输出字符的位置; 参数 stream 为要输出的流。

【返回值】 该函数输出字符到流中, 返回 true; 否则, 返回 NULL (0)。

【例 1-5】 下面的示例演示了 putc() 函数的使用, 将给定字符串输出

到控制台, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
void main( ){
    char msg[ ] = "Hello world\n"; //声明变量
    int i = 0;                 //声明变量
    while (msg[i])             //将字符数组中的所有字符输出到控制台
        putchar(msg[i++], stdout);
}
```

运行上述程序, 首先声明要输出的字符数组和字符数组下标, 然后依次输出整个字符数组的所有字符到控制台。

1.6 在 stdout 上输出字符函数: putchar()

函数 putchar() 用于将给定字符输出到控制台, stdout 为标准输出控制台, 其原型如下。

```
int putchar(int ch);
```

【参数】 参数 ch 为输出字符的 ASCII 码值。

【返回值】 该函数输出字符成功后, 将返回 true。

【例 1-6】 下面的示例演示了 putchar() 函数的使用, 采用该函数在控制台上输出一个边框, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#define LEFT_TOP 0xDA         //定义字符常量
#define RIGHT_TOP 0xBF        //定义字符常量
#define HORIZ 0xC4            //定义字符常量
#define VERT 0xB3             //定义字符常量
#define LEFT_BOT 0xC0         //定义字符常量
#define RIGHT_BOT 0xD9        //定义字符常量
void main( ){
    char i, j;
    putchar(LEFT_TOP);        //输出边框上部
    for (i=0; i<10; i++)      putchar(HORIZ);    //输出水平线
```

```

    putchar(RIGHT_TOP);           //输出右上部
    putchar('\n');                //输出换行
    for (i=0; i<4; i++)           //画中间部分线
    {
        putchar(VERT);           //输出垂直线
        for (j=0; j<10; j++)      putchar(' '); //输出空白
        putchar(VERT);           //输出垂直线
        putchar('\n');           //换行
    }
    putchar(LEFT_BOT);            //画尾部
    for (i=0; i<10; i++) putchar(HORIZ); //输出水平线
    putchar(RIGHT_BOT);          //输出右下框
    putchar('\n');               //换行
}

```

运行上述程序, 首先定义用于输出的边框常量, 然后依次在控制台输出边框的上部、边框的左右两边及边框的下部。

1.7 将字符串送到流中函数: puts()

函数 puts() 用于将字符串输出到缓冲区中, 其函数原型如下。

```
int puts(char *string);
```

【参数】 参数 string 为要输出的字符串。

【返回值】 若该函数将字符串输出到流中, 返回非零值; 否则, 返回 0。

【例 1-7】 下面的示例演示了 puts() 函数的使用, 采用该函数将指定的字符串输出到标准输出控制台, 代码如下。

```

#include <stdio.h>                //加入标准输入输出库
void main( ){
    char string[ ] = "This is an example output string\n";
                                   //声明变量
    puts(string);                //将字符串输出到流中
}

```

运行上述程序, 首先声明需要进行处理的字符串, 然后采用 puts()

函数将该字符串输出到标准输出控制台中。

1.8 从流中取字符函数: gets()

函数 gets() 用于从缓冲区中读取字符, 其原型如下。

```
char *gets(char *string);
```

【参数】 参数 string 为从给定文件流中所读取到的字符串。

【返回值】 该函数从流中取字符, 所取的字符暂存在所给的参数 string 中。

【例 1-8】 下面的示例演示了 gets() 函数的使用, 使用该函数从流中读取字符到指定变量中, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
void main( ){
    char string[80];                                //定义字符数组
    printf("Input a string:");                      //输出提示信息
    gets(string);                                    //从流中取字符
    printf("The string input was: %s\n", string);    //输出所取的字符
}
```

运行上述程序, 首先声明用于保存输入字符的字符数组; 然后从流中取所输入的字符, 并将所取字符暂存于字符数组中; 最后将该字符数组中的字符输出。

1.9 将字符或字送到流中函数: putw()

函数 putw() 用于将字符或字送到文件流中, 其原型如下。

```
int putw(int w, FILE *stream);
```

【参数】 参数 w 为要输出字符的 ASCII 码值; 参数 stream 为要输出

的文件流。

【返回值】该函数将字符或字送到文件流中，若操作成功返回 true；否则，返回 false。

【例 1-9】下面的示例演示了 putw() 函数的使用，采用该函数将字符送到文件流中，代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <stdlib.h>          //加入标准工具库
#define FNAME "test.$$$"     //声明文件名常量

void main( ){
    FILE *fp;
    int word;
    fp = fopen(FNAME, "wb");   //打开文件
    if (fp == NULL) {         //判断打开成功否，并输出提示信息
        printf("Error opening file %s\n", FNAME);
        exit(1);
    }
    word = 94;
    putw(word, fp);           //将字符输出到该文件中
    if (ferror(fp)) printf("Error writing to file\n");
    //判断输出成功否
    else printf("Successful write\n");
    fclose(fp);               //关闭文件
    fp = fopen(FNAME, "rb");   //重新打开文件
    if (fp == NULL) {         //判断打开成功否
        printf("Error opening file %s\n", FNAME);
        exit(1);
    }
    word = getw(fp);          //读取字符
    if (ferror(fp)) printf("Error reading file\n");
    //判断读取是否成功
    else printf("Successful read: word = %d\n", word);
    fclose(fp);               //关闭文件
    unlink(FNAME);            //删除文件
}
```

运行上述程序，首先定义文件常量，打开并写入字符到文件；然

后关闭文件,接着再次打开文件,从文件流中读取字符,并将其输出。

1.10 重命名文件函数: rename()

函数 rename() 用于重命名文件,其原型如下。

```
int rename(char *oldname, char *newname);
```

【参数】 参数 oldname 为旧文件名; 参数 newname 为新文件名。

【返回值】 若函数执行成功,将返回 0; 否则,返回 1。

【例 1-10】 下面的示例演示了 rename() 函数的使用,在程序中采用该函数重命名文件,代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
void main() {
    char oldname[80], newname[80];
    printf("File to rename: ");                    //输出提示信息
    gets(oldname);                                  //读取旧文件名
    printf("New name: ");                            //输出提示信息
    gets(newname);                                   //读取新文件名
    if (rename(oldname, newname) == 0) //重命名文件,并输出提示信息
        printf("Renamed %s to %s.\n", oldname, newname);
    else perror("rename");
}
```

运行上述程序,首先声明用于保存旧文件名和新文件名的字符数组,从流中读取新旧文件名;然后采用该函数重命名文件,并根据重命名结果输出提示信息。

1.11 删除文件函数: remove()

函数 remove() 用于删除指定文件,其原型如下。

```
int remove(char *filename);
```

【参数】参数 filename 为要删除的文件名。

【返回值】若该函数执行成功，将返回 0；否则，返回 1。

【例 1-11】下面的示例演示了 remove() 函数的使用，使用该函数删除指定文件，其代码如下。

```
#include <stdio.h>           //加入标准输入输出库
void main( ){
    char file[80];           //声明用于保存文件名的字符数组变量
    printf("File to delete: "); //输出提示信息
    gets(file);              //取文件名
    if (remove(file) == 0) printf("Removed %s.\n",file);
                               //删除文件，输出提示信息
    else    perror("remove");
}
```

运行上述程序，首先声明用于保存文件名的字符数组变量，从控制台获取文件名，然后删除该文件，并根据删除结果输出相应的提示信息。

1.12 将文件指针重新指向流的开头函数：

rewind()

函数 rewind() 用于将文件指针重新指向流的开头，其原型如下。

```
int rewind(FILE *stream);
```

【参数】参数 stream 为要操作的流。

【返回值】该函数将文件指针指向流的开头，若操作成功，返回非零值；否则，返回 0。

【例 1-12】下面的示例演示了 rewind() 函数的使用，采用该函数将文件指针重新指向流的开头，代码如下。


```

#include <stdio.h> //加入标准输入输出库
#include <dir.h>    //加入目录管理库

void main( ) {
    FILE *fp; //文件指针
    char *fname = "TXXXXXX", *newname, first; //声明文件名
    newname = mktemp(fname); //新建文件
    fp = fopen(newname, "w+"); //打开文件
    fprintf(fp, "abcdefghijklmnopqrstuvwxyz"); //写入内容
    rewind(fp); //将文件指针指向开头
    fscanf(fp, "%c", &first); //格式化输入
    printf("The first character is: %c\n", first); //输出第一个字符
    fclose(fp); //关闭文件
    remove(newname); //删除文件
}

```

运行上述程序, 首先声明需要处理的文件名, 打开该文件并写入内容, 然后将文件指针重新指向文件开头, 再在文件开头写入并输出在控制台输入的字符。

1.13 系统错误信息: perror()

函数 `perror()` 用于抛出特定的系统错误信息, 其原型如下。

```
void perror(char *string);
```

【参数】 参数 `string` 为要输出的错误信息。

【返回值】 该函数用于抛出系统错误, 没有返回值。

【例 1-13】 下面的示例演示了 `perror()` 函数的使用, 在程序中抛出指定的错误信息, 代码如下。

```

#include <stdio.h> //加入标准输入输出库

void main( ){
    FILE *fp;
    fp = fopen("perror.dat", "r"); //打开文件
    if (!fp) //判断打开结果

```

```

        perror("Unable to open file for reading");//抛出特定的系统错误
    信息
}

```

运行上述程序，首先打开文件，若打开失败，则抛出特定的系统错误信息。

1.14 把缓冲区与流相联函数：setbuf()

函数 setbuf() 用于将指定缓冲区与特定文件流相联，实现操作缓冲区时直接操作了文件流的功能，其原型如下。

```
void setbuf(FILE *stream, char *buf);
```

【参数】 参数 stream 为要处理的流；参数 buf 为要处理的缓冲区。

【返回值】 该函数没有返回值。

【例 1-14】 下面的示例演示了 setbuf() 函数的使用，采用该函数将缓冲区与流相联，代码如下。

```

#include <stdio.h>                                //加入标准输入输出库
char outbuf[BUFSIZ];
void main( ){
    setbuf(stdout, outbuf);                        //将缓冲区与流关联
    puts("This is a test of buffered output.\n\n"); //写字符到缓冲区
    puts("This output will go into outbuf\n");
    puts("and won't appear until the buffer\n");
    puts("fills up or we flush the stream.\n");
    fflush(stdout);                                //清除控制台
}

```

运行上述程序，首先采用 setbuf() 函数将缓冲区与标准输出控制台相联；然后采用 puts() 函数将字符输出到缓冲区，即直接输出到标准输出控制台；最后清除控制台。

1.15 把缓冲区与流相关函数: setvbuf()

函数 setvbuf() 用于将缓冲区与流相关, 其原型如下。

```
int setvbuf(FILE *stream, char *buf, int type, unsigned size);
```

【参数】 参数 stream 为要关联的流; 参数 buf 为要相关联的缓冲区; 参数 type 为关联的类型; 参数 size 为关联的大小。

【返回值】 若该函数关联成功, 返回非零值; 否则, 返回 0。

【例 1-15】 下面的示例演示了 setvbuf() 函数的使用, 采用该函数将缓冲区与流实现指定大小与类型的关联, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库

void main( ){
    FILE *input, *output;                          //声明变量
    char bufr[512];
    input = fopen("file.in", "r+b");               //打开文件
    output = fopen("file.out", "w");                //打开文件
    if (setvbuf(input, bufr, _IOFBF, 512) != 0) //关联并输出提示信息
        printf("failed to set up buffer for input file\n");
    else    printf("buffer set up for input file\n");
    if (setvbuf(output, NULL, _IOLBF, 132) != 0) //关联
        printf("failed to set up buffer for output file\n");
    else    printf("buffer set up for output file\n");
    fclose(input);                                  //关闭文件
    fclose(output);                                 //关闭文件
}
```

运行上述程序, 首先打开指定的文件, 将缓冲区与流按设定的类型与大小相关联, 然后关闭所打开的文件。

1.16 将格式化输出到字符串中函数: sprintf()

函数 sprintf() 用于将格式化输出到字符串中, 其原型如下。

```
int sprintf(char *string, char *format [,argument,...]);
```

【参数】参数 string 为要输出的字符串; 参数 format 为要输出的格式。

【返回值】该函数格式输出成功后, 返回非零值; 否则, 返回 0。

【例 1-16】下面的示例演示了 sprintf() 函数的使用, 采用该函数格式化字符串输出到指定的字符串中, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <math.h> //加入数学函数库
void main( ){
    char buffer[80]; //定义字符数组
    sprintf(buffer, "An approximation of Pi is %f\n", M_PI);
    //格式化输出到字符串
    puts(buffer); //输出字符数组
}
```

运行上述程序, 首先定义一个用于接收输出格式化字符的字符数组, 然后将指定字符进行格式输出到该数组中, 再将该字符数组进行输出。

1.17 执行字符串中的格式化输入函数: sscanf()

函数 sscanf() 用于执行字符串中格式化输入, 其原型如下。

```
int sscanf(char *string, char *format[,argument,...]);
```

【参数】参数 string 为保存所输入的字符串; 参数 format 为输入的格式。

【返回值】若该函数执行成功, 返回非零值; 否则, 返回 0。

【例 1-17】下面的示例演示了 sscanf() 函数的使用, 采用该函数格式化输入特定的值, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <conio.h> //加入控制台输入输出库
void main( ){
```



```

char label[20]; //定义字符数组
char name[20];
int entries = 0;
int loop, age;
double salary;
struct Entry_struct { //定义结构体
    char name[20];
    int age;
    float salary;
} entry[20];
printf("\n\nPlease enter a label for the chart:");//输出提示信息
sscanf("%20s", label); //输入字符
fflush(stdin); //清除标准输入控制台
printf("How many entries will there be? (less than 20) ");
//输出提示信息
sscanf("%d", &entries); //输入字符
fflush(stdin); //清除标准输入控制台
for (loop=0;loop<entries;++loop) { //循环输入
    printf("Entry %d\n", loop); //输出提示信息
    printf(" Name : ");
    sscanf("%[A-Za-z]", entry[loop].name); //输入姓名
    fflush(stdin); //清除标准输入控制台
    printf(" Age : "); //输出提示信息
    sscanf("%d", &entry[loop].age); //输入年龄
    fflush(stdin); //清除标准输入控制台
    printf(" Salary : "); //输出提示信息
    sscanf("%f", &entry[loop].salary); //输入薪水
    fflush(stdin); //清除标准输入控制台
}
printf("\nPlease enter your name, age and salary\n");
sscanf("%20s %d %lf", name, &age, &salary);//输入姓名、年龄、薪水
printf("\n\nTable %s\n",label);
printf("Compiled by %s age %d $%15.2lf\n", name, age, salary);
printf("-----\n");
for (loop=0;loop<entries;++loop) //循环输出
    printf("%4d | %-20s | %5d | %15.2lf\n", loop + 1,
entry[loop].name,

```

```

    entry[loop].age,
    entry[loop].salary);
    printf("-----\n");
}

```

运行上述程序，首先定义字符数组与结构体，循环格式化输入姓名、年龄、薪水，并保存在结构体中，然后再以表格的形式循环输出结构体中的姓名、年龄和薪水。

1.18 以二进制方式打开暂存文件函数: tmpfile()

函数 tmpfile() 用于以二进制方式打开暂存文件，其原型如下。

```
FILE *tmpfile(void);
```

【参数】 该函数没有参数。

【返回值】 该函数以二进制方式打开暂存文件后，返回该文件的句柄。

【例 1-18】 下面的示例演示了 tmpfile() 函数的使用，采用该函数以二进制方式打开暂存文件，代码如下。

```

#include <stdio.h>                                //加入标准输入输出库
#include <process.h>                                //加入进程管理库
void main( ){
    FILE *tempfp;                                  //定义文件变量
    tempfp = tmpfile( );                           //打开文件
    if (tempfp)                                     //根据结果输出信息
        printf("Temporary file created\n");
    else {
        printf("Unable to create temporary file\n");
        exit(1);
    }
}

```

运行上述程序，首先声明文件变量，再采用该函数以二进制方式打开暂存文件，并根据打开结果输出相应的提示信息。

1.19 创建一个唯一的文件名函数: tmpnam()

函数 tmpnam() 用于创建一个唯一的文件名, 其原型如下。

```
char *tmpnam(char *sptr);
```

【参数】 参数 sptr 为所创建的文件名。

【返回值】 若该函数执行成功, 将返回 0; 否则, 返回 1。

【例 1-19】 下面的示例演示了 tmpnam() 函数的使用, 在程序中采用该函数创建一个唯一的文件名, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
void main( ){
    char name[13];
    tmpnam(name);                                    //创建文件名
    printf("Temporary name: %s\n", name);          //输出
}
```

运行上述程序, 首先定义一个字符类型的数组, 采用 tmpnam() 函数创建一个唯一的文件名并保存在定义的字符数组中, 再将该文件名输出到控制台。

1.20 把字符退回到输入流函数: ungetc()

函数 ungetc() 用于将字符退回到输入流中, 其原型如下。

```
int ungetc(char c, FILE *stream);
```

【参数】 参数 c 为要退回的字符; 参数 stream 为要退出的输入流。

【返回值】 若该函数执行成功, 返回非零值; 否则, 返回 0。

【例 1-20】 下面的示例演示了 ungetc() 函数的使用, 使用该函数将字符退回到输入流中, 其代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
```

```

#include <ctype.h> //加入字符函数库
int main( void ){
    int i=0;
    char ch;
    puts("Input an integer followed by a char:");//输出提示信息
    while((ch = getchar( )) != EOF && isdigit(ch))
        //读取字符,直到以数字结束
        i = 10 * i + ch - 48; //转换为 ASCII 码值
    if (ch != EOF) //将字符退回流中
        ungetc(ch, stdin);
    printf("i = %d, next char in buffer = %c\n", i, getchar( ));
    //输出流中所退回的字符
}

```

运行上述程序,首先输出提示信息,从控制台读取字符并转换为 ASCII 码值,直到以数字结束,然后将字符退回到控制台,并输出所退回的字符。

1.21 把字符退回到键盘缓冲区函数: ungetch()

函数 ungetch ()用于把字符退回到键盘缓冲区中,其原型如下。

```
int ungetch(int c);
```

【参数】 参数 c 表示要退回的字符。

【返回值】 若退回成功,返回非零值;否则,返回 0。

【例 1-21】 下面的示例演示了 ungetch()函数的使用,采用该函数将指定字符退回到键盘缓冲区中,代码如下。

```

#include <stdio.h> //加入标准输入输出库
#include <ctype.h> //加入字符函数库
#include <conio.h> //加入控制台输入输出库
void main( ){
    int i=0;
    char ch;

```



```

puts("Input an integer followed by a char:");//输出提示信息
while((ch = getche( )) != EOF && isdigit(ch))
    //输入字符,直到以数字结束
    i = 10 * i + ch - 48; //转换为ASCII码值
if (ch != EOF) //如果没有数字,则将字符退回到键盘缓冲区
    ungetch(ch);
printf("\n\ni = %d, next char in buffer = %c\n", i, getch( ));
//输出所输入的字符
}

```

运行上述程序,首先在控制台输出提示信息,然后输入字符,直到输入数字结束。若没有结束,则将字符退回到键盘缓冲区,并将这些字符输出到控制台。

1.22 取得当前文件的句柄函数: fgetpos()

函数 fgetpos() 用于获取当前文件的句柄,其原型如下。

```
int fgetpos(FILE *stream);
```

【参数】 参数 stream 为文件流。

【返回值】 若该函数取得当前文件句柄,返回非零值;否则,返回 0。

【例 1-22】 下面的示例演示了 fgetpos() 函数的使用,采用该函数获取当前文件的句柄,代码如下。

```

#include <string.h> //加入字符串库
#include <stdio.h> //加入标准输入输出库
void main( ){
    FILE *stream;
    char string[ ] = "This is a test"; //定义要写入的变量
    fpos_t filepos;
    stream = fopen("DUMMY.FIL", "w+"); //打开文件
    fwrite(string, strlen(string), 1, stream); //写入字符
    fgetpos(stream, &filepos); //定位流上的文件指针
    printf("The file pointer is at byte %ld\n", filepos);
}

```

```
//输出文件指针
fclose(stream);           //关闭文件
}
```

运行上述程序，首先声明用于写入文件的字符数组；然后打开指定文件，将字符数组中的字符写入到该文件中，获取并输出当前文件的句柄；最后关闭打开的文件。

1.23 定位流上的文件指针函数：fsetpos()

函数 fsetpos() 用于定位流上的文件指针，其原型如下。

```
int fsetpos(FILE *stream, const fpos_t *pos);
```

【参数】参数 stream 为要处理的流；参数 pos 为文件指针位置。

【返回值】若该函数执行成功，则返回 true；否则，返回 NULL (0)。

【例 1-23】下面的示例演示了 fsetpos() 函数的使用，采用该函数定位流上的文件指针，其代码如下。

```
#include <stdlib.h>           //加入标准工具库
#include <stdio.h>           //加入标准输入输出库
void showpos(FILE *stream);   //声明自定义变量
void main( ){
    FILE *stream;
    fpos_t filepos;
    stream = fopen("DUMMY.FIL", "w+"); //打开文件
    fgetpos(stream, &filepos);         //保存文件指针位置
    fprintf(stream, "This is a test"); //向文件写入字符串
    showpos(stream);                   //显示当前位置
    if (fsetpos(stream, &filepos) == 0) //设置并显示新位置
        showpos(stream);
    else {
        fprintf(stderr, "Error setting file \
        pointer.\n"); //失败，输出提示信息
        exit(1);
    }
}
```



```

    }
    fclose(stream);           //关闭文件
}
void showpos(FILE *stream){
    fpos_t pos;
    fgetpos(stream, &pos);    //获取文件流中文件指针
    printf("File position: %ld\n", pos); //输出
}

```

运行上述程序,首先声明用于获取文件流中文件指针的自定义函数,然后打开指定文件并写入字符串,显示当前指针位置;接着设置新位置,若失败,则输出提示信息。

1.24 打开流函数: fopen()

函数 `fopen()` 用于打开流,其原型如下。

```
FILE *fopen(char *filename, char *type);
```

【参数】 参数 `filename` 为要打开的文件;参数 `type` 为打开的类型。

【返回值】 该函数以特定类型打开文件后,返回文件流。

【例 1-24】 下面的示例演示了 `fopen()` 函数的使用,采用该函数打开指定的文件,代码如下。

```

#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    FILE *fp;
    char buf[11] = "0123456789"; //声明要写入的字符数组
    fp = fopen("DUMMY.FIL", "w"); //打开指定的文件
    fwrite(&buf, strlen(buf), 1, fp); //写入字符
    fclose(fp);               //关闭打开的文件
}

```

运行上述程序,首先声明要写入文件的字符数组,然后打开一个指定的文件,写入字符串,最后采用 `fclose()` 函数将打开的文件关闭。

1.25 关闭流函数: fclose()

函数 fclose() 用于关闭文件流, 其原型如下。

```
int fclose(FILE *stream);
```

【参数】 参数 stream 为要关闭的文件流。

【返回值】 若该函数关闭成功, 返回非零值; 否则, 返回 0。

【例 1-25】 下面的示例演示了 fclose() 函数的使用, 采用该函数关闭一个打开的文件, 其代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    FILE *fp;
    char buf[11] = "0123456789"; //声明要写入文件字符数组
    fp = fopen("DUMMY.FIL", "w"); //打开文件
    fwrite(&buf, strlen(buf), 1, fp); //写入文件
    fclose(fp);                //关闭打开的文件
}
```

运行上述程序, 首先声明要写入文件的字符数组; 然后打开一个指定的文件, 写入字符串; 最后采用 fclose() 函数将打开的文件关闭。

1.26 清除流函数: fflush()

函数 fflush() 用于清除流, 其原型如下。

```
int fflush(FILE *stream);
```

【参数】 参数 stream 为要清除的文件流。

【返回值】 若该函数清除流成功, 返回非零值; 否则, 返回 0。

【例 1-26】 下面的示例演示了 fflush() 函数的使用, 采用该函数清除流, 其代码如下。


```

#include <string.h>           //加入字符串库
#include <stdio.h>            //加入标准输入输出库
#include <conio.h>            //加入控制台输入输出库
#include <io.h>               //加入输入输出库
void flush(FILE *stream);    //声明自定义变量
void main( ){
    FILE *stream;
    char msg[ ] = "This is a test";
    stream = fopen("DUMMY.FIL", "w"); //打开文件
    fwrite(msg, strlen(msg), 1, stream); //向文件中写入字符
    clrscr( );                     //清屏
    printf("Press any key to flush\
DUMMY.FIL:");                     //输出提示信息
    getch( );                      //输入字符
    flush(stream);                 //清除流
    printf("\nFile was flushed, Press any key\
to quit:");                       //输出提示信息
    getch( );                     //获取字符
}
void flush(FILE *stream){
    int duphandle;
    fflush(stream);               //清除流
    duphandle = dup(fileno(stream)); //复制文件句柄
    close(duphandle);             //关闭复制的文件句柄
}

```

运行上述程序, 首先声明用于清除流的自定义变量, 然后打开文件并写入字符串, 再采用自定义函数清除文件流。

1.27 检测流上的错误函数: ferror()

函数 ferror() 用于检测流上的错误, 其原型如下。

```
int ferror(FILE *stream);
```

【参数】 参数 stream 为要检测的流。

【返回值】若该函数检测到错误，返回非零值；否则，返回 0。

【例 1-27】下面的示例演示了 `ferror()` 函数的使用，采用该函数判断当前流是否有误，代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
void main( ){
    FILE *stream;
    stream = fopen("DUMMY.FIL", "w");              //打开文件
    (void) getc(stream);                            //强行读取流而产生错误
    if (ferror(stream)) {                          //检测当前流上的错误，并输出提示信息
        printf("Error reading from DUMMY.FIL\n");
        clearerr(stream);                          //清除错误
    }
    fclose(stream);                                //关闭文件
}
```

运行上述程序，首先打开一个文件，然后强行读取流而产生错误，再采用 `ferror()` 函数检测当前流上的错误，并输出提示信息，最后关闭文件。

1.28 替换流函数：freopen()

函数 `freopen()` 用于替换流，其原型如下。

```
FILE *freopen(char *filename, char *type, FILE *stream);
```

【参数】参数 `filename` 为要打开的文件；参数 `type` 为打开的类型；参数 `stream` 为文件指针。

【返回值】若函数执行成功，将返回非零值；否则，返回 `NULL (0)`。

【例 1-28】下面的示例演示了 `freopen()` 函数的使用，在程序中将输出重定向到文件，代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
void main( ){
    if (freopen("OUTPUT.FIL", "w", stdout) //将标准输出重定向到文件
```



```

    == NULL)
    fprintf(stderr, "error redirecting\stdout\n");//输出错误信息
    printf("This will go into a file."); //这里输出将重定向到文件
    fclose(stdout); //关闭流
}

```

运行上述程序, 首先采用 `freopen()` 函数将标准输出重定向到某个文件, 若出现错误将输出提示信息; 然后输出一些字符串, 这些字符串将全部输出到所定位的文件; 最后关闭打开的文件。

1.29 复位错误标志函数: `clearerr()`

函数 `clearerr()` 用于复位错误标志, 其原型如下。

```
void clearerr(FILE *stream);
```

【参数】 参数 `stream` 为出错的流。

【返回值】 该函数没有返回值。

【例 1-29】 下面的示例演示了 `clearerr()` 函数的使用, 使用该函数复位流中的错误标志, 其代码如下。

```

#include <stdio.h> //加入标准输入输出库
void main( ){
    FILE *fp;
    char ch;
    fp = fopen("DUMMY.FIL", "w"); //打开文件
    ch = fgetc(fp); //尝试读取流强行产生一个错误
    printf("%c\n", ch);
    if (ferror(fp)) { //检测错误
        printf("Error reading from DUMMY.FIL\n"); //输出提示信息
        clearerr(fp); //复位错误标志
    }
    fclose(fp); //关闭文件
}

```

运行上述程序, 首先打开指定文件, 尝试从流读取强行产生一个

错误，然后检测流中的错误，输出提示信息，并复位错误标志；最后关闭打开的文件。

1.30 从流中读取字符函数：fgetc()

函数 fgetc() 用于从流中读取字符，其原型如下。

```
int fgetc(FILE *stream);
```

【参数】 参数 stream 为要读取的流。

【返回值】 该函数返回所读取字符的 ASCII 码值。

【例 1-30】 下面的示例演示了 fgetc() 函数的使用，采用该函数从流中读取一个字符，代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
#include <conio.h>           //加入控制台输入输出库

void main() {
    FILE *stream;
    char string[] = "This is a test";
    char ch;
    stream = fopen("DUMMY.FIL", "w+"); //打开文件
    fwrite(string, strlen(string), 1, stream); //写字符串到文件
    fseek(stream, 0, SEEK_SET); //定位到文件头
    do {
        ch = fgetc(stream); //从文件读取一字符
        putchar(ch); //输出字符
    } while (ch != EOF);
    fclose(stream); //关闭文件
}
```

运行上述程序，首先定义要写入文件的字符串，打开指定的文件并写入字符串；然后将文件指针移动到文件头，逐个从文件读取字符并输出，直到文件结束停止读取；最后关闭文件。

1.31 从流中读取字符函数: fgetchar()

函数 fgetchar() 用于从流中读取字符, 其原型如下。

```
int fgetchar(void);
```

【参数】该函数没有参数。

【返回值】该函数从流中读取并返回字符。

【例 1-31】下面的示例演示了 fgetchar() 函数的使用, 采用该函数从流中读取字符, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
void main( ){
    char ch;
    printf("Enter a character followed by \<Enter>: ");
                                                //输出提示信息
    ch = fgetchar( );                                //从控制台读取字符
    printf("The character read is: '%c'\n", ch); //输出读取的字符
}
```

运行上述程序, 首先输出提示信息, 然后从流中读取字符, 并将所读取到的字符输出到控制台。

1.32 输出字符到标准输出流 (stdout) 函数:

fputchar()

函数 fputchar() 用于将字符输出到标准输出流, 其原型如下。

```
int fputchar(char ch);
```

【参数】参数 ch 为要输出的字符。

【返回值】若该函数执行成功, 则返回 true; 否则, 返回 NULL (0)。

【例 1-32】下面的示例演示了 fputchar() 函数的使用, 采用该函数将

字符送到标准输出流中，代码如下。

```
#include <stdio.h> //加入标准输入输出库
void main( ){
    char msg[ ] = "This is a test"; //定义字符数组
    int i = 0;
    while (msg[i]) {
        fputc(msg[i]); //输出字符到控制台
        i++;
    }
}
```

运行上述程序，首先定义用于输出的字符数组，然后再采用 `fputc()` 函数逐个输出字符数组中的每一个字符到标准输出控制台。

1.33 从流中读取字符函数：fgets()

函数 `fgets()` 用于从流中读取字符串，其原型如下。

```
char *fgets(char *string, int n, FILE *stream);
```

【参数】 参数 `string` 为所读取到的字符串；参数 `n` 为要读取的字符串长度；参数 `stream` 为要读取的流。

【返回值】 该函数没有返回值。

【例 1-33】 下面的示例演示了 `fgets()` 函数的使用，采用该函数从流中读取字符串，代码如下。

```
#include <string.h> //加入字符串库
#include <stdio.h> //加入标准输入输出库
void main( ){
    FILE *stream;
    char string[ ] = "This is a test"; //定义字符串
    char msg[20];
    stream = fopen("DUMMY.FIL", "w+"); //打开文件
    fwrite(string, strlen(string), 1, stream); //写入字符串
    fseek(stream, 0, SEEK_SET); //将文件指针定位到文件头
}
```

```

    fgets(msg, strlen(string)+1, stream); //从文件读取字符串
    printf("%s", msg);                    //输出字符串
    fclose(stream);                       //关闭文件
}

```

运行上述程序, 首先定义要写入文件的字符串; 然后打开指定文件, 写入字符串, 将文件指针定位到文件头; 再从文件流读取并输出字符串。

1.34 检测流上的文件结束符函数: feof()

函数 feof() 用于检测流上的文件结束符, 其原型如下。

```
int feof(FILE *stream);
```

【参数】 参数 stream 为需要检测的流。

【返回值】 若检测成功, 返回非零值; 否则, 返回 0。

【例 1-34】 下面的示例演示了 feof() 函数的使用, 采用该函数检测流上的文件是否结束, 代码如下。

```

#include <stdio.h> //加入标准输入输出库

void main( ){
    FILE *stream;
    stream = fopen("DUMMY.FIL", "r"); //打开文件
    fgetc(stream); //读取字符
    if (feof(stream)) printf("We have reached end-of-file\n");
    //检测文件是否结束
    fclose(stream); //关闭文件
}

```

运行上述程序, 首先打开指定的文件, 然后从该文件读取字符, 检测文件是否结束。若结束, 则输出提示信息, 最后关闭所打开的文件。

1.35 送字符串到流中函数: fputs()

函数 fputs() 用于送字符串到流中, 其原型如下。

```
int fputs(char *string, FILE *stream);
```

【参数】 参数 string 为需要输出的字符串; 参数 stream 为需要输出的流。

【返回值】 若输出成功, 返回非零值; 否则, 返回 0。

【例 1-35】 下面的示例演示了 fputs() 函数的使用, 采用该函数将一字符串输出到标准输出控制台, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
void main( ){
    fputs("Hello world\n", stdout);                //输出字符到标准输出控制台
}
```

运行上述程序, 直接将给定的字符串输出到标准输出控制台, 输出结果如下。

```
Hello world
```

1.36 从流中读数据函数: fread()

函数 fread() 用于从流中读取数据, 其原型如下。

```
int fread(void *ptr, int size, int nitems, FILE *stream);
```

【参数】 参数 ptr 为暂存从流中读取到的数据; 参数 size 为要读取的大小; 参数 nitems 为开始读取的位置; 参数 stream 为要读取的数据流。

【返回值】 若读取成功, 返回非零值; 否则, 返回 0。

【例 1-36】 下面的示例演示了 fread() 函数的使用, 采用该函数从流中读取数据, 代码如下。

```
#include <string.h>                                //加入字符串库
```

```

#include <stdio.h>                                //加入标准输入输出库
void main( ){
    FILE *stream;
    char msg[ ] = "this is a test";              //定义字符数组
    char buf[20];
    if ((stream = fopen("DUMMY.FIL", "w+")) == NULL) { //打开文件
        fprintf(stderr, "Cannot open output file.\n"); //输出提示信息
        return 1;
    }
    fwrite(msg, strlen(msg)+1, 1, stream); //写入字符串
    fseek(stream, SEEK_SET, 0);              //设置文件指针到文件头
    fread(buf, strlen(msg)+1, 1, stream); //读取数据
    printf("%s\n", buf);                      //输出
    fclose(stream);                          //关闭
}

```

运行上述程序, 首先定义要写入文件的字符串, 然后打开文件, 若打开失败则输出提示信息, 向该文件写入字符串, 并将文件指针定位到文件头, 再从文件流中读取并输出数据, 最后关闭文件。

1.37 写内容到流函数: fwrite()

函数 fwrite() 用于写内容到流, 其原型如下。

```
int fwrite(void *ptr, int size, int nitems, FILE *stream);
```

【参数】 参数 ptr 为要写入的内容; 参数 size 为要写入的大小; 参数 nitems 为写入的位置; 参数 stream 为要写入的流。

【返回值】 若写入成功, 返回非零值; 否则, 返回 0。

【例 1-37】 下面的示例演示了 fwrite() 函数的使用, 采用该函数将数据写入到流, 其代码如下。

```

#include <stdio.h>                                //加入标准输入输出库
struct mystruct {                                //定义结构体
    int i;

```

```

    char ch;
};

void main( ){
    FILE *stream;
    struct mystruct s;
    if ((stream = fopen("TEST.$$$", "wb")) == NULL) { //打开文件
        fprintf(stderr, "Cannot open output file.\n"); //输出提示信息
        return 1;
    }
    s.i = 0;
    s.ch = 'A';
    fwrite(&s, sizeof(s), 1, stream); //将结构体写入到文件
    fclose(stream); //关闭文件
}

```

运行上述程序，首先定义一个结构体，然后打开文件，给结构体赋值，再将该结构体写入到文件中，最后关闭打开的文件。

1.38 将格式化内容输出到流函数：fprintf()

函数 fprintf() 用于将格式化内容输出到流，其原型如下。

```
int fprintf(FILE *stream, char *format[, argument,...]);
```

【参数】 参数 stream 为要输出的流；参数 format 为要输出的格式。

【返回值】 若输出成功，返回非零值；否则，返回 0。

【例 1-38】 下面的示例演示了 fprintf() 函数的使用，使用该函数将给定内容格式化输出到流，代码如下。

```

#include <stdio.h> //加入标准输入输出库

void main( ){
    FILE *in, *out;
    if ((in = fopen("\\AUTOEXEC.BAT", "rt")) == NULL) { //打开文件
        fprintf(stderr, "Cannot open input \
file.\n"); //格式化输出错误信息
    }
}

```



```

        return 1;
    }
    if ((out = fopen("\\AUTOEXEC.BAK", "wt")) == NULL){ //打开文件
        fprintf(stderr, "Cannot open output \\file.\n");
                                                    //格式化输出错误信息

        return 1;
    }
    while (!feof(in))    fputc(fgetc(in), out); //复制文件
    fclose(in);          //关闭文件
    fclose(out);         //关闭文件
}

```

运行上述程序，首先定义两个用于读取文件和写入文件的文件流，然后分别打开这两个文件，再逐个读取文件，实现文件的复制，最后关闭两个文件。

1.39 从流中执行格式化输入内容函数: fscanf()

函数 `fscanf()` 用于从流中执行格式化输入内容，其原型如下。

```
int fscanf(FILE *stream, char *format[, argument, ...]);
```

【参数】 参数 `stream` 为要输入的流；参数 `format` 为指定的格式。

【返回值】 若该函数执行成功，返回非零值；否则，返回 0。

【例 1-39】 下面的示例演示了 `fscanf()` 函数的使用，采用该函数从标准输入流中读取指定格式的数据，代码如下。

```

#include <stdlib.h> //输入标准工具库
#include <stdio.h>  //加入标准输入输出库
void main( ){
    int i;
    printf("Input an integer: "); //输出提示信息
    if (fscanf(stdin, "%d", &i)) //从标准输入控制台输入整数
        printf("The integer read was: %i\n", i); //输出所输入的整数
}

```

```

else {
    fprintf(stderr, "Error reading an \integer from stdin.\n");
    //输出错误信息

    exit(1);
}
}

```

运行上述程序,首先输出提示信息,然后从标准输入流输入整数。若执行成功,则输出所输入的整数;若失败,则输出错误信息。

1.40 格式化输入函数: scanf()

函数 scanf()用于格式化输入,其原型如下。

```
int scanf(char *format[,argument,...]);
```

【参数】 参数 format 为要输入的格式。

【返回值】 若该函数输入成功,返回非零值;否则,返回 0。

【例 1-40】 下面的示例演示了 scanf()函数的使用,采用该函数进行格式化输入,代码如下。

```

#include <stdio.h> //加入标准输入输出库
#include <conio.h> //加入控制台输入输出库
void main( ){
    char label[20]; char name[20]; int entries = 0;
    int loop, age; double salary;
    struct Entry_struct { //定义结构体
        char name[20];
        int age;
        float salary;
    } entry[20];
    printf("\n\nPlease enter a label for the chart: ");
    //输出提示信息
    scanf("%20s", label); //格式化输入 20 个字符
    fflush(stdin); //清除标准输入控制台

```

```

printf("How many entries will there be? (less than 20) ");
                                                                    //输出提示信息
scanf("%d", &entries);
                                                                    //格式化输入整数
fflush(stdin);
                                                                    //清除标准输入控制台
for (loop=0;loop<entries;++loop) {
    printf("Entry %d\n", loop);
                                                                    //格式化输出
    printf(" Name  : ");
                                                                    //输出提示信息
    scanf("%[A-Za-z]", entry[loop].name);
                                                                    //格式化输入大小写字母
    fflush(stdin);
                                                                    //清除标准输入控制台
    printf(" Age   : ");
                                                                    //输出提示信息
    scanf("%d", &entry[loop].age);
                                                                    //格式化输入年龄
    fflush(stdin);
                                                                    //清除标准输入控制台
    printf(" Salary : ");
                                                                    //输出提示信息
    scanf("%f", &entry[loop].salary);
                                                                    //格式化输入薪水
    fflush(stdin);
                                                                    //清除标准输入控制台
}
printf("\nPlease enter your name, age and salary\n");
                                                                    //输出提示信息
scanf("%20s %d %lf", name, &age, &salary); //格式化输入
printf("\n\nTable %s\n",label);
                                                                    //以表格形式输出
printf("Compiled by %s age %d $%15.2lf\n", name, age, salary);
printf("-----\n");
for (loop=0;loop<entries;++loop)
    printf("%4d | %-20s | %5d | %15.2lf\n",
        loop + 1,
        entry[loop].name,
        entry[loop].age,
        entry[loop].salary);
    printf("-----\n");
}

```

运行上述程序,首先定义结构体,然后依次格式化输入标签、条目数,再依次输入每个条目的姓名、年龄和薪水,最后以表格的形式输出到控制台。

1.41 格式化输出函数: printf()

函数 printf() 用于格式化输出, 其原型如下。

```
int printf(char *format, ...);
```

【参数】 参数 format 为要输出的格式。

【返回值】 若输出成功, 返回 true; 否则, 返回 NULL (0)。

【例 1-41】 下面的示例演示了 printf() 函数的使用, 采用该函数进行格式化输出, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
main( ){
    int i;    char *str="GGV";
    textmode(0x00);

    printf("Printf Demo-%%c");                    //格式化输出
    printf("-----");
    printf("%c-%c-%c-%c\n", 'D', 'e', 'm', 'o');
    printf("%2c-%2c-%2c-%2c\n", 'D', 'e', 'm', 'o');
    printf("%02c-%02c-%02c-%02c\n", 'D', 'e', 'm', 'o');
    printf("%-2c-%-2c-%-2c-%-2c\n", 'D', 'e', 'm', 'o');
    getchar( );                                    //从 stdin 流中读字符
    textmode(0x00);                                //将屏幕设置成文本模式
    i=7412;
    printf("Printf Demo-%%d");
    printf("-----");
    printf("%d\n", i);
    printf("%14d\n", i);
    printf("%+10d\n", i);
    printf("%-10d\n", i);
    getchar( );                                    //从 stdin 流中读字符
    clrscr( );
    printf("Printf - d,o,x");
    printf("-----");
    printf("%d\n", i);
    printf("%o\n", i);
```

```
printf("%x\n", i);
```

运行上述程序, 首先清除屏幕, 然后将屏幕设置为文本模式, 再依次进行格式化输出。

1.42 重定位流上的文件指针函数: fseek()

函数 `fseek()` 用于重定位流上的文件指针, 其原型如下。

```
int fseek(FILE *stream, long offset, int fromwhere);
```

【参数】参数 `stream` 为要重定位的流; 参数 `offset` 为重定位的偏移量; 参数 `fromwhere` 为重定位的位置。

【返回值】若执行成功, 返回非零值; 否则, 返回 0。

【例 1-42】下面的示例演示了 `fseek()` 函数的使用, 采用该函数重定位流上的文件指针, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
long filesize(FILE *stream);                       //声明自定义变量
void main( ){
    FILE *stream;
    stream = fopen("MYFILE.TXT", "w+");           //打开文件
    fprintf(stream, "This is a test");            //格式化输出流中
    printf("Filesize of MYFILE.TXT is %ld bytes\n", //格式化输出文件大小
    filesize(stream));
    fclose(stream);                                //关闭文件
}

long filesize(FILE *stream){
    long curpos, length;
    curpos = ftell(stream);                         //返回当前指针
    fseek(stream, 0L, SEEK_END);                    //重定位指针到文件尾
    length = ftell(stream);                          //返回当前指针
    fseek(stream, curpos, SEEK_SET);                 //重定位指针到原位置
    return length;                                  //返回文件大小
}
```

}

运行上述程序，首先声明需要获取文件大小的自定义函数；然后打开文件，格式化输出到文件流中；最后采用自定义函数获取文件大小，并进行格式化输出。

1.43 返回当前文件指针函数：ftell()

函数 ftell() 用于返回当前文件指针，其原型如下。

```
long ftell(FILE *stream);
```

【参数】 参数 stream 为需要处理的文件流。

【返回值】 该函数返回当前文件指针位置。

【例 1-43】 下面的示例演示了 ftell() 函数的使用，采用该函数返回当前文件指针，其代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
void main( ){
    FILE *stream;
    stream = fopen("MYFILE.TXT", "w+");           //打开文件
    fprintf(stream, "This is a test");             //格式化输出到流
    printf("The file pointer is at byte %ld\n", ftell(stream));
                                                    //获取并输出当前文件指针
    fclose(stream);                                //关闭文件
}
```

运行上述程序，首先打开文件，然后格式化输出到文件流，再获取并输出当前文件指针位置，最后关闭打开的文件。

第2章 数学函数库: math.h

在程序设计中,常常需要用到一些数学函数进行相应的操作。C语言专门提供了一个针对数学运算的函数库——math.h,该文件库中存在的常用函数主要包括三角函数、双曲线函数、对数函数等数学运算函数。

2.1 浮点数绝对值函数: fabs()

函数 fabs() 用于返回浮点数的绝对值,其原型如下。

```
double fabs(double x);
```

【参数】 参数x为要求绝对值的浮点数,其类型为双精度实数。

【返回值】 该函数执行后,将返回双精度实数的绝对值。

【例 2-1】 下面的示例演示了 fabs() 函数的使用,采用该函数获取浮点数的绝对值,代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <math.h> //加入数学函数库
void main() {
    float number = -1234.0; //定义浮点数
    printf("number: %f absolute value: %f\n",
        number, fabs(number)); //取浮点数,并输出
}
```

运行上述程序,首先定义一个浮点数,然后采用 fabs() 函数取浮点数的绝对值,再使用 printf() 函数输出,输出结果如下。

```
number: -1234.000000 absolute value: 1234.000000
```

2.2 整数绝对值函数: abs()

函数 `abs()` 用于取整数的绝对值, 其原型如下。

```
int abs(int i);
```

【参数】 参数 `i` 为整型数。

【返回值】 该函数执行后, 将返回一个整型数的绝对值。

【例 2-2】 下面的示例演示了 `abs()` 函数的使用, 采用该函数获取给定整型数的绝对值, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <math.h> //加入数学函数库
void main() {
    int number = -1234; //定义一个整型数
    printf("number: %d absolute value: %d\n", number, abs(number));
    //取整数的绝对值, 并输出
}
```

运行上述程序, 采用 `abs()` 函数获取所定义的整型数的绝对值, 并将其输出, 输出结果如下。

```
number: -1234 absolute value: 1234
```

2.3 反余弦值函数: acos()

函数 `acos()` 用于取给定值的反余弦函数, 其原型如下。

```
double acos(double x);
```

【参数】 参数 `x` 为要取反余弦的双精度数。

【返回值】 该函数返回给定值的反余弦值。

【例 2-3】 下面的示例演示了 `acos()` 函数的使用, 采用该函数获取给定值的反余弦值, 并输出相应结果, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
```

```

#include <math.h>                                //加入数学函数库
void main( ){
    double result;                                //定义结果值
    double x = 0.5;                               //定义一个双精度实数
    result = acos(x);                             //取反余弦值
    printf("The arc cosine of %lf is %lf\n", x, result); //输出结果
}

```

运行上述程序, 首先定义一个双精度实数, 然后采用 `acos()` 函数获取其反余弦值, 并输出如下结果。

```
The arc cosine of 0.500000 is 1.047198
```

2.4 反正弦值函数: `asin()`

函数 `asin()` 用于取给定值的反正弦函数, 其原型如下。

```
double asin(double x);
```

【参数】 参数 `x` 表示要取反正弦值的双精度数。

【返回值】 该函数返回给定值的反正弦值。

【例 2-4】 下面的示例演示了 `asin()` 函数的使用, 采用该函数获取给定值的反正弦值, 代码如下。

```

#include <stdio.h>                                //加入标准输入输出库
#include <math.h>                                //加入数学函数库
void main( ){
    double result;                                //定义结果值
    double x = 0.5;                               //定义双精度浮点数
    result = asin(x);                             //获取反正弦值
    printf("The arc sin of %lf is %lf\n", x, result); //输出结果
}

```

运行上述程序, 首先定义需要取反正弦值的双精度浮点数, 再以 `asin()` 函数获取其反正弦值, 最后输出如下结果。

```
The arc sin of 0.500000 is 0.523599
```


2.5 反正切函数: atan()

函数 atan() 用于取给定值的反正切值, 其原型如下。

```
double atan(double x);
```

【参数】 参数x为要取反正切值的双精度浮点数。

【返回值】 该函数返回参数x的反正切值。

【例 2-5】 下面的示例演示了 atan() 函数的使用, 采用该函数获取给定值的反正切值, 其代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <math.h>             //加入数学函数库
void main( ){
    double result;           //定义结果值
    double x = 0.5;          //定义双精度浮点数
    result = atan(x);        //取反正切值
    printf("The arc tangent of %lf is %lf\n", x, result); //输出结果
}
```

运行上述程序, 首先定义一个需要求反正切值的双精度浮点数, 再采用 atan() 函数求其反正切值, 输出如下结果。

```
The arc tangent of 0.500000 is 0.463648
```

2.6 Y/X 反正切函数: atan2()

函数 atan2() 用于计算 Y/X 的反正切值, 其原型如下。

```
double atan2(double y, double x);
```

【参数】 参数y和x均为双精度浮点数。

【返回值】 该函数返回Y和X的商的反正切值。

【例 2-6】 下面的示例演示了 atan2() 函数的使用, 采用该函数计算给定值的反正切值, 代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <math.h>            //加入数学函数库
void main( ){
    double result;
    double x = 90.0, y = 45.0;           //定义双精度浮点数
    result = atan2(y, x);                //取 y/x 的反正切值
    printf("The arc tangent ratio of %lf is %lf\n", (y / x), result);
                                         //输出结果
}

```

运行上述程序, 首先定义两个双精度浮点数, 再采用 `atan2()` 函数取两个数的反正切值, 输出如下结果。

```
The arc tangent ratio of 0.500000 is 0.463648
```

2.7 不小于某数的最小整数函数: `ceil()`

函数 `ceil()` 用于取不小于某数的最小整数, 其原型如下。

```
double ceil(double x);
```

【参数】 参数 `x` 表示要取整的双精度浮点数。

【返回值】 该函数返回一个不小于给定值的最小整数。

【例 2-7】 下面的示例演示了 `ceil()` 函数的使用, 采用该函数返回一个不小于给定值的最小整数, 代码如下。

```

#include <math.h>           //加入数学函数库
#include <stdio.h>          //加入标准输入输出库
void main( ){
    double number = 123.54;           //定义一个双精度浮点数
    double up;                        //定义结果变量
    up = ceil(number);                //取最小整数
    printf("original number    %5.2lf\n", number); //输出原始数
    printf("number rounded up  %5.2lf\n", up);      //输出结果
}

```

运行上述程序, 首先定义一个双精度浮点数, 再采用 `ceil()` 函数

取不小于给定数的最小整数，输出如下结果。

```
original number    123.54
number rounded up  124.00
```

2.8 余弦值函数：cos()

函数 cos() 用于获取给定值的余弦值，其原型如下。

```
double cos(double x);
```

【参数】 参数 x 为要取余弦值的双精度浮点数。

【返回值】 该函数返回给定值的余弦值。

【例 2-8】 下面的示例演示了 cos() 函数的使用，采用该函数获取给定值的余弦值，代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <math.h>             //加入数学函数库
void main( ){
    double result;           //定义结果变量
    double x = 0.5;          //定义求余弦值的双精度变量
    result = cos(x);          //求余弦值
    printf("The cosine of %lf is %lf\n", x, result); //输出结果
}
```

运行上述程序，首先定义一个需要求余弦值的双精度变量，然后采用 cos() 函数取其余弦值，输出如下结果。

```
The cosine of 0.500000 is 0.877583
```

2.9 双曲余弦值函数：cosh()

函数 cosh() 用于取给定值的双曲余弦值，其原型如下。


```
double cosh(double x);
```

【参数】 参数x为要取双曲余弦值的双精度浮点数。

【返回值】 该函数返回给定值的双曲余弦值。

【例 2-9】 下面的示例演示了cosh()函数的使用, 采用该函数获取给定值的双曲余弦值, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <math.h>            //加入数学函数库
void main( ){
    double result;           //定义结果变量
    double x = 0.5;         //定义一个需要求双曲余弦值的双精度浮点数
    result = cosh(x);        //取双曲余弦值
    printf("The hyperboic cosine of %lf is %lf\n", x, result);
                               //输出结果
}
```

运行上述程序, 首先定义一个需要求双曲余弦值的双精度浮点数, 再采用 cosh()函数获取其双曲余弦值, 输出如下结果。

```
The hyperboic cosine of 0.500000 is 1.127326
```

2.10 e 的次幂函数: exp()

函数 exp()用于计算取 e 的给定次幂值, 其原型如下。

```
double exp(double x);
```

【参数】 参数x为要取e的指定次幂。

【返回值】 该函数返回e的给定次幂值。

【例 2-10】 下面的示例演示了exp()函数的使用, 采用该函数计算欧拉数e的指定次幂值, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <math.h>            //加入数学函数库
void main( ){
```

```

double result;           //定义结果变量
double x = 4.0;          //定义双精度浮点数
result = exp(x);          //计算欧拉数e的x次幂值
printf("'e' raised to the power \
of %lf (e ^ %lf) = %lf\n", x, x, result); //输出结果
}

```

运行上述程序,首先定义一个要计算e的x次幂,然后采用exp()函数获取e的幂值,输出结果如下。

```
'e' raised to the power of 4.000000 (e ^ 4.000000) = 54.598150
```

2.11 不大于某数的最大整数函数: floor()

函数 floor()用于获取不大于某数的最大整数,其原型如下。

```
double floor(double x);
```

【参数】参数x为要取整的双精度浮点数。

【返回值】该函数返回一个不大于给定值的最大整数值。

【例 2-11】下面的示例演示了floor()函数的使用,使用该函数获取不大于给定值的最大整数值,代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <math.h>            //加入数学函数库
void main( ){
    double number = 123.54;   //定义一个双精度浮点数
    double down;             //定义结果变量
    down = floor(number);     //取不小于给定值的最大整数值
    printf("original number %10.2lf\n", number); //输出原始给定值
    printf("number rounded down %10.2lf\n", down);
                                //输出不大于某数的最大整数值
}

```

运行上述程序后,将采用 floor()函数获取不大于给定值的最大整数,并将其输出,输出如下结果。

```
original number      123.54
number rounded down  123.00
```

2.12 余数函数: fmod()

函数 `fmod()` 用于计算给定值的余数, 其原型如下。

```
double fmod(double x, double y);
```

【参数】 参数 `x` 和参数 `y` 为要进行计算的双精度浮点数。

【返回值】 该函数返回参数 `x` 除以参数 `y` 后的余数值。

【例 2-12】 下面的示例演示了 `fmod()` 函数的使用, 采用该函数计算给定参数值相除的余数, 并输出相应结果, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <math.h>            //加入数学函数库
void main( ){
    double x = 5.0, y = 2.0; //定义要进行计算的双精度浮点数
    double result;           //定义结果变量
    result = fmod(x,y);      //计算给定值的余数
    printf("The remainder of (%lf / %lf) is %lf\n", x, y, result);
                               //输出结果
}
```

运行上述程序, 首先定义要进行计算的双精度浮点数, 并采用 `fmod()` 函数获取给定值相除后的余数, 输出如下结果。

```
The remainder of (5.000000 / 2.000000) is 1.000000
```

2.13 计算浮点数尾数和指数函数: frexp()

函数 `frexp()` 把一个双精度数分解为尾数的指数, 原型如下。

```
double frexp(double value, int *eptr);
```


【参数】 参数value为要进行分解的双精度浮点数；参数*eptr为指针类型的形参。

【返回值】 该函数将给定的双精度数分解为数字部分和以2为底的指数n，即 $value = x * 2^n$ ，n存储在eptr所指向的变量中。

【例 2-13】 下面的示例演示了frexp()函数的使用，采用该函数将给定值分解为尾数的指数，代码如下。

```
#include <math.h>           //加入数学函数库
#include <stdio.h>          //加入标准输入输出库
void main( ){
    double mantissa, number; //定义两个双精度变量
    int exponent;            //定义整数变量
    number = 8.0;
    mantissa = frexp(number, &exponent); //计算给定浮点数的尾数和指数
    printf("The number %lf is ", number); //输出原始浮点数
    printf("%lf times two to the ", mantissa); //输出尾数
    printf("power of %d\n", exponent); //输出指数
}
```

运行上述程序，首先定义两个用于保存尾数和指数的双精度浮点数，然后采用frexp()函数计算给定值的尾数和指数，输出如下结果。

```
The number 8.000000 is 0.500000 times two to the power of 4
```

2.14 计算幂函数：ldexp()

函数ldexp()用于计算指定的幂次数，其原型如下。

```
double ldexp(double value, int exp);
```

【参数】 参数value为要计算指定次幂的双精度浮点数；参数exp为指定的幂次数。

【返回值】 该函数返回指定幂次值。

【例 2-14】 下面的示例演示了ldexp()函数的使用，采用该函数计算

给定数的指定次幂值,代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <math.h> //加入数学函数库
void main( ){
    double value; //定义结果变量
    double x = 2; //定义要计算幂的浮点数
    value = ldexp(x,3); //计算给定次幂值
    printf("The ldexp value is: %lf\n", value); //输出结果
}
```

运行上述程序,首先定义一个要计算指定幂次的浮点数和指定的幂次,然后计算该浮点数的指定次幂值,输出如下结果。

```
The ldexp value is: 8.000000
```

2.15 取浮点数小数部分函数: modf()

函数 modf() 用于取给定浮点数的小数部分,其原型如下。

```
double modf(double value, double *iptr);
```

【参数】 参数 value 为要取小数部分的浮点数; 参数 iptr 为取小数后的整数值。

【返回值】 该函数将参数 value 分割为整数和小数, 返回小数部分并将整数部分赋给 iptr。

【例 2-15】 下面的示例演示了 modf() 函数的使用, 采用该函数获取给定浮点数的小数部分, 代码如下。

```
#include <math.h> //加入数学函数库
#include <stdio.h> //加入标准输入输出库
void main( ){
    double fraction, integer; //定义保存小数部分和整数部分的变量
    double number = 100000.567; //定义双精度浮点数
    fraction = modf(number, &integer); //获取浮点数的小数部分
    printf("The whole and fractional parts of %lf are %lf and %lf\n",
```

```
number, integer, fraction); //输出小数部分和整数部分
```

```
}
```

运行上述程序，首先定义两个用于保存小数部分和整数部分的浮点数变量，然后采用 `modf()` 函数取给定变量值的小数部分和整数部分，并输出如下结果。

```
The whole and fractional parts of 100000.567000 are 100000.000000
and 0.567000
```

2.16 计算直角三角形斜边长度函数: `hypot()`

函数 `hypot()` 用于计算直角三角形的斜边长度值，其原型如下。

```
double hypot(double x, double y);
```

【参数】 参数 `x` 和参数 `y` 为直角三角形的两条直角边。

【返回值】 该函数返回给定两直角边的直角三角形的斜边值。

【例 2-16】 下面的示例演示了 `hypot()` 函数的使用，采用该函数计算给定两直角边的直角三角形的斜边，代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <math.h> //加入数学函数库
void main( ){
    double result; //定义用于保存斜边的双精度变量
    double x = 3.0; //定义直角三角形的一边
    double y = 4.0; //定义直角三角形的另一边
    result = hypot(x, y); //计算直角三角形的斜边
    printf("The hypotenuse is: %lf\n", result); //输出结果
}
```

运行上述程序，首先定义直角三角形的两直角边，然后采用 `hypot()` 函数计算直角三角形的斜边，并输出如下结果。

```
The hypotenuse is: 5.000000
```


2.17 自然对数函数: log()

函数 log() 用于取给定值的自然对数值, 其原型如下。

```
double log(double x);
```

【参数】 参数 x 为要取自然对数的双精度浮点数。

【返回值】 该函数返回给定参数值的自然对数值。

【例 2-17】 下面的示例演示了 log() 函数的使用, 采用该函数取给定数值的自然对数值, 代码如下。

```
#include <math.h>           //加入数学函数库
#include <stdio.h>          //加入标准输入输出库
void main( ){
    double result;           //定义保存结果的双精度变量
    double x = 8.6872;       //定义要取自然对数的变量
    result = log(x);          //取自然对数
    printf("The natural log of %lf is %lf\n", x, result); //输出结果
}
```

运行上述程序, 首先定义要取自然对数的双精度变量, 然后采用 log() 函数取指定值的自然对数, 并输出如下结果。

```
The natural log of 8.687200 is 2.161851
```

2.18 对数函数: log10()

函数 log10() 用于取指定数值的以 10 为基数的对数, 其原型如下。

```
double log10(double x);
```

【参数】 参数 x 为要取对数的双精度浮点数。

【返回值】 该函数返回给定数值的以 10 为基数的对数。

【例 2-18】 下面的示例演示了 log10() 函数的使用, 采用该函数计算给定值的以 10 为基数的对数, 代码如下。

```

#include <math.h>           //加入数学函数库
#include <stdio.h>          //加入标准输入输出库
void main( ){
    double result;          //定义保存结果的双精度变量
    double x = 800.6872;    //定义要取对数的变量
    result = log10(x);       //取以 10 为基数的对数
    printf("The common log of %lf is %lf\n", x, result); //输出结果
}

```

运行上述程序，首先定义要取对数的双精度变量，然后采用 `log10()` 函数取其以 10 为基数的对数，并输出如下结果。

```
The common log of 800.687200 is 2.903463
```

2.19 计算 x 的 y 次幂函数: `pow()`

函数 `pow()` 用于计算给定数值的指定次幂，其原型如下。

```
double pow(double x, double y);
```

【参数】 参数 x 为要计算幂的值；参数 y 为要计算的幂次。

【返回值】 该函数返回给定值 x 的 y 次幂。

【例 2-19】 下面的示例演示了 `pow()` 函数的使用，采用该函数计算给定数值的指定次幂，代码如下。

```

#include <math.h>           //加入数学函数库
#include <stdio.h>          //加入标准输入输出库
void main( ){
    double x = 2.0, y = 3.0; //定义要计算幂的值和幂次
    printf("%lf raised to %lf is %lf\n", x, y, pow(x, y));
                                //计算幂次值并输出结果
}

```

运行上述程序，首先定义要计算幂和幂次值，然后采用 `pow()` 函数计算幂次值，并输出如下结果。

```
2.000000 raised to 3.000000 is 8.000000
```

2.20 计算 10 的 x 次幂函数: pow10()

函数 pow10() 用于计算给定数值的 10 的指定次幂值, 其原型如下。

```
double pow10(int p);
```

【参数】 参数 p 为要计算 10 的指定次幂。

【返回值】 该函数返回 10 的指定次幂 p 的值。

【例 2-20】 下面的示例演示了 pow10() 函数的使用, 使用该函数计算 10 的指定次幂值, 代码如下。

```
#include <math.h> //加入数学函数库
#include <stdio.h> //加入标准输入输出库
void main() {
    double p = 3.0; //定义要计算的幂次
    printf("Ten raised to %lf is %lf\n", p, pow10(p));
    //计算 10 的指定幂次值并输出结果
}
```

运行上述程序后, 将采用 pow10() 函数计算指定次幂值, 并输出如下结果。

```
Ten raised to 3.000000 is 1000.000000
```

2.21 正弦值函数: sin()

函数 sin() 用于计算给定值的正弦值, 其原型如下。

```
double sin(double x);
```

【参数】 参数 x 为要计算正弦值的值。

【返回值】 该函数返回给定值的正弦值。

【例 2-21】 下面的示例演示了 sin() 函数的使用, 采用该函数输出给定值的正弦值, 并输出相应结果, 代码如下。


```

#include <stdio.h> //加入标准输入输出库
#include <math.h> //加入数学函数库
void main( ){
    double result, x = 0.5; //定义双精度变量
    result = sin(x); //取给定值的正弦值
    printf("The sin( ) of %lf is %lf\n", x, result); //输出结果
}

```

运行上述程序，首先定义要取正弦值的双精度浮点数，然后采用 `sin()` 函数取其正弦值，并输出如下结果。

```
The sin( ) of 0.500000 is 0.479426
```

2.22 双曲正弦值函数: `sinh()`

函数 `sinh()` 用于计算给定数值的双曲正弦值，其原型如下。

```
double sinh(double x);
```

【参数】 参数 `x` 为要计算双曲正弦值的值。

【返回值】 该函数返回给定值的双曲正弦值。

【例 2-22】 下面的示例演示了 `sinh()` 函数的使用，采用该函数计算给定值的双曲正弦值，代码如下。

```

#include <stdio.h> //加入标准输入输出库
#include <math.h> //加入数学函数库
void main( ){
    double result, x = 0.5; //定义双精度浮点数
    result = sinh(x); //取双曲正弦值
    printf("The hyperbolic sin( ) of %lf is %lf\n", x, result); //输出结果
}

```

运行上述程序，首先定义要取双曲正弦值的双精度浮点数，然后采用 `sinh()` 函数取双曲正弦值，并输出如下结果。

```
The hyperbolic sin( ) of 0.500000 is 0.521095
```

2.23 平方根函数: sqrt()

函数 sqrt() 用于计算给定值的平方根, 其原型如下。

```
double sqrt(double x);
```

【参数】 参数 x 为要取平方根的值。

【返回值】 该函数返回给定值的平方根值。

【例 2-23】 下面的示例演示了 sqrt() 函数的使用, 采用该函数计算给定值的平方根, 代码如下。

```
#include <math.h> //加入数学函数库
#include <stdio.h> //加入标准输入输出库
void main() {
    double x = 4.0, result; //定义双精度浮点数
    result = sqrt(x); //取平方根
    printf("The square root of %lf is %lf\n", x, result); //输出结果
}
```

运行上述程序, 首先定义要取平方根的值和保存平方根的双精度浮点数变量, 然后采用 sqrt() 函数计算给定值的平方根, 并输出结果。

2.24 正切值函数: tan()

函数 tan() 用于计算给定值的正切值, 其原型如下。

```
double tan(double x);
```

【参数】 参数 x 为要计算正切值的双精度浮点数值。

【返回值】 该函数返回给定数值的正切值。

【例 2-24】 下面的示例演示了 tan() 函数的使用, 采用该函数计算给定数值的正切值, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <math.h> //加入数学函数库
```

```

void main( ){
    double result, x;           //定义双精度浮点数变量
    x = 0.5;                     //定义要取正切的变量
    result = tan(x);             //取正切值
    printf("The tan of %lf is %lf\n", x, result); //输出结果
}

```

运行上述程序,首先定义两个双精度浮点数变量,然后采用 `tan()` 函数计算给定值的正切值,并输出如下结果。

```
The tan of 0.500000 is 0.546302
```

2.25 双曲正切值函数: `tanh()`

函数 `tanh()` 用于计算给定值的双曲正切值,其原型如下。

```
double tanh(double x);
```

【参数】 参数 `x` 为要计算双曲正切值的双精度浮点数值。

【返回值】 该函数返回给定值的双曲正切值。

【例 2-25】 下面的示例演示了 `tanh()` 函数的使用,采用该函数计算给定值的双曲正切值,代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <math.h>            //加入数学函数库
void main( ){
    double result, x;        //定义两个双精度浮点数
    x = 0.5;                 //定义要取双曲正切值的变量
    result = tanh(x);        //取双曲正切值
    printf("The hyperbolic tangent of %lf is %lf\n", x, result);
                                //输出结果
}

```

运行上述程序,首先定义两个双精度浮点数,然后取给定值的双曲正切值,并输出如下结果。

```
The hyperbolic tangent of 0.500000 is 0.462117
```


第3章 字符函数库: ctype.h

在程序设计中,很大部分工作都是在对字符进行相应的操作。C语言专门提供了一个针对字符操作的函数库——ctype.h,该文件库中存在的常用函数主要包括字符的类型判断和字符的转换两大类函数。

3.1 判断字符是否为控制字符函数: iscntrl()

函数 iscntrl() 用于判断所给的字符是否为控制字符(即其十进制 ASCII 值为 0~31 的字符为控制字符),其原型如下。

```
extern int iscntrl(int c);
```

【参数】参数c表示需要进行判断的字符。

【返回值】当参数c的十六进制ASCII值为 0x00~0x1F或等于 0x7F (DEL) 时,将返回非零值;否则,返回 0。

【例 3-1】下面的示例演示了 iscntrl() 函数的使用,判断所给参数是否为控制字符,代码如下。

```
#include <ctype.h>           //加入字符函数库
#include <stdio.h>           //加入标准输入输出库
#include <conio.h>           //加入控制台输入输出库
void main() {
    int c; clrscr(); c='a';
    printf("%x:%s\n",c,iscntrl(c)?"yes":"no"); //显示判断结果
    c=0x0d; //十六进制 d
    printf("%x:%s\n",c,iscntrl(c)?"yes":"no"); //显示判断结果
    c=0x7f; //十六进制 7f
    printf("%x:%s\n",c,iscntrl(c)?"yes":"no"); //显示判断结果
    getch();
```

}

运行上述程序，将依次使用 `isctrl()` 函数判断所给的 3 个参数是否为控制字符。若是控制字符，则输出“yes”；否则，输出“no”，其输出结果如下。

```
6l:no
d:yes
7f:yes
```

3.2 判断字符是否为字母或数字函数: `isalnum()`

函数 `isalnum()` 用于判断所给的字符是否为大小写字母或数字，其原型如下。

```
extern int isalnum(int c);
```

【参数】 参数 `c` 表示需要进行判断的字符。

【返回值】 当 `c` 为数字 0~9 或字母 a~z 及 A~Z 时，返回非零值；否则，返回 0。

【例 3-2】 下面的示例演示了 `isalnum()` 函数的使用，使用该函数判断所给的参数是否为字母或数字，代码如下。

```
#include <ctype.h>           //加入字符函数库
#include <stdio.h>           //加入标准输入输出库
#include <conio.h>           //加入控制台输入输出库
void main( ){
    int c; clrscr( ); c='a';
    printf("%c:%s\n",c,isalnum(c)?"yes":"no"); //显示判断结果
    c='7';
    printf("%c:%s\n",c,isalnum(c)?"yes":"no"); //显示判断结果
    c='@';
    printf("%c:%s\n",c,isalnum(c)?"yes":"no"); //显示判断结果
    getch( );
}
```

运行上述代码,将依次对所给的3个变量进行判断,判断是否为字母和数字。采用三元运算符进行判断,若是字母和数字,则输出“yes”;否则,输出“no”,其输出结果如下。

```
a:yes
7:yes
@:no
```

3.3 判断字符是否为英文字母函数: isalpha()

函数 isalpha() 用于判断字符是否为英文字母,其原型如下。

```
extern int isalpha(int c);
```

【参数】 参数c表示要进行判断的字符。

【返回值】 当参数c为英文字母a~z或A~Z时,返回非零值;否则,返回0。

【例 3-3】 下面的示例演示了 isalpha() 函数的使用,判断用户在控制台所输入的字符是否为英文字母,并输出相应结果,代码如下。

```
#include <ctype.h>           //加入字符函数库
#include <stdio.h>           //加入标准输入输出库
void main() {
    int c;
    printf("Press a key:");
    for(;;) {
        c=getchar();          //获取输入字符
        printf("%c: %s letter",c,isalpha(c)?"is":"not");
                                //输入判断结果
    }
}
```

运行上述代码,首先提示用户输入字符,用户输入字符后,判断是否为英文字母,并输出如下结果。

```
Press a key:a
```



```
a: is letter
```

注意：上述运行结果与读者运行程序时所输入的字符有关。

3.4 判断字符是否为 ASCII 码函数：isascii()

函数 isascii() 用于判断字符是否为 ASCII 码，其原型如下。

```
extern int isascii(int c);
```

【参数】 参数 c 表示要判断的字符。

【返回值】 当 c 为 ASCII 码时，返回非零值；否则，返回 0。ASCII 码指 0x00~0x7F 字符。

【例 3-4】 下面的示例演示了 isascii() 函数的使用，判断数组中每一个字符是否是 ASCII 字符，代码如下。

```
#include <ctype.h>           //加入字符函数库
#include <system.h>          //加入标准库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    char s[]="ABCD";int i=12;
    textmode(0xE0);          //设置字符显示模式
    printf("%s\n",s);
    for(i=0;i<12;i++) {
        if(isascii(s[i])) putchar('^');//判断每一位是否为 ASCII 码
        else putchar('.');
    }
    getchar( );
}
```

运行上述程序，首先设置屏幕的显示模式，再以字符串的形式输出所有字符，最后采用 isascii() 函数判断字符串每一位是否为 ASCII 码。若是 ASCII 码，则输出 “^”；否则，输出 “.”，其输出结果如下。

```
ABCD
^^^^^^
```

注意: 读者在运行时可自行修改变量 *s* 的值, 此时将得到不同结果。

3.5 判断字符是否为 TAB 或空格函数: isblank()

函数 `isblank()` 用于判断字符是否为 TAB 或空格, 其原型如下。

```
extern int isblank(int c);
```

【参数】参数 *c* 表示要进行判断的字符。

【返回值】若参数 *c* 为空格, 则返回 `true`; 否则, 返回 `NULL (0)`。

【例 3-5】下面的示例演示了 `isblank()` 函数的使用, 判断字符串中的每一个字符是否为 TAB 或者空格, 代码如下。

```
#include <ctype.h> //加入字符函数库
#include <stdio.h> //加入标准输入输出库
void main() {
    char str[] = "1234fsdfk jksd@ fsdf ?fsdf j d f";
    int i;
    for(i=0; str!=0; i++)
        if(isblank(str)) printf("str[%d] is blank character: %d\n", i, str);
}
```

运行上述程序, 依次遍历字符串的每一位, 并判断其是否为空格。若该位不是空格, 忽略不计; 若是空格, 则输出相应的提示信息。

3.6 判断字符是否为除空格外的可打印字符

函数: `isgraph()`

函数 `isgraph()` 用于判断字符是否为除空格外的可打印字符, 其原

型如下。

```
extern int isgraph(int c);
```

【参数】参数c表示要进行判断的字符。

【返回值】若参数c为可打印字符，即其十六进制ASCII码为 0x21~0x7e时，返回非零值；否则，返回 0。

【例 3-6】下面的示例演示了isgraph()函数的使用，判断字符是否为除空格外的可打印字符，代码如下。

```
#include <ctype.h>           //加入字符函数库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    int c;    c='a';         //字符 a
    printf("%c:%s\n",c,isgraph(c)?"yes":"no");
                                //判断 a 是否为可打印字符
    c=' ';                  //空格，其 ASCII 码为 0x20
    printf("%c:%s\n",c,isgraph(c)?"yes":"no");
                                //判断空格是否为可打印字符
    c=0x7f;
    printf("%c:%s\n",c,isgraph(c)?"yes":"no");
                                //判断 0x7f 是否为可打印字符
    getchar( );
}
```

运行上述程序，依次判断每一个字符是否为除空格外的可打印字符。若该位是除空格以外的可打印字符，则输出“yes”；否则，输出“no”，其输出结果如下。

```
a:yes
:no
0:no
```

3.7 判断字符是否为小写英文字母函数: islower()

函数 islower()用于判断字符是否为小写英文字母，其原型如下。


```
extern int islower(int c);
```

【参数】参数c表示要进行判断的字符。

【返回值】若参数c为小写英文字母(a~z)时, 返回非零值; 否则, 返回0。

【例 3-7】下面的示例演示了islower()函数的使用, 判断每一个字符是否为小写的英文字母, 代码如下。

```
#include <ctype.h>           //加入字符函数库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    int c;    c='a';
    printf("%c:%s\n",c,islower(c)?"yes":"no");
                                   //判断字母 a 是否为小写英文字母
    c='A';
    printf("%c:%s\n",c,islower(c)?"yes":"no");
                                   //判断字母 A 是否为小写英文字母
    c='7';
    printf("%c:%s\n",c,islower(c)?"yes":"no");
                                   //判断 7 是否为小写英文字母
    getchar( );
}
```

运行上述程序, 依次判断每一个字符是否为小写英文字母。若该字符是小写英文字母, 则输出“yes”; 否则, 输出“no”, 其输出结果如下。

```
a:yes
A:no
7:no
```

3.8 判断字符是否为可打印字符(含空格)

函数: isprint()

函数 isprint()用于判断字符是否为可打印字符(含空格), 其原

型如下。

```
extern int isprint(int c);
```

【参数】参数c表示要进行判断的字符。

【返回值】若参数c为可打印字符，即其十六进制ASCII码为 0x20~0x7e 时，返回非零值；否则，返回 0。

【例 3-8】下面的示例演示了 isprint() 函数的使用，判断每一个字符是否为含空格的可打印字符，代码如下。

```
#include <ctype.h>           //加入字符函数库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    int c;    c='a';
    printf("%c:%s\n",c,isprint(c)?"yes":"no");
                                //判断字符 a 是否为可打印字符
    c=' ';
    printf("%c:%s\n",c,isprint(c)?"yes":"no");
                                //判断空格字符是否为可打印字符
    c=0x7f;
    printf("%c:%s\n",c,isprint(c)?"yes":"no");
                                //判断 0x7f 是否为可打印字符
    getchar( );
}
```

运行上述程序，依次判断给定的每一个字符是否为含空格的可打印字符。若该字符是含空格的可打印字符，则输出“yes”；否则，输出“no”，其输出结果如下。

```
a:yes
 :yes
0:no
```

(含空格) 判断字符是否为可打印字符 8.8

3.9 判断字符是否为标点符号函数：ispunct()

函数 ispunct() 用于判断字符是否为标点符号，其原型如下。

```
extern int ispunct(int c);
```

【参数】参数c表示要进行判断的字符。

【返回值】若参数c为标点符号时，返回非零值；否则，返回0。标点符号是指那些既不是字母数字，也不是空格的可打印字符。

【例3-9】下面的示例演示了ispunct()函数的使用，判断字符串数组每一个字符是否为标点符号，代码如下。

```
#include <ctype.h>           //加入字符函数库
#include <string.h>          //加入字符串库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    char s[ ]="Hello, Rain!";int i;
    printf("%s\n",s);
    for(i=0;i<strlen(s);i++) { //遍历字符串数组
        if(ispunct(s[i])) printf("^");//判断每一个字符是否为标点符号
        else printf(".");
    }
    getchar( );
}
```

运行上述程序，依次遍历字符串的每一位，并判断每一位是否为标点符号。若该位不是空格，则在该位对应输出“.”；若该位是标点符号，则在该位对应输出“^”，其输出内容如下。

```
Hello, Rain!
.....^.....^
```

3.10 判断字符是否为空白符函数: isspace()

函数 isspace()用于判断字符是否为空白符，其原型如下。

```
extern int isspace(int c);
```

【参数】参数c表示要进行判断的字符。

【返回值】若参数c为空白符时，返回非零值；否则，返回0。空白符

是指空格、水平制表符、垂直制表符、换页符、回车符和换行符。

【例 3-10】下面的示例演示了 `isspace()` 函数的使用，判断字符串每一个字符是否为空白字符，代码如下。

```
#include <ctype.h>           //加入字符函数库
#include <string.h>          //加入字符串库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    char s[ ]="Test Line 1\tend\nTest Line 2\r"; int i;
    for(i=0;i<strlen(s);i++) { //遍历字符串数组
        if(isspace(s[i])) putchar('.');//判断每一个字符是否为空白符
        else putchar(s[i]);
    }
    getchar( );
}
```

运行上述程序，依次遍历字符串的每一位，并判断其是否为空白符。若该位是空白符，则输出“.”；否则，输出该字符，其输出结果如下。

```
Test.Line.1.end.Test.Line.2.
```

3.11 判断字符是否为大写英文字母

函数: `isupper()`

函数 `isupper()` 用于判断字符是否为大写英文字母，其原型如下。

```
extern int isupper(int c);
```

【参数】 参数 `c` 表示要进行判断的字符。

【返回值】 若参数 `c` 为大写英文字母 (A~Z) 时，返回非零值；否则，返回 0。

【例 3-11】下面的示例演示了 `isupper()` 函数的使用，判断所给的字符

是否为大写英文字母,代码如下。

```
#include <ctype.h>           //加入字符函数库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    int c;    c='a';
    printf("%c:%s\n",c,isupper(c)?"yes":"no");
                                //判断字符 a 是否为大写英文字母

    c='A';
    printf("%c:%s\n",c,isupper(c)?"yes":"no");
                                //判断字符 A 是否为大写英文字母

    c='7';
    printf("%c:%s\n",c,isupper(c)?"yes":"no");
                                //判断字符 7 是否为大写英文字母

    getchar( );
}
```

运行上述程序,依次判断每一个字符是否为大写英文字母。若该字符是大写英文字母,则输出“yes”;否则,输出“no”,其输出结果如下。

```
a:no
A:yes
7:no
```

3.12 判断字符是否为十六进制数字

函数: isxdigit()

函数 isxdigit()用于判断字符是否为十六进制数字,其原型如下。

```
extern int isxdigit(int c);
```

【参数】参数c表示要进行判断的字符。

【返回值】若参数c为a~f、A~F或0~9的十六进制数字时,返回非零值;否则,返回0。

【例 3-12】下面的示例演示了 `isxdigit()` 函数的使用, 判断给定的每一个字符是否为十六进制数字, 代码如下。

```
#include <ctype.h>           //加入字符函数库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    int c;    c='a';
    printf("%c:%s\n",c,isxdigit(c)?"yes":"no");
                                   //判断字符 a 是否为十六进制数字
    c='9';
    printf("%c:%s\n",c,isxdigit(c)?"yes":"no");
                                   //判断字符 9 是否为十六进制数字
    c='*';
    printf("%c:%s\n",c,isxdigit(c)?"yes":"no");
                                   //判断字符*是否十六进制数字
    getchar( );
}
```

运行上述程序, 依次判断每一个字符是否为十六进制数字。若该字符是十六进制数字, 则输出 “yes”; 否则, 输出 “no”, 其输出结果如下。

```
a:yes
9:yes
*:no
```

3.13 将字符转换为 ASCII 码函数: `toascii()`

函数 `toascii()` 用于将字符转换为 ASCII 码, 其原型如下。

```
extern int toascii(int c);
```

【参数】 参数 `c` 表示要进行判断的字符。

【返回值】 该函数将字符 `c` 的高位清零, 仅保留低七位, 返回转换后的数值。

【例 3-13】下面的示例演示了 toascii() 函数的使用, 依次将字符串每一个字符转换为 ASCII 码并输出, 代码如下。

```
#include <ctype.h>           //加入字符函数库
#include <stdio.h>           //加入标准输入输出库
#include <system.h>          //加入标准库
void main( ){
    char s[ ]="中国-ABCDEF";   int i=12;
    textmode(0xE0);           //设置屏幕显示模式
    printf("%s\n",s);          //输出字符串
    for(i=0;i<12;i++)          //依次遍历
        putchar(toascii(s[i])); //将每一个字符转换成 ASCII 码并输出结果
    getchar( );
}
```

运行上述程序, 依次遍历字符串的每一个字符, 并将其转换为 ASCII 码, 并输出如下结果。

```
┌─┴─┐ -ABCDEF
VP9z-ABCDEF
```

3.14 将字符转换为小写英文字母

函数: tolower()

函数 tolower() 用于将字符转换为小写英文字母, 其原型如下。

```
extern int tolower(int c);
```

【参数】 参数 c 表示要进行判断的字符。

【返回值】 若参数 c 为大写英文字母, 则返回对应的小写字母; 否则, 返回原来的值。

【例 3-14】下面的示例演示了 tolower() 函数的使用, 将字符串每一个字符转换为小写英文字母并输出, 代码如下。

```
#include <ctype.h>           //加入字符函数库
```

```

#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    char *s="Hello, World!";  int i;
    printf("%s\n",s);         //输出字符串
    for(i=0;i<strlen(s);i++)  //遍历字符串
        putchar(tolower(s[i])); //将字符串中每一个字符转换为小写并输出
    getchar( );
}

```

运行上述程序，依次遍历字符串的每一位，将其转换为小写英文字母，并输出如下结果。

```

Hello, World!
hello, world!

```

3.15 将字符转换为大写英文字母

函数：toupper()

函数 toupper()用于将字符转换为大写英文字母，其原型如下。

```
extern int toupper(int c);
```

【参数】参数c表示要进行判断的字符。

【返回值】如果c为小写英文字母，则返回对应的大写字母；否则，返回原来的值。

【例 3-15】下面的示例演示了toupper()函数的使用，将字符串每一个字符转换为大写英文字母并输出，代码如下。

```

#include <ctype.h>           //加入字符函数库
#include <string.h>         //加入字符串库
#include <stdio.h>         //加入标准输入输出库
void main( ){
    char *s="Hello, World!";  int i;
    printf("%s\n",s);         //输出字符串
}

```

```

for(i=0;i<strlen(s);i++)           //遍历字符串
    putchar(toupper(s[i]));         //转换每一个字符并输出
getchar( );
}

```

运行上述程序,依次遍历字符串中的每一个字符,并将其转换为大写英文字母,并输出如下结果。

```

Hello, World!
HELLO, WORLD!

```

()qmod 模函数 计算各字符串字模

在C语言中,模函数qmod是用于计算字符串字模的函数。其原型如下:

```
int qmod(const char *s, int n);
```

其中,s为待计算的字符串,n为模数。该函数的作用是计算字符串s中每个字符的ASCII码值对n取模的结果,并将结果存储在s指向的内存空间中。如果n为0,则返回-1。

该函数在C语言中主要用于字符串的加密和解密操作。例如,可以将字符串中的每个字符的ASCII码值对26取模,从而实现字母表的循环移位。

该函数在C语言中属于ctype.h头文件。其实现原理如下:

该函数的实现原理如下:

```

int qmod(const char *s, int n)
{
    if (n == 0) return -1;
    for (int i = 0; i < strlen(s); i++)
    {
        s[i] = (s[i] % n + n) % n;
    }
    return 0;
}

```

该函数的实现原理如下:

```

int qmod(const char *s, int n)
{
    if (n == 0) return -1;
    for (int i = 0; i < strlen(s); i++)
    {
        s[i] = (s[i] % n + n) % n;
    }
    return 0;
}

```

该函数的实现原理如下:

```

int qmod(const char *s, int n)
{
    if (n == 0) return -1;
    for (int i = 0; i < strlen(s); i++)
    {
        s[i] = (s[i] % n + n) % n;
    }
    return 0;
}

```

该函数的实现原理如下:

```

int qmod(const char *s, int n)
{
    if (n == 0) return -1;
    for (int i = 0; i < strlen(s); i++)
    {
        s[i] = (s[i] % n + n) % n;
    }
    return 0;
}

```


第4章 字符串函数库: string.h

在程序设计中,常常需要对字符串进行相应的操作。C语言专门提供了一个针对字符串操作的函数库——string.h,该文件库中存在的常用函数主要包括字符串的比较、复制、查找等操作函数。

4.1 比较字符串是否相等函数: bcmp()

函数 bcmp() 用于比较给定的两字符串是否相等,其原型如下。

```
int bcmp(const void *s1, const void *s2, int n);
```

【参数】 参数s1和s2为要比较的字符串;参数n为要比较的字符长度。

【返回值】 如果字符串s1等于s2或者字符长度n等于0,则返回0;否则,返回非零值。

【例4-1】 下面的示例演示了bcmp()函数的使用,在程序中动态地更改内存数据段空间,代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <string.h> //加入字符串库
void main() {
    char *s1="Golden Global View"; //声明要比较的字符串
    char *s2="Golden global view"; //声明要比较的字符串
    clrscr(); //清屏
    if(!bcmp(s1,s2,7)) //比较字符串
        printf("s1 equal to s2 in first 7 bytes");
        //根据比较结果输出提示信息
    else printf("s1 not equal to s2 in first 7 bytes");
    if(!bcmp(s1,s2,12)) //比较字符串
        printf("s1 equal to s2 in first 12 bytes");
        //根据比较结果输出提示信息
```

```
else printf("s1 not equal to s2 in first 12 bytes");
getchar( );
```

```
}
```

运行上述程序, 首先声明要进行比较的字符串, 清除屏幕信息, 然后比较两个字符串的前7个字符, 并根据比较结果输出提示信息。接着, 比较两个字符串的前12个字符, 并根据比较结果输出相应提示信息。

注意: `bcmp()`函数不检查 `NULL`。

4.2 复制字符串函数: `bcopy()`

函数 `bcopy()` 用于复制指定长度的字符串, 其原型如下。

```
void bcopy(const void *src, void *dest, int n);
```

【参数】参数 `src` 为要复制的源字符串; 参数 `dest` 为复制的目标字符串; 参数 `n` 为要复制的字符串长度。

【返回值】该函数不检查字符串中的空字节 `NULL`, 函数没有返回值。

【例 4-2】下面的示例演示了 `bcopy()` 函数的使用, 使用该函数将源字符串指定长度的字符复制到目标字符串中, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <string.h> //加入字符串库
#define bcopy(a,b,c) memcpy(a,b,c)
void main( ){
    char *s="Golden Global View"; //声明字符串
    int i ;
    char d[20]; //声明字符数组
    clrscr( ); //清屏
    bcopy(s,d,6); //复制指定字符串
    printf("s: %s\n",s); //输出源字符串
    printf("d: %s\n",d); //输出目标字符数组
    getchar( ); //等待输入任意键
    s[13]=0;
```

```

    bcopy(s+7,d,11);           //复制字符串
    printf("%s\n",s+7);        //输出字符串
    for(i=0;i<11;i++) putchar(d[i]); //在控制台输出字符
    getchar( );
}

```

运行上述代码，首先声明需要进行复制的字符串，清除屏幕，然后复制指定字符串，输出源字符串和目标字符串。待输入任意键后清除屏幕，再次进行复制指定长度的字符串，输出源字符串和目标字符数组到控制台。

4.3 将字符串指定字节置零函数：bzero()

函数 bzero()用于置字符串 s 的前 n 个字节为 0，其原型如下。

```
void bzero(void *s, int n);
```

【参数】 参数s为要设置的字符串；参数n为要设置的字符数。

【返回值】 该函数没有返回值。

【例 4-3】 下面的示例演示了bzero()函数的使用，将指定字符串指定字节置 0，代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <string.h>          //加入字符串库
void main( ){
    Struct{                  //声明结构体
        int a;
        char s[5];
        float f;
    } tt;
    char s[20];              //声明字符数组
    bzero(&tt,sizeof(tt));   //结构体初始化为 0
    bzero(s,20);             //字符数组初始化为 0
    clrscr( );               //清屏
    printf("Initail Success"); //输出提示信息
}

```



```
getchar( );
```

运行上述代码后, 首先声明要处理的结构体和字符数组, 然后将结构体和字符数组初始化为 0, 清除屏幕, 并输出提示信息。

4.4 把内存区域的字节设置成字符函数:

setmem()

函数 setmem() 用于把内存区域的字符设置为字符, 其原型如下。

```
void setmem(void *addr, int len, char value);
```

【参数】参数addr为要设置的内存区域; 参数len为要设置的长度; 参数value为要设置的值。

【返回值】该函数没有返回值。

【例 4-4】下面的示例演示了 setmem() 函数的使用, 使用该函数将内存区域的字节设置成字符, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <alloc.h>           //加入动态内存管理库
#include <mem.h>             //加入内存管理库
void main( ){
    char *dest;               //声明要设置的目标区域
    dest = calloc(21, sizeof(char)); //分配内存
    setmem(dest, 20, 'c');    //设置为字符
    printf("%s\n", dest);    //输出
}
```

运行上述代码, 首先声明要设置的内存区域变量, 并分配内存空间, 然后将该内存区域设置为字符, 再输出设置后的内存区域字符。

4.5 从源字符中移动字节到目标字符函数:

movmem()

函数 `movmem()` 用于从源字符中移动字节到目标字符, 其原型如下。

```
void movmem(void *src, void *dest, unsigned int count);
```

【参数】 参数 `src` 为要移动的源区域; 参数 `dest` 为目标区域; 参数 `count` 为要移动的总数。

【返回值】 该函数没有返回值。

【例 4-5】 下面的示例演示了 `movmem()` 函数的使用, 采用该函数从源字符中移动字节到目标字符, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <string.h>           //加入字符串库
void main( ){
    char *s="Golden Global View"; //声明要处理的字符串
    movmem(s,s+7,strlen(s)-7);    //移动字符
    s[strlen(s)-7]=0;
    printf("%s",s);              //输出字符串
}
```

运行上述代码, 首先声明要处理的字符串, 清除屏幕, 然后移动指定字节数字符, 再将移动后的字符输出到控制台。

4.6 把字符串复制到数组函数: stpcpy()

函数 `stpcpy()` 用于把字符串复制到数组, 其原型如下。

```
char *stpcpy(char *destin, char *source);
```

【参数】 参数 `destin` 为目标数组; 参数 `source` 为要复制的字符串。

【返回值】该函数执行后返回字符数组的地址。

【例 4-6】下面的示例演示了 `strcpy()` 函数的使用, 在程序中将指定字符串复制到数组, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <string.h> //加入字符串库
void main( ){
    char string[10]; //声明字符数组
    char *str1 = "abcdefghi"; //声明字符串
    strcpy(string, str1); //复制字符串到数组
    printf("%s\n", string); //输出字符数组
}
```

运行上述程序, 首先声明要复制的字符数组和字符串, 然后将字符串复制到字符数组中, 再输出复制后的字符数组到控制台。

4.7 复制字符串到数组函数: `strcpy()`

函数 `strcpy()` 用于将字符串复制到数组, 其原型如下。

```
char *strcpy(char *dest, char *src);
```

【参数】参数 `dest` 为要复制的目标字符数组; 参数 `src` 为复制的源字符串。

【返回值】该函数执行后返回指向 `dest` 的指针。

【例 4-7】下面的示例演示了 `strcpy()` 函数的使用, 在程序中将指定字符串复制到数组, 其代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <string.h> //加入字符串库
void main( ){
    char *s="Golden Global View"; //声明要处理的字符串
    char d[20]; //声明目标字符数组
    strcpy(d, s); //复制字符串
    printf("%s", d); //输出字符数组
}
```


}

运行上述程序,首先声明要处理的字符串和字符数组,清除屏幕,然后复制指定字符串到字符数组,再将该字符数组输出到控制台。

4.8 字符串追加函数: strcat()

函数 strcat()用于实现字符串的追加。其原型如下。

```
char *strcat(char *destin, char *source);
```

【参数】参数destin为追加的目标字符串;参数source为追加的字符串。

【返回值】该函数追加字符后返回目标字符串的地址。

【例 4-8】下面的示例演示了strcat()函数的使用,在程序中向指定的字符串追加字符串,其代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    char destination[25];      //声明目标字符数组
    char *blank = " ", *c = "C++", *Borland = "Borland";
                                //声明字符串
    strcpy(destination, Borland); //追加字符串
    strcat(destination, blank);   //追加字符串
    strcat(destination, c);       //追加字符串
    printf("%s\n", destination); //输出目标字符数组
}
```

运行上述程序,首先声明目标字符数组和要追加的字符串,然后依次向目标字符数组追加字符串,最后将追加完成的字符数组输出到控制台。

4.9 查找字符串首次出现位置函数: strchr()

函数 `strchr()` 用于查找字符串首次出现的位置, 其原型如下。

```
char *strchr(char *str, char c);
```

【参数】 参数 `str` 为要查找的目标字符串; 参数 `c` 为要查找的字符。

【返回值】 该函数查找到字符后返回首次出现的字符位置; 否则, 返回 `-1`。

【例 4-9】 下面的示例演示了 `strchr()` 函数的使用, 查找目标字符串中字符首次出现的位置, 代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
void main() {
    char string[15];

    char *ptr, c = 'r';       //声明字符串数组

    strcpy(string, "This is a string"); //声明要查找的字符
    ptr = strchr(string, c);    //追加字符串
    if (ptr)                  //查找字符串
        printf("The character %c is at position: %d\n", c, ptr-string); //根据结果输出提示信息
    else printf("The character was not found\n");
}
```

运行上述程序, 首先声明要查找的字符数组和字符串, 然后将目标字符数组追加字符串, 再查找指定的字符, 并根据查找结果输出相应的提示信息。

4.10 字符串比较函数: strcmp()

函数 `strcmp()` 用于实现字符串的比较, 其原型如下。

```
int strcmp(char *str1, char *str2);
```

【参数】 参数str1 和参数str2 为要比较的字符串。

【返回值】 当字符串str1 小于str2 时，返回值小于 0；当字符串str1 等于str2 时，返回值为 0；当字符串str1 大于str2 时，返回值大于 0。

【例 4-10】 下面的示例演示了strcmp()函数的使用，在程序中比较两字符串，代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
void main( ) {
    char *buf1 = "aaa", *buf2 = "bbb", *buf3 = "ccc";
    //声明要比较的字符串
    int ptr;                   //声明变量
    ptr = strcmp(buf2, buf1);  //比较字符串
    if (ptr > 0)               //根据比较结果输出提示信息
        printf("buffer 2 is greater than buffer 1\n");
    else printf("buffer 2 is less than buffer 1\n");
    ptr = strcmp(buf2, buf3);  //比较字符串
    if (ptr > 0)               //根据比较结果输出提示信息
        printf("buffer 2 is greater than buffer 3\n");
    else printf("buffer 2 is less than buffer 3\n");
}
```

运行上述程序，首先声明要进行比较的字符串和用于暂存比较结果的变量，然后进行字符串的比较，并根据字符串比较结果输出相应的提示信息。

4.11 字符串比较（不区分大小写）函数：

stricmp()

函数 stricmp() 以不区分大小写的方式进行字符串比较，其原型如下。


```
int stricmp(char *str1, char *str2);
```

【参数】参数str1 和str2 为要进行比较的字符串。

【返回值】当字符串str1 小于str2 时, 返回值小于 0; 当字符串str1 等于str2 时, 返回值为 0; 当字符串str1 大于str2 时, 返回值大于 0。

【例 4-11】下面的示例演示了stricmp()函数的使用, 在程序中以不区分大小写的方式进行字符串的比较, 代码如下。

```
#include <string.h>                                //加入字符串库
#include <stdio.h>                                    //加入标准输入输出库
void main( ){
    char *buf1 = "BBB", *buf2 = "bbb";              //声明要进行比较的字符串
    int ptr;                                          //声明变量
    ptr = stricmp(buf2, buf1);                        //比较字符串
    if (ptr > 0)                                     //根据比较结果输出提示信息
        printf("buffer 2 is greater than buffer 1\n");
    if (ptr < 0) printf("buffer 2 is less than buffer 1\n");
    if (ptr == 0) printf("buffer 2 equals buffer 1\n");
}
```

运行上述程序, 首先声明要进行比较的字符串和暂存比较结果的变量, 然后进行字符串的比较, 并根据比较结果输出相应的提示信息。

4.12 字符串比较（不区分大小写）函数：

strcmpi()

函数 strcmpi()以不区分大小写的方式进行字符串的比较, 其原型如下。

```
int strcmpi(char *str1, char *str2);
```

【参数】参数str1 和str2 为要进行比较的字符串。

【返回值】当字符串str1 小于str2 时, 返回值小于 0; 当字符串str1 等于str2 时, 返回值为 0; 当字符串str1 大于str2 时, 返回值大于 0。

【例 4-12】下面的示例演示了 `strcmpi()` 函数的使用，在程序中以不区分大小写的方式进行字符串的比较，代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    char *buf1 = "BBB", *buf2 = "bbb";    //声明要进行比较的字符串
    int ptr;
    ptr = strcmpi(buf2, buf1);            //比较字符串
    if (ptr > 0)                          //根据比较结果输出提示信息
        printf("buffer 2 is greater than buffer 1\n");
    if (ptr < 0) printf("buffer 2 is less than buffer 1\n");
    if (ptr == 0) printf("buffer 2 equals buffer 1\n");
}
```

运行上述程序，首先声明要进行比较的字符串，然后以不区分大小的方式进行字符串比较，并根据比较结果输出相应的提示信息。

4.13 字符串查找函数： `strcspn()`

函数 `strcspn()` 用于实现字符串的查找，其原型如下。

```
int strcspn(char *str1, char *str2);
```

【参数】参数 `str1` 为要查找的字符串；参数 `str2` 为查找结果的字符串。

【返回值】该函数执行后返回字符串 `str1` 开头连续不含字符串 `str2` 内的字符数目。

【例 4-13】下面的示例演示了 `strcspn()` 函数的使用，使用该函数进行字符串的查找，代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <string.h>          //加入字符串库
#include <alloc.h>           //加入内存管理库
void main( ){
    char *string1 = "1234567890";        //声明要查找的字符串
```

```

char *string2 = "747DC8";           //声明结束字符串
int length;
length = strcspn(string1, string2); //字符串查找
printf("Character where strings intersect is at position %d\n",
length);                             //输出查找结果
}

```

运行上述代码, 首先声明要进行比较的字符串和结束字符串, 然后进行字符串的查找, 并输出字符串查找结果。

4.14 字符串复制函数: strdup()

函数 `strdup()` 用于实现字符串的复制, 其原型如下。

```
char *strdup(char *str);
```

【参数】 参数 `str` 为复制的字符串。

【返回值】 函数执行后返回字符串 `str` 的一个副本。

【例 4-14】 下面的示例演示了 `strdup()` 函数的使用, 采用该函数返回字符串的一个副本, 代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <string.h>          //加入字符串库
#include <alloc.h>           //加入内存管理库
void main( ) {
    char *dup_str, *string = "abcde"; //声明要复制的字符串
    dup_str = strdup(string);         //复制字符串
    printf("%s\n", dup_str);          //输出复制的字符串
    free(dup_str);                   //释放复制的字符串
}

```

运行上述代码, 首先声明要复制的字符串, 然后进行字符串复制, 并输出复制的字符串, 最后释放复制的字符串。

4.15 字符串长度函数: strlen()

函数 strlen() 用于计算字符串长度, 其原型如下。

```
int strlen(char *s);
```

【参数】 参数s表示要计算长度的字符串。

【返回值】 该函数返回给定字符串的长度。

【例 4-15】 下面的示例演示了 strlen() 函数的使用, 采用该函数返回给定字符串长度, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <string.h>          //加入字符串库
#include <conio.h>            //加入控制台输入输出库
void main( ){
    char *s="Golden Global View"; //声明字符串
    clrscr( );                 //清屏
    printf("%s has %d chars",s,strlen(s)); //输出字符串长度
    getch( );                 //等待输入
}
```

运行上述程序, 首先声明需要计算字符串长度的字符串变量, 清除屏幕, 然后计算并输出字符串长度。

4.16 将字符串转换为小写形式函数: strlwr()

函数 strlwr() 用于将字符串转换为小写形式, 其原型如下。

```
char *strlwr(char *s);
```

【参数】 参数s为要转换的字符串。

【返回值】 该函数返回转换为小写形式的字符串。

【例 4-16】 下面的示例演示了 strlwr() 函数的使用, 将给定的字符串转换为小写形式, 代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <string.h>          //加入字符串库
#include <conio.h>           //加入控制台输入输出库
void main( ){
    char *s="Copywrite 1999-2000 GGV Technologies";//声明字符串
    clrscr( );              //清屏
    printf("%s",strlwr(s));  //输出转换后的字符串
    getchar( );             //等待输入
}

```

运行上述程序,首先声明要转换的字符串,清除屏幕,然后输出转换为小写的字符串,等待用户按任意键后结束程序。

4.17 将字符串转换为大写形式函数:strupr()

函数strupr()用于将字符串转换为大写形式,其原型如下。

```
char *strupr(char *s);
```

【参数】参数s表示要进行转换的字符。

【返回值】该函数返回转换为大写形式的字符串。

【例 4-17】下面的示例演示了strupr()函数的使用,将给定的字符串转换为大写形式,代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <string.h>          //加入字符串库
#include <conio.h>           //加入控制台输入输出库
void main( ){
    char *s="Copywrite 1999-2000 GGV Technologies";//声明要转换的字符串
    clrscr( );              //清屏
    printf("%s",strupr(s)); //输出转换后的字符串
    getchar( );
}

```

运行上述程序,首先声明需要转换为大写的字符串,清除屏幕,然后将给定的字符串转换为大写并输出到控制台。

4.18 字符串尾部追加函数: `strncat()`

函数 `strncat()` 实现在字符串尾部追加字符串, 其原型如下。

```
char *strncat(char *dest, char *src, int n);
```

【参数】 参数 `dest` 为追加目标字符串; 参数 `src` 为要追加的源字符串; 参数 `n` 为追加的字符串个数。

【返回值】 该函数执行后返回追加后的字符串。

【例 4-18】 下面的示例演示了 `strncat()` 函数的使用, 将指定字符串追加到现有字符串的尾部, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <string.h>          //加入字符串库
#include <conio.h>            //加入控制台输入输出库
void main( ){
    char d[20]="Golden Global"; //声明要追加字符串的目标字符串
    char *s=" View WinIDE Library"; //源字符串
    clrscr( );                //清屏
    strncat(d,s,5);           //追加指定字符串
    printf("%s",d);           //输出追加后的字符串
}
```

运行上述程序, 声明要追加的目标字符串和源字符串, 清除屏幕, 然后追加指定长度的字符串, 并将其输出到控制台。

4.19 字符串比较函数: `strncmp()`

函数 `strncmp()` 用于实现字符串的比较, 其原型如下。

```
int strncmp(char *s1, char *s2, int n);
```

【参数】 参数 `s1` 和 `s2` 为要比较的字符串; 参数 `n` 为比较的字符串个数。

【返回值】 当字符串 `s1` 小于 `s2` 时, 返回值小于 0; 当字符串 `s1` 等于 `s2`

时,返回值等于0;当字符串s1大于s2时,返回值大于0。

【例 4-19】下面的示例演示了strncmp()函数的使用,采用该函数对指定字符串的前N个字符进行比较,代码如下。

```
#include <string.h>                //加入字符串库
#include <stdio.h>                //加入标准输入输出库
int main(void){
    char *buf1 = "aaabbb", *buf2 = "bbbccc", *buf3 = "ccc";
                                //声明要比较的字符串

    int ptr;
    ptr = strncmp(buf2,buf1,3);    //比较字符串前3个字符
    if (ptr > 0)                  //根据结果输出相应信息
        printf("buffer 2 is greater than buffer 1\n");
    else printf("buffer 2 is less than buffer 1\n");
    ptr = strncmp(buf2,buf3,3);    //比较字符串前3个字符
    if (ptr > 0)                  //根据结果输出相应信息
        printf("buffer 2 is greater than buffer 3\n");
    else printf("buffer 2 is less than buffer 3\n");
}
```

运行上述程序,首先声明需要进行比较的字符串,然后依次进行字符串,并根据比较结果输出相应的提示信息。

4.20 字符串比较(不区分大小写)函数:

strnicmp()

函数 strnicmp()以不区分大小写的方式进行字符串比较,其原型如下。

```
int strnicmp(char *str1, char *str2, unsigned maxlen);
```

【参数】参数str1和str2为要进行比较的字符串;参数maxlen为要比较的字符串长度。

【返回值】当字符串str1 小于str2 时, 返回值小于 0; 当字符串str1 等于str2 时, 返回值等于 0; 当字符串str1 大于str2 时, 返回值大于 0。

【例 4-20】下面的示例演示了strnicmp()函数的使用, 采用该函数以不区分大小写方式比较给定字符串中指定长度的字符, 代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    char *buf1 = "BBBccc", *buf2 = "bbbccc"; //声明要比较的字符串
    int ptr;
    ptr = strnicmp(buf2, buf1, 3);           //比较字符串
    if (ptr > 0)                             //根据比较结果输出相应提示信息
        printf("buffer 2 is greater than buffer 1\n");
    if (ptr < 0) printf("buffer 2 is less than buffer 1\n");
    if (ptr == 0) printf("buffer 2 equals buffer 1\n");
}
```

运行上述程序, 首先声明需要进行比较的字符串, 然后以不区分大小写方式比较指定长度字符, 再根据比较结果输出相应的提示信息。

4.21 字符串比较（不区分大小写）函数：

strncmpi()

函数 strncmpi()以不区分大小写的方式进行字符串的比较, 其原型如下。

```
int strncmpi(char *str1, char *str2, unsigned maxlen);
```

【参数】参数str1 和str2 为要进行比较的字符串; 参数maxlen为要比较的字符串长度。

【返回值】当字符串str1 小于str2 时, 返回值小于 0; 当字符串str1 等于str2 时, 返回值等于 0; 当字符串str1 大于str2 时, 返回值大于 0。

【例 4-21】下面的示例演示了strncmpi()函数的使用, 在程序中以不

区分大小写的方式进行字符串的比较,代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    char *s1="Hello, Programmers!";    //声明要比较的字符串
    char *s2="Hello, programmers!";    //声明要比较的字符串
    int r;
    r=strncmp(s1,s2,strlen(s1));        //比较字符串
    if(!r)    printf("s1 and s2 are identical");
                                           //根据结果输出相应提示信息
    else
    if(r<0)    printf("s1 less than s2");
    else    printf("s1 greater than s2");
}
```

运行上述程序,首先声明需要进行比较的字符串,然后以不区分大小写的方式进行字符串的比较,并根据比较结果输出相应的提示信息。

4.22 将字符串复制到数组函数: strncpy()

函数 `strncpy()` 将字符串复制到数组,其原型如下。

```
char *strncpy(char *destin, char *source, int maxlen);
```

【参数】 参数 `destin` 为目标数组; 参数 `source` 为要复制的字符串; 参数 `maxlen` 为要复制的字符串长度。

【返回值】 该函数没有返回值。

【例 4-22】 下面的示例演示了 `strncpy()` 函数的使用,使用该函数将指定长度字符串复制到指定数组中,代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <string.h>           //加入字符串库
void main( ){
    char string[10];           //声明字符数组
    char *str1 = "abcdefghi";  //声明字符串
```



```

    strncpy(string, str1, 3);           //复制字符串到数组
    string[3] = '\0';
    printf("%s\n", string);
}

```

运行上述代码，首先声明需要复制的字符串和目标字符数组，然后将给定字符串指定长度字符复制到目标字符数组中，并将该字符数组输出到控制台。

4.23 字符串查找函数：strpbrk()

函数 strpbrk() 用于实现字符串的查找，其原型如下。

```
char *strpbrk(char *str1, char *str2);
```

【参数】 参数 str1 为要进行查找的目标字符；参数 str2 为要查找的字符串。

【返回值】 该函数执行后，返回指向 str1 中第一个相匹配的字符的指针；如果没有匹配字符，则返回空指针 NULL。

【例 4-23】 下面的示例演示了 strpbrk() 函数的使用，使用该函数进行字符串的查找，并输出相应结果，代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <string.h>          //加入字符串库
void main() {
    char *string1 = "abcdefghijklmnopqrstuvwxyz"; //声明字符串
    char *string2 = "onm";                       //声明字符串
    char *ptr;
    ptr = strpbrk(string1, string2);              //进行字符串查找
    if (ptr) printf("strpbrk found first character: %c\n", *ptr);
                                                //根据结果输出提示信息
    else printf("strpbrk didn't find character in set\n");
}

```

运行上述代码，首先声明需要进行查找的目标字符串和要查找的字符串。

字符串, 然后进行字符串的查找, 并根据查找结果输出相应的提示信息。

4.24 字符串倒序函数: `strrev()`

函数 `strrev()` 用于实现字符串倒序操作, 其原型如下。

```
char *strrev(char *str);
```

【参数】 参数 `str` 为要进行操作的字符串。

【返回值】 该函数执行后, 返回指向颠倒顺序后的字符串指针。

【例 4-24】 下面的示例演示了 `strrev()` 函数的使用, 将给定字符串进行倒序操作, 代码如下。

```
#include <string.h> //加入字符串库
#include <stdio.h> //加入标准输入输出库
void main( ){
    char *forward = "string"; //声明需要处理的字符串
    printf("Before strrev( ): %s\n", forward); //输出倒序前的字符串
    strrev(forward); //倒序
    printf("After strrev( ): %s\n", forward); //输出倒序后的字符串
}
```

运行上述程序, 首先声明需要进行处理字符串, 输出倒序前的字符串, 然后对该字符串进行倒序操作, 再输出倒序后的字符串。

4.25 将字符串设置成指定字符函数: `strset()`

函数 `strset()` 将字符串设置为指定字符, 其原型如下。

```
char *strset(char *str, char c);
```

【参数】 参数 `str` 为设置的目标字符串; 参数 `c` 为设置的字符。

【返回值】 该函数执行后, 返回指向 `s` 的指针。

【例 4-25】下面的示例演示了 `strset()` 函数的使用，将指定字符串设置为指定字符，代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <string.h>          //加入字符串库
void main( ){
    char string[10] = "123456789"; //声明要处理的字符数组
    char symbol = 'c';           //声明要设置的字符
    printf("Before strset( ): %s\n", string); //输出设置前的字符数组
    strset(string, symbol);      //设置字符
    printf("After strset( ): %s\n", string);  //输出设置后的字符数组
}
```

运行上述程序，首先声明要进行处理的字符数组和字符，输出该字符数组，然后将该字符数组设置为指定字符，再输出设置后的字符数组。

4.26 在字符串中查找指定字符首次出现位置

函数： `strstr()`

函数 `strstr()` 用于在字符串中查找指定字符首次出现的位置，其原型如下。

```
char *strstr(char *str1, char *str2);
```

【参数】参数 `str1` 为要查找的目标字符串；参数 `str2` 为查找的字符串。

【返回值】该函数执行后，返回指向第一次出现 `str2` 位置的指针。如果没找到，则返回 `NULL`。

【例 4-26】下面的示例演示了 `strstr()` 函数的使用，采用该函数在给定字符串中查找指定字符首次出现的位置，代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <string.h>          //加入字符串库
void main( ){
```



```

char *str1 = "Borland International", *str2 = "nation", *ptr;
//声明要查找的字符串
ptr = strstr(str1, str2); //查找字符串
printf("The substring is: %s\n", ptr); //输出查找到的字符串
}

```

运行上述程序,首先声明要查找的目标字符串和要查找的子字符串,然后进行字符串查找,再输出查找到的字符串。

4.27 用分隔符分解字符串函数: strtok()

函数 strtok()实现用分隔符分解字符串,其原型如下。

```
char *strtok(char *str1, char *str2);
```

【参数】参数str1 为要分解的字符串; str2 为分隔符字符串。

【返回值】该函数在str1 中查找包含在str2 中的字符并用NULL ('\0')来替换,直到找遍整个字符串,返回指向下一个标记串;当没有标记串时,则返回空字符串NULL。

【例 4-27】下面的示例演示了strtok()函数的使用,采用该函数用分隔符分解指定字符串,代码如下。

```

#include <string.h> //加入字符串库
#include <stdio.h> //加入标准输入输出库
void main( ){
    char input[16] = "abc,d"; //声明要分解的字符串
    char *p;
    p = strtok(input, ","); //分解字符串
    if (p) printf("%s\n", p); //根据结果输出信息
    p = strtok(NULL, ","); //分解字符串
    if (p) printf("%s\n", p);
}

```

运行上述程序,首先声明要进行分解的字符串,然后分解指定字符串,并根据分解结果输出提示信息。

第5章 标准库函数库: system.h

在程序设计中,常常需要进行屏幕的相关操作。C语言专门提供了一个针对屏幕操作的函数库——system.h,该文件库中存在的常用函数主要包括屏幕及光标的控制操作函数。

5.1 清屏函数: clrscr()

函数 clrscr() 用于清除屏幕,其原型如下。

```
void clrscr(void);
```

【参数】该函数没有参数。

【返回值】该函数没有返回值。

【例 5-1】下面的示例演示了 clrscr() 函数的使用,在程序中清除屏幕,代码如下。

```
#include <conio.h>           //加入控制台输入输出库
#include <system.h>          //加入标准库
void main() {
    int i;
    clrscr();                 //清屏
    for (i = 0; i < 20; i++) cprintf("%d\r\n", i); //输出
    cprintf("\r\nPress any key to clear screen");
    getch();
    clrscr();                 //清除屏幕
    cprintf("The screen has been cleared!");      //输出
}
```

运行上述程序,首先清除屏幕上的文字,依次输出数字和提示信息,待按任意键后清除屏幕,然后再输出提示信息。

5.2 以指定模式刷新屏幕函数:

UpdateLCD()

函数 UpdateLCD() 以指定模式刷新屏幕, 其原型如下。

```
void UpdateLCD(unsigned int mode);
```

【参数】参数 mode 为设置的模式。

【返回值】该函数没有返回值。

【例 5-2】下面的示例演示了 UpdateLCD() 函数的使用, 采用该函数在程序中动态地刷新屏幕, 代码如下。

```
#include <system.h>           //加入标准库
#include <stdio.h>             //加入标准输入输出库
void main( ){
    clrscr( );                 //清屏
    printf("Line 1\n");        //输出提示信息
    printf("Line 2\n");
    printf("Line 3\n");
    printf("Line 4\n");
    printf("Line 5\n");
    printf("Line 6\n");
    UpdateLCD(0x00);           //刷新所有行
    getchar( );                //从 stdin 流中读取字符
    UpdateLCD(0xE0);           //刷新屏幕
    getchar( );
    UpdateLCD(0x40);           //刷新屏幕
    getchar( );
    UpdateLCD(0x20);           //刷新屏幕
    getchar( );
}
```

运行上述代码, 首先清除当前屏幕, 然后输出相应信息, 再以特定模式刷新当前屏幕。

5.3 在屏幕上指定位置输出字符串函数:

TextOut()

函数 TextOut() 用于在屏幕上指定位置输出字符串, 其原型如下。

```
void TextOut(int x, int y, char *str, int mode);
```

【参数】 参数 x 和参数 y 分别表示屏幕的像素位置; 参数 str 为输出的字符串; 参数 mode 为输出的模式。其中, mode 取值为 0, 表示背景透明, 点阵中为 0 的点不显示; 值为 1 表示正常方式, 即点阵中为 0 的点擦除, 为 1 的点显示; 值为 2 表示反显方式, 即点阵中为 1 的点擦除, 为 0 的点显示; 值为 3 表示异或方式, 即点阵中点的值和屏幕当前位置的值作异或后取值, 为 0 则擦除, 为 1 则显示。

【返回值】 该函数没有返回值。

【例 5-3】 下面的示例演示了 TextOut() 函数的使用, 采用该函数在屏幕指定位置输出字符串, 代码如下。

```
#include <system.h>                //加入标准库
void main( ){
    clrscr( );                       //清屏
    TextOut(0,0,"文曲星",1);         //在屏幕左上角输出字符串
    TextOut(10,10,"文曲星",0);       //在第十行第十列输出
    TextOut(20,20,"您好",2);         //在第二十行第二十列输出
    TextOut(30,30,"GGV-金远见",3);   //在第三十行第三十列输出
}
```

运行上述程序, 首先清屏, 然后依次在屏幕左上角、第十行第十列、第二十行第二十列、第三十行第三十列以不同模式输出给定的字符。

5.4 响铃函数: bell()

函数 bell() 用于实现响铃, 其原型如下。

```
void bell(void);
```

【参数】该函数没有参数。

【返回值】该函数没有返回值。

【例 5-4】下面的示例演示了bell()函数的使用,代码如下。

```
#include <system.h> //加入标准库
#include <stdio.h> //加入标准输入输出库
void main( ){
    int c;
    printf("press any key\n"); //输出提示信息
    c=getchar( ); //输入字符
    while(c!=0x1b) {
        bell( ); //响铃
        c=getchar( );
    }
    printf("Thank you!"); //输出提示信息
    getchar( );
}
```

运行上述程序,首先清除屏幕,然后输出提示信息,等用户按下任意键后响铃,直到按【Esc】键退出。

5.5 在屏幕上画一矩形并填充函数: block()

函数 block()用于在屏幕上画一矩形,并填充,其原型如下。

```
void block(int left, int top, int right, int bottom, int mode);
```

【参数】参数 left 和 top 用于指定左上角坐标;参数 right 和 bottom 用于指定右下角坐标;参数 mode 为画图方式。其中,mode 的值为 0 表示清除方式;值为 1 表示正常方式;值为 2 表示反相方式。

【返回值】该函数没有返回值。

【例 5-5】下面的示例演示了block()函数的使用,代码如下。

```
#include <system.h> //加入标准库
```

```

#include <stdio.h> //加入标准输入输出库
void main( ){
    int c;
    gotoxy(10,10); //隐藏光标
    block(1,0,111,47,1); //画一矩形
    getchar( );
    block(20,10,100,30,0); //画另一矩形
    getchar( );
    block(40,0,80,47,2); //画另一矩形
}

```

运行上述程序，首先清屏，并隐藏光标，然后画一个矩形，等按下任意键后再画第二个矩形，然后按下任意键后再画第三个矩形。

5.6 设定光标形态函数：cursor()

函数 cursor() 用于设置光标形态，其原型如下。

```
void cursor(int mode);
```

【参数】 参数 mode 为要设定的光标模式，其值为 0x00 表示块状光标（默认）；值为 0x80 表示下划线光标，其他值无意义。

【返回值】 该函数没有返回值。

【例 5-6】 下面的示例演示了 cursor() 函数的使用，采用该函数在程序中设定光标形态，代码如下。

```

#include <system.h> //加入标准库
#include <stdio.h> //加入标准输入输出库
void main( ){
    int c;
    cursor(0x00); //设置为块状光标
    printf("press key to change cursor");
    getchar( );
    cursor(0x80); //设置为下划线光标
}

```


运行上述程序, 首先清除屏幕, 将光标设置为块状光标, 然后待按下任意键后将光标设置为下划线光标。

5.7 短暂延时函数: delay()

函数 delay() 用于实现短暂延时, 其原型如下。

```
void delay(unsigned milliseconds);
```

【参数】 参数 milliseconds 为要延时时间。

【返回值】 该函数没有返回值。

【例 5-7】 下面的示例演示了 delay() 函数的使用, 在程序中实现短暂的延时, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <system.h> //加入标准库
void main() {
    int c;
    printf("\nHello, world!"); //输出提示信息
    delay(250); //延时 1 秒
    printf("\nHi, guys"); //输出提示信息
}
```

运行上述程序, 首先清除屏幕, 输出提示信息, 然后延时 1 秒, 清除屏幕, 输出提示信息。

5.8 取汉字的点阵函数: get_chi_font()

函数 get_chi_font() 用于把 str 所指向的汉字的 16*16 点阵信息存放在 buf 中, 其原型如下。

```
void get_chi_font(char *str, char *buf);
```

【参数】 参数 `str` 为要取汉字点阵的字符串；参数 `buf` 为要暂存点阵的缓冲区。

【返回值】 该函数没有返回值。

【例 5-8】 下面的示例演示了 `get_chi_font()` 函数的使用，采用该函数获取指定字符点阵信息，代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <system.h> //加入标准库
void main( ){
    char buf[32]; //声明缓冲区
    char *s="文曲星",*p;
    int i,j,k; //循环变量
    int x=0,y=0; //显示频率
    int r; //当前字节
    clrscr( ); //清屏
    gotoxy(10,10); //隐藏光标
    p=s;
    while(*p){
        get_chi_font(p,buf); //获取点阵
        for(i=0;i<2;i++){
            for(j=0;j<16;j++){
                r=buf[i*16+j];
                for(k=0;k<8;k++){
                    putpixel(x+i*8+k,y+j,((r<<k)&0x80)?1:0);
                }
            }
            p+=2; x+=16;
        }
        getchar( );
    }
}
```

运行上述代码，首先清除屏幕，获取光标，然后循环汉字点阵信息，并将其输出到控制台。

注意： `buf` 必须有足够的空间容纳返回的数据。

5.9 取英文字符的点阵函数: get_eng_font()

函数 get_eng_font() 用于取英文字符的点阵信息, 其原型如下。

```
void get_eng_font(char ch, char *buf, int mode);
```

【参数】 参数 ch 为要取点阵的字符; 参数 buf 为暂存点阵信息的缓冲区; 参数 mode 为所取模式, 为 0 表示取 8*16 点阵信息, 为 1 表示取 8*8 点阵信息。

【返回值】 该函数没有返回值。

【例 5-9】 下面的示例演示了 get_eng_font() 函数的使用, 采用该函数取英文字符的点阵信息, 代码如下。

```
#include <system.h> //加入标准库
void main( ){
    char buf[32]; char *s="Global View",*p;
    int i,j,k; //循环变量
    int x=0,y=0; int r;
    gotoxy(10,10); //隐藏光标
    x=y=0; p=s;
    while(*p) {
        get_eng_font(*p,buf,0); //循环取英文字符光标

        for(j=0;j<16;j++) {
            r=buf[j];
            for(k=0;k<8;k++) putpixel(x+k,y+j,((r<<k)&0x80)?1:0); //以图像方式输出
        }
        p++;
        x+=8;
    }
    x=0,y=16; p=s;
    while(*p) {
        get_eng_font(*p,buf,1); //循环取英文字符光标
        for(j=0;j<8;j++) {
            r=buf[j];
```



```

        for(k=0;k<8;k++) putpixel(x+k,y+j,((r<<k)&0x80)?1:0);
                                                //以图像方式输出
    }
    p++;
    x+=8;
}
}

```

运行上述程序，首先清除屏幕，隐藏光标，然后循环获取英文字符光标，并以图像形式输出到控制台。

注意：buf 必须有足够的空间容纳返回的数据。

5.10 读键函数：getkey()

函数 getkey() 用于获取自键盘输入的字符，其原型如下。

```
int getkey(void);
```

【参数】该函数没有参数。

【返回值】该函数执行后，返回所输入字符的 ASCII 码。

【例 5-10】下面的示例演示了 getkey() 函数的使用，在程序中采用该函数获取自键盘输入的字符，代码如下。

```

#include <stdio.h>                                //加入标准输入输出库
#include <system.h>                                //加入标准库
void main( ){
    int c;
    printf("Press key...");                        //输出提示信息
    while((c=getkey())!='Q') {                    //循环输入
        clrscr();
        printf("key: %c\nvalue: %x",c,c);        //输出输入的键
    }
}

```

运行上述程序，首先清除屏幕，输出提示信息，获取循环输入任意键并输出至控制台，直到输入 **【Q】** 键退出。

5.11 将光标移动到指定位置函数: move()

函数 move() 用于将光标移动到指定位置, 实际上该函数是 gotoxy() 函数的宏, 其原型如下。

```
void gotoxy(int x, int y);
```

【参数】 参数x和y分别表示要移动目标位置的列和行。

【返回值】 该函数没有返回值。

【例 5-11】 下面的示例演示了 move() 函数的使用, 在程序中将光标移动到指定位置, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
#include <system.h>                                //加入标准库
void main( ){
    int i;
    char *scrbuf=(char *)0x280;
    for(i=0;i<85;i++) scrbuf[i]='+';              //循环设置缓冲区数组的值
    UpdateLCD(0x00);                               //刷新屏幕
    gotoxy(2,2);                                    //移动到第三行第三列
    getchar( );                                     //获取字符
    move(2,2);                                       //移动到第二行第二列
}
```

运行上述程序, 首先刷新屏幕, 将光标移动到第三行第三列, 待输入任意键后, 将光标移动到第二行二列。

5.12 调用系统例程函数: noidle()

函数 noidle() 用于调用系统例程, 其原型如下。

```
void noidle(void);
```

【参数】 该函数没有参数。

【返回值】该函数没有返回值。

【例 5-12】下面的示例演示了noidle()函数的使用,使用该函数调用系统例程,代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <system.h> //加入标准库
Void main( ){
    int i,j;
    printf("Waiting...\n"); //输出提示信息
    for(i=0;i<10;i++)
        for(j=0;j<32767;j++) noidle( ); //调用系统例程
    printf("System Shutdown?");
}
```

运行上述代码,首先清除屏幕,输出提示信息,然后进行大数的循环,在循环中调用系统例程来防止系统关机。

注意:当程序运算时间过长(一般为循环过程)时,请调用 noidle() 函数来防止系统关机。

5.13 查询汉字拼音: pyfc()

函数 pyfc()用于查询给定汉字的拼音,其原型如下。

```
char *pyfc(char *str);
```

【参数】参数str为要查询拼音的汉字。

【返回值】该函数执行后返回一个指向对应拼音字符串,多音字的几个读音之间用空格分隔。

【例 5-13】下面的示例演示了pyfc()函数的使用,在程序中查询给定汉字的拼音,代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <system.h> //加入标准库
#include <string.h> //加入字符串库
```



```

void main( ){
    char *s1,*s2,*s3, *x,*y,*z;
    x=pyfc("文");           //取汉字拼音
    s1=strdup(x);
    y=pyfc("曲");           //取汉字拼音
    s2=strdup(y);
    z=pyfc("星");           //取汉字拼音
    s3=strdup(z); printf("文曲星\n");
    printf("%s%s%s",s1,s2,s3); //输出拼音
    getchar( );
    while(*s2++!=' ');
    printf("文曲星\n"); printf("%s%s%s",s1,s2,s3);
}

```

运行上述代码,首先声明用于暂存拼音的指针变量,然后依次取汉字拼音,依次输出汉字拼音。

5.14 写汉字点阵函数: write_chi_font()

函数 write_chi_font()用于写汉字点阵,其原型如下。

```
void write_chi_font(int x,int y,char *buf);
```

【参数】 参数x和参数y为屏幕坐标; 参数buf为写的点阵信息。

【返回值】 该函数没有返回值。

【例 5-14】 下面的示例演示了write_chi_font()函数的使用,采用该函数将汉字点阵信息输出到指定位置,代码如下。

```

#include <system.h>           //加入标准库
void main( ){
    char buf[32];              //声明缓冲区
    char *s="文曲星",*p;      //声明变量
    int x=0,y=0;
    clrscr( );                 //清屏
    gotoxy(10,10);             //隐藏光标

```

```

p=s;
while(*p) {
    get_chi_font(p,buf);           //取汉字点阵信息
    write_chi_font(x,y,buf);       //输出点阵信息
    p+=2;    x+=16;
}
}

```

运行上述程序，首先清除屏幕，隐藏光标，然后循环取指定字符汉字点阵信息，再将该汉字点阵信息输出到指定位置。

5.15 写英文字符的点阵函数: write_eng_font()

函数 write_eng_font() 用于写英文字符点阵信息，其原型如下。

```
void write_eng_font(int x,int y,char *buf,int mode);
```

【参数】 参数x和参数y为屏幕坐标。

【返回值】 该函数没有返回值。

【例 5-15】 下面的示例演示了 write_eng_font() 函数的使用，采用该函数写英文字符点阵信息，代码如下。

```

#include <system.h>           //加入标准库
void main() {
    char buf[32]; char *s="Global View",*p; int x=0,y=0;
    gotoxy(10,10);           //隐藏光标
    x=y=0; p=s;
    while(*p) {
        get_eng_font(*p,buf,0);           //获取英文字符点阵信息
        write_eng_font(x,y,buf,0);       //写英文字符点阵信息
        p++;    x++;
    }
    x=0,y=2; p=s;
    while(*p) {
        get_eng_font(*p,buf,1);           //获取英文字符点阵信息

```

```

        write_eng_font(x,y,buf,1);           //写英文字符点阵信息
        p++; x++;
    }
}

```

运行上述程序, 首先清除屏幕, 隐藏光标, 然后循环获取英文字符点阵信息, 再将英文字符点阵信息写到指定位置。

5.16 显示七段数码管数字函数: DispBCD()

函数 DispBCD() 用于显示七段数码管数字, 其原型如下。

```
void DispBCD(int);
```

【参数】 参数int为要显示的数字。

【返回值】 该函数没有返回值。

【例 5-16】 下面的示例演示了DispBCD()函数的使用, 在程序中动态地显示七段数码管数字, 代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <system.h>         //加入标准库
void main( ){
    int i;
    i=9;
    clrscr( );               //清屏
    printf("Now Display 9");
    DispBCD(9);              //显示七段数码管数字
    getchar( );
    i=99;
    clrscr( );               //清屏
    printf("Now Display 99");
    DispBCD(99);             //显示七段数码管数字
    getchar( );
    i=999;
    clrscr( );               //清屏
    printf("Now Display 99");
}

```



```
DispBCD(999); //显示七段数码管数字
}
```

运行上述程序，首先清除屏幕，输出提示信息，然后再输出七段数码管数字，待按下任意键后，输出其他七段数码管数字。

5.17 显示滚动条函数：SetScrollBar()

函数 SetScrollBar()用于显示滚动条，其原型如下。

```
void SetDispBar(int sum, int cur);
```

【参数】 参数sum为总长度；参数cur为当前位置。

【返回值】 函数调用后在屏幕左侧图标区显示百分比滚动条，没有返回值。

【例 5-17】 下面的示例演示了SetScrollBar()函数的使用，采用该函数显示滚动条，代码如下。

```
#include <system.h> //加入标准库
void main( ){
    int i;
    clrscr( ); //清屏
    printf("Scroll Bar Test"); //输出提示信息
    for(i=0;i<60;i+=5) {
        SetScrollBar(60,i); //设置滚动条
        sleep(1); //暂停1秒钟
    }
}
```

运行上述代码，首先清除屏幕，输出提示信息，然后以循环的方式设置滚动条。

第6章 图形处理函数库: graphics.h

在程序设计中,常常需要进行图形处理等操作。C语言专门提供了一个针对图形处理操作的函数库——graphics.h,该文件库中存在的常用函数主要包括图形的绘制、填充,背景的设置等操作函数。

6.1 取得最后一次调用画弧线坐标

函数: getarccoords()

函数 getarccoords()用于取最后一次调用画弧线坐标,其原型如下。

```
void far getarccoords(struct arccoordstype far *arccoords);
```

【参数】参数arccoords为保存坐标值的结构体。

【返回值】该函数返回最后一次调用圆或椭圆函数的相应起终点与中心坐标值,并存放在coordsp指向的结构里。

【例6-1】下面的示例演示了getarccoords()函数的使用,采用该函数获取最后一次画弧线的坐标,代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    struct arccoordstype arcinfo;
    int midx, midy;
    int stangle = 45, endangle = 270;
    char sstr[80], estr[80];
```

```

initgraph(&gdriver, &gmode, ""); //初始化
errorcode = graphresult(); //读取初始化结果
if (errorcode != grOk) { //若出错, 输出提示信息, 退出
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
}

midx = getmaxx() / 2; //返回屏幕中心 x 坐标
midy = getmaxy() / 2; //返回屏幕中心 y 坐标
setcolor(getmaxcolor()); //设置当前画线颜色
arc(midx, midy, stangle, endangle, 100); //画弧线
getarccoords(&arcinfo); //取最后一次画弧线的坐标
sprintf(sstr, "%d %d", arcinfo.xstart, arcinfo.ystart);
//将坐标转换为字符串
sprintf(estr, "%d %d", arcinfo.xend, arcinfo.yend);
outtextxy(arcinfo.xstart, arcinfo.ystart, sstr);
//在指定位置输出
outtextxy(arcinfo.xend, arcinfo.yend, estr);
getch();
closegraph(); //关闭图形系统
}

```

运行上述程序, 首先声明相应的结构体和变量, 初始化图形系统, 若初始化有误, 输出提示信息并退出; 否则取屏幕中心坐标, 画弧线, 然后取画弧线的坐标, 将其坐标信息转换为字符串, 并在指定位置输出。

6.2 画弧线函数: arc()

函数 arc() 用于画弧线, 其原型如下。

```
void far arc(int x, int y, int stangle, int endangle, int radius);
```

【参数】 参数 x 和参数 y 为圆心坐标; 参数 stangle 为起始角; 参数

endangle为终止角; 参数radius为半径。

【返回值】 该函数没有返回值。

【例 6-2】 下面的示例演示了arc()函数的使用, 使用该函数画弧线, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int stangle = 45, endangle = 135;
    int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );                //取初始化结果
    midx = getmaxx( ) / 2;
    midy = getmaxy( ) / 2;
    setcolor(getmaxcolor( ));                  //设置颜色
    arc(midx, midy, stangle, endangle, radius); //画弧线
    getch( );
    closegraph( );                             //关闭图形系统
}
```

运行上述代码, 首先初始化图形系统, 然后取屏幕中心坐标, 设置颜色, 画弧线, 待按下任意键后关闭图形系统。

6.3 画圆函数: circle()

函数 circle()用来画圆, 其原型如下。

```
void far circle(int x, int y, int radius);
```

【参数】 参数x和参数y为圆心坐标; 参数radius为要画圆的半径。

【返回值】 该函数没有返回值。

【例 6-3】下面的示例演示了 circle() 函数的使用, 采用该函数在指定位置画圆, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    midx = getmaxx( ) / 2;
    midy = getmaxy( ) / 2;                     //取屏幕中心坐标
    setcolor(getmaxcolor( ));                 //设置颜色
    circle(midx, midy, radius);                //画圆
    getch( );
    closegraph( );                             //关闭图形系统
}
```

运行上述代码, 首先初始化图形系统; 然后获取屏幕中心坐标, 设置颜色, 在指定位置画圆, 待按下任意键后关闭图形系统。

6.4 绘制扇形并填充函数: pieslice()

函数 pieslice() 用于绘制扇形并填充, 其原型如下。

```
void pieslice(int x, int y, int startangle, int endangle, int radius);
```

【参数】 参数 x 和参数 y 为圆心坐标; 参数 startangle 为起始角; 参数 endangle 为终止角; 参数 radius 为扇区半径。

【返回值】 该函数没有返回值。

【例 6-4】下面的示例演示了 pieslice() 函数的使用, 在程序中采用该函数绘制扇形并填充, 代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int stangle = 45, endangle = 135, radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    midx = getmaxx( ) / 2;
    midy = getmaxy( ) / 2;
    setfillstyle(EMPTY_FILL, getmaxcolor( )); //设置填充风格颜色
    pieslice(midx, midy, stangle, endangle, radius); //画扇区
    getch( );
    closegraph( );                          //关闭图形系统
}

```

运行上述代码,首先初始化图形系统;然后获取屏幕中心坐标,设置填充风格和颜色,在指定位置画扇区,待按下任意键后关闭图形系统。

6.5 显示当前图形模式的纵横比函数:

getaspectratio()

函数 `getaspectratio()` 用于显示当前图形模式的纵横比,其原型如下。

```
void far getaspectratio(int far *xasp, int far *yasp);
```

【参数】 参数 `xasp` 指向的变量存放返回的 `x` 方向比例系数; 参数 `yasp` 指向的变量存放返回的 `y` 方向比例系数。

【返回值】 该函数没有返回值。

【例 6-5】下面的示例演示了 `getaspectratio()` 函数的使用，在程序中显示当前图形模式的纵横比，代码如下。

```
#include <graphics.h>           //加入图形处理库
void main()
{
    int xasp,yasp;               //声明变量
    float aspectratio;           //声明变量
    getaspectratio(&xasp,&yasp);  //取当前图形模式的纵横比
    aspectratio=xasp/yasp;       //计算纵横比
    printf("aspect ratio: %f",aspectratio); //输出纵横比
}
```

运行上述程序，首先声明用于暂存图形纵横比的变量，然后采用该函数获取当前图形纵横比，再计算并输出根据所获取的图形纵横比。

6.6 设置图形纵横比函数： `setaspectratio()`

函数 `setaspectratio()` 用于设置图形纵横比，其原型如下。

```
void far setaspectratio(int xasp, int yasp);
```

【参数】 参数 `xasp` 为要设置的 `x` 方向比例系数；参数 `yasp` 为要设置的 `y` 方向比例系数。

【返回值】 该函数没有返回值。

【例 6-6】下面的示例演示了 `setaspectratio()` 函数的使用，在程序中动态地设置图形纵横比，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
void main() {
    int gdriver = DETECT, gmode, errorcode;
```

```

int midx, midy, xasp, yasp; int radius = 100;
initgraph(&gdriver, &gmode, ""); //初始化图形系统
errorcode = graphresult();
midx = getmaxx() / 2;
midy = getmaxy() / 2; //取屏幕中心坐标
setcolor(getmaxcolor()); //设置颜色
getaspectratio(&xasp, &yasp); //取纵横比
circle(midx, midy, 100); //画圆
getch();
cleardevice(); //清除图形屏幕
setaspectratio(xasp/2, yasp); //设置纵横比
circle(midx, midy, 100); //画圆
getch();
cleardevice(); //清除图形屏幕
setaspectratio(xasp, yasp/2); //设置纵横比
circle(midx, midy, 100); //画圆
getch();
closegraph(); //关闭图形系统
}

```

运行上述程序, 首先初始化图形系统; 然后获取屏幕中心坐标, 设置填充风格和颜色, 获取当前图形模式纵横比, 画圆, 待按下任意键后, 重新设置纵横比, 再次重新画圆, 最后待按下任意键后再次设置纵横比, 重新画圆。

6.7 绘制并填充椭圆扇区函数: sector()

函数 sector() 用于绘制并填充椭圆扇区, 其原型如下。

```
void far sector(int x, int y, int stangle, int endangle);
```

【参数】 参数x和参数y为圆心坐标; 参数stangle为起始角; 参数endangle为终止角。

【返回值】 该函数没有返回值。

【例 6-7】下面的示例演示了 sector() 函数的使用，在程序中动态地绘制并填充椭圆扇区，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, i;   int radius = 100;
    int stangle = 45, endangle = 135;
    int xrad = 100, yrad = 50;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    midx = getmaxx( ) / 2;
    midy = getmaxy( ) / 2;                       //获取屏幕中心坐标
    for (i=EMPTY_FILL; i<USER_FILL; i++) {
        setfillstyle(i, getmaxcolor( ));        //设置填充风格
        sector(midx, midy, stangle, endangle, xrad, yrad);
    }
    getch( );
    closegraph( );                               //关闭图形系统
}
```

运行上述程序，首先初始化图形系统；然后获取屏幕中心坐标，循环设置填充风格和颜色，绘制并填充椭圆扇区，待按下任意键后退出程序。

6.8 绘制椭圆函数：ellipse()

函数 ellipse() 用于绘制椭圆，其原型如下。

```
void far ellipse(int x, int y, int stangle, int endangle, int xradius,
int yradius);
```


【参数】 参数x和参数y为圆心坐标; 参数stangle为起始角; 参数endangle为终止角; 参数xradius和参数yradius分别为椭圆的x轴半径与y轴半径。

【返回值】 该函数没有返回值。

【例 6-8】 下面的示例演示了ellipse()函数的使用, 在程序中动态地绘制椭圆, 其代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult( );
    midx = getmaxx( ) / 2;
    midy = getmaxy( ) / 2;           //获取屏幕中心坐标
    setcolor(getmaxcolor( ));        //设置颜色
    ellipse(midx, midy, stangle, endangle, xradius, yradius); //画椭圆
    getch( );
    closegraph( );                  //关闭图形系统
}
```

运行上述程序, 首先初始化图形系统; 然后获取屏幕中心坐标, 设置颜色, 在指定位置画一椭圆, 待按下任意键后关闭图形系统。

6.9 绘制并填充椭圆函数: fillellipse()

函数 fillellipse()用于绘制并填充椭圆, 其原型如下。

```
void far fillellipse(int x, int y, int xradius, int yradius);
```

【参数】 参数x和参数y为椭圆中心坐标; 参数xradius和参数yradius为

水平轴半径和垂直轴半径。

【返回值】该函数没有返回值。

【例 6-9】下面的示例演示了 `fillellipse()` 函数的使用，在程序中动态地绘制并填充椭圆，其代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int xcenter, ycenter, i;
    int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult();
    xcenter = getmaxx() / 2;
    ycenter = getmaxy() / 2;         //获取屏幕中心坐标
    for (i=0; i<13; i++) {
        setfillstyle(i, WHITE);     //设置填充风格和颜色
        fillellipse(xcenter, ycenter, 100, 50); //画椭圆并填充
        getch();
    }
    closegraph();                   //关闭图形系统
}
```

运行上述程序，首先初始化图形系统；然后获取屏幕中心坐标，循环设置填充风格和颜色，绘制并填充椭圆，最后关闭图形系统。

6.10 获取填充模式和填充颜色函数：

`getfillsettings()`

函数 `getfillsettings()` 用于获取填充模式和填充颜色，其原型如下。

```
void far getfillsettings(struct fillsettingstype far *fillinfo);
```

【参数】 参数fillinfo为暂存填充模式和颜色的结构体。

【返回值】 该函数没有返回值。

【例 6-10】 下面的示例演示了getfillsettings()函数的使用, 在程序中采用该函数获取当前填充模式和填充颜色, 代码如下。

```
#include <graphics.h> //加入图形处理库
#include <stdlib.h> //加入标准工具库
#include <stdio.h> //加入标准输入输出库
#include <conio.h> //加入控制台输入输出库

char *fname[ ] = { "EMPTY_FILL", "SOLID_FILL", "LINE_FILL",
"LTSLASH_FILL", "SLASH_FILL", "BKSLASH_FILL", "LTBKSLASH_FILL",
"HATCH_FILL", "XHATCH_FILL", "INTERLEAVE_FILL", "WIDE_DOT_FILL",
"CLOSE_DOT_FILL", "USER_FILL" }; //结构体

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy; int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult( );
    midx = getmaxx( ) / 2;
    midy = getmaxy( ) / 2; //获取屏幕中心坐标
    getfillsettings(&fillinfo); //取填充模式和颜色
    sprintf(patstr, "%s is the fill style.", fname[fillinfo.pattern]); //将填充信息转换为字符串
    sprintf(colstr, "%d is the fill color.", fillinfo.color);
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置对齐方式
    outtextxy(midx, midy, patstr); //输出
    outtextxy(midx, midy+2*textheight("W"), colstr); //输出
    getch( );
    closegraph( ); //关闭图形系统
}
```

运行上述程序, 首先初始化图形系统; 然后获取屏幕中心坐标, 获取当前填充模式和颜色, 并将该信息转换为字符串输出到控制台; 最后关闭图形系统。

6.11 将用户定义的填充模式复制到内存

函数: getfillpattern()

函数 `getfillpattern()` 将用户定义的填充模式复制到内存, 其原型如下。

```
void far getfillpattern(char far *upattern);
```

【参数】 参数 `upattern` 为暂存填充模式的数组。

【返回值】 该函数执行后把定义当前用户填充图样的 8 个字节填入 `pattern` 所指向的数组。

【例 6-11】 下面的示例演示了 `getfillpattern()` 函数的使用, 使用该函数将用户定义的填充模式复制到内存, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库

void main() {
    int gdriver = DETECT, gmode, errorcode;
    int maxx, maxy;  int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult();
    maxx = getmaxx() / 2;
    maxy = getmaxy() / 2;           //获取屏幕中心坐标
    setcolor(getmaxcolor());        //设置颜色
    setfillpattern(pattern, getmaxcolor()); //设置填充模式
    bar(0, 0, maxx, maxy);         //以设置模式填充屏幕
    getch();
    getfillpattern(pattern);        //取当前用户设置的填充模式
    pattern[4] -= 1;                //改变模式
    pattern[5] -= 3;
    pattern[6] += 3;
    pattern[7] -= 4;
    setfillpattern(pattern, getmaxcolor()); //设置填充模式
```

```

    bar(0, 0, maxx, maxy);           //以设置模式填充屏幕
    getch( );
    closegraph( );                   //关闭图形系统
}

```

运行上述程序, 首先初始化图形系统; 然后获取屏幕中心坐标, 设置填充模式, 以该模式填充屏幕, 待按下任意键后, 获取当前用户设置的填充模式并强行修改, 再次设置修改后的模式; 最后关闭图形系统。

() 6.12 选择用户定义的填充模式函数: 8.1.8

setfillpattern()

函数 setfillpattern() 用于选择用户定义的填充模式, 其原型如下。

```
void far setfillpattern(char far *upattern, int color);
```

【参数】 参数 upattern 为要设置的模式; 参数 color 为要设置的颜色。

【返回值】 该函数没有返回值。

【例 6-12】 下面的示例演示了 setfillpattern() 函数的使用, 采用该函数设置用户定义的填充模式, 代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int maxx, maxy;  int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    maxx = getmaxx( ) / 2;
    maxy = getmaxy( ) / 2;                      //获取屏幕中心坐标
    setcolor(getmaxcolor( ));                   //设置颜色
    setfillpattern(pattern, getmaxcolor( ));    //设置用户模式
}

```

```

    bar(0, 0, maxx, maxy);           //填充屏幕
    getch( );
    closegraph( );                   //关闭图形系统
}

```

运行上述程序，首先初始化图形系统；然后获取屏幕中心坐标，设置颜色和用户模式，并以该模式填充整个屏幕；最后待按下任意键后关闭图形系统。

6.13 设置填充模式和颜色函数：setfillstyle()

函数 setfillstyle()用于设置填充模式和颜色，其原型如下。

```
void far setfillstyle(int pattern, int color);
```

【参数】 参数pattern为要设置的模式；参数color为设置的颜色。

【返回值】 该函数没有返回值。

【例 6-13】 下面的示例演示了setfillstyle()函数的使用，采用该函数设置填充模式和颜色，代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <string.h>              //加入字符串库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库

char *fname[ ] = { "EMPTY_FILL", "SOLID_FILL",
    "LINE_FILL", "LTSLASH_FILL",
    "SLASH_FILL", "BKSLASH_FILL",
    "LTBKSLASH_FILL", "HATCH_FILL",
    "XHATCH_FILL", "INTERLEAVE_FILL",
    "WIDE_DOT_FILL", "CLOSE_DOT_FILL", "USER_FILL" };

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int style, midx, midy;  int radius = 100;
    initgraph(&gdriver, &gmode, "");    //初始化图形系统
}

```



```

    errorcode = graphresult( );
    xcenter = getmaxx( ) / 2;
    ycenter = getmaxy( ) / 2;           //获取屏幕中心坐标
    for (style = EMPTY_FILL; style < USER_FILL; style++) {
        setfillstyle(style, getmaxcolor( )); //设置模式和颜色
        strcpy(stylestr, fname[style]);     //转换风格为字符串
        bar3d(0, 0, midx-10, midy, 0, 0);  //绘制三维条码
        outtextxy(midx, midy, stylestr);    //输出设置的风格
        getch( );
        cleardevice( );                    //清除图形屏幕
    }
    getch( );
    closegraph( );                        //关闭图形系统
}

```

运行上述代码, 首先初始化图形系统; 然后获取屏幕中心坐标, 循环设置填充模式和颜色, 以该模式绘制三维条码, 将所设置的模式转换为字符串并输出, 待按下任意键后清除图形屏幕; 最后按下任意键关闭图形系统。

6.14 输出当前的文本属性函数: settextstyle()

函数 settextstyle()用于输出当前的文本属性, 其原型如下。

```
void far settextstyle (int font, int direction, char size);
```

【参数】 参数font为输出的字形; 参数direction为文本方向; 参数size为字符大小。

【返回值】 该函数没有返回值。

【例 6-14】 下面的示例演示了settextstyle()函数的使用, 采用该函数输出当前的文本属性, 代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库

```

```

#include <conio.h> //加入控制台输入输出库
char *fname[ ] = { "DEFAULT font", "TRIPLEX font",
                  "SMALL font", "SANS SERIF font",
                  "GOTHIC font" };
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int style, midx, midy; int radius = 100;
    int size=1;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult( );
    midx = getmaxx( ) / 2;
    midy = getmaxy( ) / 2; //获取屏幕中心坐标
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    for (style=DEFAULT_FONT; style<=GOTHIC_FONT; style++) {
        cleardevice( ); //清除图形屏幕
        if (style == TRIPLEX_FONT) size = 4;
        setttextstyle(style, HORIZ_DIR, size); //设置文本属性
        outtextxy(midx, midy, fname[style]); //输出
        getch( );
    }
    closegraph( ); //关闭图形系统
}

```

运行上述程序，首先初始化图形系统；然后获取屏幕中心坐标，设置文本对齐方式，循环设置文本属性，并在指定位置输出文本属性；最后关闭图形系统。

6.15 获取当前图形文本字体的信息

函数：getttextsettings()

函数 `getttextsettings()` 用于获取当前图形文本字符的信息，其原型如下。

```
void far getttextsettings(struct textsettingstype far *textinfo);
```


【参数】 参数textinfo为暂存文件字符信息的结构体。

【返回值】 该函数没有返回值。

【例 6-15】 下面的示例演示了gettextsettings()函数的使用, 采用该函数获取当前图形文本字符的信息, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

char *font[ ] = { "DEFAULT_FONT", "TRIPLEX_FONT",
                  "SMALL_FONT", "SANS_SERIF_FONT",
                  "GOTHIC_FONT" };

char *dir[ ] = { "HORIZ_DIR", "VERT_DIR" }; //文本方向
char *hjust[ ] = { "LEFT_TEXT", "CENTER_TEXT", "RIGHT_TEXT" };
                                     //水平对齐方式
char *vjust[ ] = { "BOTTOM_TEXT", "CENTER_TEXT", "TOP_TEXT" };
                                     //垂直对齐方式

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, ht;  int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    midx = getmaxx( ) / 2;
    midy = getmaxy( ) / 2;                      //获取屏幕中心坐标
    gettextsettings(&textinfo);                 //获取当前文本字体信息
    sprintf(fontstr, "%s is the text style.", font[textinfo.font]);
                                               //将字体信息转换为字符串
    sprintf(dirstr, "%s is the text direction.",
dir[textinfo.direction]);
    sprintf(sizestr, "%d is the text size.", textinfo.charsize);
    sprintf(hjuststr, "%s is the horizontal justification.",
hjust[textinfo.horiz]);
    sprintf(vjuststr, "%s is the vertical justification.",
vjust[textinfo.vert]);
    ht = textheight("W");                      //获取字符串高度
    settextjustify(CENTER_TEXT, CENTER_TEXT); //设置对齐方式
    outtextxy(midx, midy, fontstr);             //输出字符信息
```



```

    outtextxy(midx, midy+2*ht, dirstr);
    outtextxy(midx, midy+4*ht, sizestr);
    outtextxy(midx, midy+6*ht, hjuststr);
    outtextxy(midx, midy+8*ht, vjuststr);
    getch( );
    closegraph( );           //关闭图形系统
}

```

运行上述代码，首先初始化图形系统；然后获取屏幕中心坐标，获取当前文本字体信息，并将该信息转换为字符串；最后设置文本对齐方式，输出文本信息，待按下任意键后关闭图形系统。

6.16 在视区显示字符串函数：outtext()

函数 outtext() 用于在视区显示字符串，其原型如下。

```
void far outtext(char far *textstring);
```

【参数】 参数 textstring 为要显示的字符串。

【返回值】 该函数没有返回值。

【例 6-16】 下面的示例演示了 outtext() 函数的使用，采用该函数在视区显示字符串，代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;  int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    midx= getmaxx( ) / 2;
    midy= getmaxy( ) / 2;                       //获取屏幕中心坐标
    moveto(midx, midy);                         //移动光标到屏幕中心
}

```

```

    outtext("This ");           //输出字符串
    outtext("is ");
    outtext("a ");
    outtext("test.");
    getch( );
    closegraph( );              //关闭图形系统
}

```

运行上述代码, 首先初始化图形系统, 然后获取屏幕中心坐标, 将光标移动到屏幕中心, 输出字符串, 待按下任意键后关闭图形系统。

6.17 在指定位置显示字符串函数: outtextxy()

函数 outtextxy() 用于在指定位置显示字符串, 其原型如下。

```
void far outtextxy(int x, int y, char *textstring);
```

【参数】 参数x和参数y为要显示的位置坐标; 参数textstring为要显示的字符串。

【返回值】 该函数没有返回值。

【例 6-17】 下面的示例演示了 outtextxy() 函数的使用, 在屏幕指定位置显示字符串, 代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;  int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    midx = getmaxx( ) / 2;
    midy = getmaxy( ) / 2;                      //获取屏幕中心坐标
    outtextxy(midx, midy, "This is a test."); //输出字符串
}

```



```

    getch( );
    closegraph( );           //关闭图形系统
}

```

运行上述代码，首先初始化图形系统，然后获取屏幕中心坐标，在屏幕中心输出所给字符串，待按下任意键后关闭图形系统。

6.18 注册链入的字体代码函数：

registerbgifont()

函数 registerbgifont() 用于注册链入的字体代码，其原型如下。

```
int registerbgifont(void(*font)(void))
```

【参数】参数 font 为要链入的字符。

【返回值】若该函数执行成功，则返回 1；否则，返回 0。

【例 6-18】下面的示例演示了 registerbgifont() 函数的使用，使用该函数注册链入的字体代码，代码如下。

```

#include<dos.h>
#include<conio.h>
#include<graphics.h>
#include<stdio.h>
void initgr()           //声明初始化图形系统字体的函数
{
    int gd=DETECT,gm=0;
    registerbgidriver(EGAVGA_driver); //注册驱动
    registerbgifont(triplex_font);    //注册字体
    registerbgifont(small_font);      //注册字体
    registerbgifont(sansserif_font);
    registerbgifont(gothic_font);
    initgraph(&gd,&gm,"");           //初始化图形系统
}
...
void start()

```



```

{
    int i=0;

    initgr(); //调用初始化

    setbkcolor(LIGHTGRAY); //设置背景
}
main()
{
    int i=0;
    int key=0;
    start();
    while(key!=ESC)
    {
        for(i=0;i<7;i++)
        {
            up(100+i*(KEY_WIDTH+5),110);
        }

        key=bioskey(0);
        switch(key)
        {
            case s:down(100,110);sound(523);delay(3000);nosound();break;
            case d:down(165,110);sound(587);delay(3000);nosound();break;
            case f:down(230,110);sound(659);delay(3000);nosound();break;
            case g:down(295,110);sound(699);delay(3000);nosound();break;
            case h:down(360,110);sound(784);delay(3000);nosound();break;
            case j:down(425,110);sound(880);delay(3000);nosound();break;
            case k:down(490,110);sound(988);delay(3000);nosound();break;
            default:break;
        }
    }
    closegr(); //关闭图形系统
}

```

运行上述程序,首先声明用于初始化图形系统字体的函数,然后在主程序中调用该函数进行图形系统字体的初始化。

6.19 设置文本的对齐方式函数:

settextjustify()

函数 settextjustify()用于设置文本的对齐方式, 其原型如下。

```
void far settextjustify(int horiz, int vert);
```

【参数】 参数horiz为水平对齐方式; 参数vert为垂直对齐方式。

【返回值】 该函数没有返回值。

【例 6-19】 下面的示例演示了settextjustify()函数的使用, 在程序中采用该函数设置文本的对象方式, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
void xat(int x, int y);         //声明函数原型
char *hjust[ ] = { "LEFT_TEXT", "CENTER_TEXT", "RIGHT_TEXT" };
                                //水平对齐方式
char *vjust[ ] = { "LEFT_TEXT", "CENTER_TEXT", "RIGHT_TEXT" };
                                //垂直对齐方式

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, hj, vj;    int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult( );
    xcenter = getmaxx( ) / 2;
    ycenter = getmaxy( ) / 2;    //获取最大坐标
    for (hj=LEFT_TEXT; hj<=RIGHT_TEXT; hj++)
        for (vj=LEFT_TEXT; vj<=RIGHT_TEXT; vj++) {
            cleardevice( );      //清除图形
            settextjustify(hj, vj); //设置文本对齐方式
            sprintf(msg, "%s %s", hjust[hj], vjust[vj]); //转换为字符串
            xat(midx, midy);
            outtextxy(midx, midy, msg); //输出对齐方式
            getch( );
        }
```



```

    }
    closegraph( );           //关闭文件系统
}
void xat(int x, int y) {     //画一个x
    line(x-4, y, x+4, y);
    line(x, y-4, x, y+4);
}

```

运行上述代码, 首先初始化图形系统; 然后获取最大坐标, 循环清除屏幕, 设置文本对齐方式, 将文本对齐方式转换为字符串, 并在屏幕上画一个“X”, 在指定位置输出字符串; 最后待按下任意键后关闭图形系统。

6.20 为矢量字体改变字符宽度和高度函数:

setusercharsize()

函数 setusercharsize()用于为矢量字符设置字符宽度和高度, 其原型如下。

```
void far setusercharsize(int multx, int divx, int multy, int divy);
```

【参数】参数multx和divx为设置的字符宽度比例; 参数multy和divy为设置的字符高度比例。

【返回值】该函数调用后, 屏幕上显示的字符宽为其默认值 8 个像素乘以宽度比例, 高为其默认值 8 个像素乘以高度比例。

【例 6-20】下面的示例演示了setusercharsize()函数的使用, 使用该函数改变矢量字体的字符宽度和高度, 代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
void main( ){

```



```

int gdriver = DETECT, gmode, errorcode;
int midx, midy; int radius = 100;
initgraph(&gdriver, &gmode, ""); //初始化图形系统
errorcode = graphresult();
settextstyle(TRIPLEX_FONT, HORIZ_DIR, 4); //设置字体风格
moveto(0, getmaxy() / 2); //移动到字符开始位置
outtext("Norm "); //输出字符
setusercharsize(1, 3, 1, 1); //设置字符为一般字体的1/3宽
outtext("Short "); //输出字符
setusercharsize(3, 1, 1, 1); //设置字符为一般字符的3倍
outtext("Wide"); //输出字符
getch();
closegraph(); //关闭图形系统
}

```

运行上述代码，首先初始化图形系统；然后设置字体风格，将光标移动到字符开始位置，输出字符，设置字符宽度为一般字符的 1/3，输出字符，再将宽度设置为一般字符的 3 倍，输出字符；最后待按下任意键后关闭图形系统。

注意：只有在调用 `settextstyle()` 函数设置参数 `charsize` 为 0 时，字符调节参数才有效。

6.21 获取以像素为单位的字符串高度函数：

`textheight()`

函数 `textheight()` 用于获取以像素为单位的字符串高度，其原型如下。

```
int far textheight(char far *textstring);
```

【参数】 参数 `textstring` 为处理的字符串。

【返回值】 该函数按当前字体、字符大小、比例因子和文本方向，计

算出字符串高度的像素数。这可以是当前单字符的高度,也可以是当前整个字符串的高度值。

【例 6-21】下面的示例演示了textheight()函数的使用,获取以像素为单位的字符串高度,代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;  int radius = 100;
    initgraph(&gdriver, &gmode, "");      //初始化图形系统
    errorcode = graphresult( );
    for (i=1; i<11; i++) {
        settextstyle(TRIPLEX_FONT, HORIZ_DIR, i); //设置文本风格
        sprintf(msg, "Size: %d", i);           //字符串
        outtextxy(1, y, msg);                  //指定位置输出
        y += textheight(msg);                  //获取字符串高度
    }
    getch( );
    closegraph( );                            //关闭图形系统
}
```

运行上述代码,首先初始化图形系统;然后循环设置文本风格,在指定位置输出,获取字符串高度;最后待按下任意键后关闭图形系统。

6.22 获取以像素为单位的字符串宽度

函数: textwidth()

函数 textwidth()用于获取以像素为单位的字符串宽度,其原型

如下。

```
int far textwidth(char far *textstring);
```

【参数】参数textstring为要处理的字符串。

【返回值】该函数按当前字体、字符大小、比例因子和文本方向，计算出字符串宽度的像素数。这既可以是单字符的宽度，又可以是整个字符串的宽度，并且通常是整个字符串的宽度值。

【例 6-22】下面的示例演示了textwidth()函数的使用，获取以像素为单位的字符串宽度，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;  int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    xcenter = getmaxx() / 2;
    ycenter = getmaxy( ) / 2;                  //获取屏幕中心坐标
    setttextjustify(LEFT_TEXT, CENTER_TEXT);   //设置文本对齐方式
    for (i=1; i<11; i++) {
        setttextstyle(TRIPLEX_FONT, HORIZ_DIR, i); //设置文本风格
        sprintf(msg, "Size: %d", i);
        outtextxy(x, y, msg);                  //输出文本
        x += textwidth(msg);                   //获取宽度
    }
    getch( );
    closegraph( );                             //关闭图形系统
}
```

运行上述代码，首先初始化图形系统；然后循环设置文本风格，在指定位置输出，获取字符串宽度；最后待按下任意键后关闭图形系统。

6.23 返回最后一次不成功的图形操作的错误

代码函数: graphresult()

函数 graphresult()用于返回最后一次不成功的图形操作错误, 其原型如下。

```
int far graphresult(void);
```

【参数】 该函数没有参数。

【返回值】 该函数返回最后一次失败图形操作的错误代码。

【例 6-23】 下面的示例演示了 graphresult()函数的使用, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;  int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    line(0, 0, getmaxx( ), getmaxy( ));        //画一直线
    getch( );
    closegraph( );                             //关闭图形系统
}
```

运行上述代码, 首先初始化图形系统; 若没有错误, 画一直线; 最后待按下任意键后关闭图形系统。

6.24 初始化图形系统函数: initgraph()

函数 initgraph()用于初始化图形系统, 其原型如下。

```
void far initgraph(int far *graphdriver, int far *graphmode, char
far *pathtodriver);
```

【参数】参数graphdriver和graphmode分别表示图形驱动器和模式；参数pathtodriver是指图形驱动程序所在的目录路径。

【返回值】该函数没有返回值。

该函数的使用方法已在前面的例子中进行了介绍，此处不再赘述。

6.25 获取当前视口的信息函数：

getviewsettings()

函数 getviewsettings() 用于获取当前视口的信息，其原型如下。

```
void far getviewsettings(struct viewporttype far *viewport);
```

【参数】参数viewport为暂存视口信息的结构体。

【返回值】该函数执行后，把有关当前视口的信息装入viewport所指向的viewporttype型结构中。

【例 6-24】下面的示例演示了getviewsettings()函数的使用，采用该函数获取当前视区的信息，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
char *clip[ ] = { "OFF", "ON" };
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy; int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult();
    xcenter = getmaxx( ) / 2;
    ycenter = getmaxy( ) / 2;       //获取屏幕中心坐标
```

```

    getviewsettings(&viewinfo);           //取当前视区信息
    sprintf(topstr, "(%d, %d) is the upper left viewport corner.",
viewinfo.left, viewinfo.top);           //转换为字符串
    sprintf(botstr, "(%d, %d) is the lower right viewport corner.",
viewinfo.right, viewinfo.bottom);
    sprintf(clipstr, "Clipping is turned %s.", clip[viewinfo.clip]);
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    ht = textheight("W");
    outtextxy(midx, midy, topstr);           //输出视区信息
    outtextxy(midx, midy+2*ht, botstr);
    outtextxy(midx, midy+4*ht, clipstr);
    closegraph();           //关闭图形系统
}

```

运行上述代码, 首先初始化图形系统; 然后获取屏幕中心坐标, 获取视区信息, 并转换为字符串, 设置文本对齐方式后输出视区信息; 最后待按下任意键后关闭图形系统。

6.26 清除图形视区函数: clearviewport()

函数 clearviewport() 用于清除图形视区, 其原型如下。

```
void far clearviewport(void);
```

【参数】该函数没有参数。

【返回值】该函数没有返回值。

【例 6-25】下面的示例演示了 clearviewport() 函数的使用, 采用该函数清除图形视区, 代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>           //加入标准工具库
#include <stdio.h>           //加入标准输入输出库
#include <conio.h>           //加入控制台输入输出库
#define CLIP_ON 1
void main() {

```



```

int gdriver = DETECT, gmode, errorcode;
int midx, midy, ht; int radius = 100;
initgraph(&gdriver, &gmode, ""); //初始化图形系统
errorcode = graphresult();
setcolor(getmaxcolor()); //设置颜色
ht = textheight("W");
outtextxy(0, 0, "* <-- (0, 0) in default viewport");//输出文本
setviewport(50, 50, getmaxx()-50, getmaxy()-50, CLIP_ON);
//设置当前视口
outtextxy(0, 0, "* <-- (0, 0) in smaller viewport");//输出
outtextxy(0, 2*ht, "Press any key to clear viewport:");
getch();
clearviewport(); //清除视口
outtextxy(0, 0, "Press any key to quit:"); //输出
closegraph(); //关闭图形系统
}

```

运行上述代码,首先初始化图形系统;然后设置颜色,输出文本,设置当前视口,再输出文本,清除当前视口,再输出其他文本;最后待按下任意键后关闭图形系统。

6.27 获取当前图形位置的 x 坐标函数: getx()

函数 getx() 用于获取当前图形位置的 x 坐标,其原型如下。

```
int far getx(void);
```

【参数】该函数没有参数。

【返回值】该函数返回当前图形位置的 x 坐标。

【例 6-26】下面的示例演示了 getx() 函数的使用,采用该函数获取当前图形位置的 x 坐标,代码如下。

```

#include <graphics.h> //加入图形处理库
#include <stdlib.h> //加入标准工具库
#include <stdio.h> //加入标准输入输出库

```

```

#include <conio.h> //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy; int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult( );
    moveto(getmaxx( ) / 2, getmaxy( ) / 2); //移动光标到屏幕中心位置
    sprintf(msg, "<-(%d, %d) is the here.", getx( ), gety( ));
    //将坐标转换为字符串
    outtext(msg); //输出
    closegraph( ); //关闭图形系统
}

```

运行上述代码, 首先初始化图形系统; 然后将光标移动到当前屏幕中心位置, 获取当前图形位置的 x 坐标并转换为字符串, 再输出至屏幕; 最后待按下任意键后关闭图形系统。

6.28 获取当前图形位置的 y 坐标函数: gety()

函数 gety() 用于获取当前图形位置的 y 坐标, 其原型如下。

```
int far gety(void);
```

【参数】 该函数没有参数。

【返回值】 该函数返回当前图形位置的 y 坐标。

【例 6-27】 下面的示例演示了 gety() 函数的使用, 采用该函数获取当前图形位置的 y 坐标, 代码如下。

```

#include <graphics.h> //加入图形处理库
#include <stdlib.h> //加入标准工具库
#include <stdio.h> //加入标准输入输出库
#include <conio.h> //加入控制台输入输出库
#define CLIP_ON 1
void main( ){

```

```

int gdriver = DETECT, gmode, errorcode;
int midx, midy; int radius = 100;
initgraph(&gdriver, &gmode, "");           //初始化图形系统
errorcode = graphresult();
moveto(getmaxx() / 2, getmaxy() / 2); //移动光标到屏幕中心位置
sprintf(msg, "<-(%d, %d) is the here.", getx(), gety());
                                           //将坐标转换为字符串
outtext(msg);                             //输出
closegraph();                             //关闭图形系统
}

```

运行上述代码，首先初始化图形系统；然后将光标移动到当前屏幕中心位置，获取当前图形位置的 y 坐标并转换为字符串，再输出至屏幕；最后待按下任意键后关闭图形系统。

6.29 为图形输出设置当前视口函数：

setviewport()

函数 setviewport() 用于设置当前视口，其原型如下。

```
void far setviewport(int left, int top, int right, int bottom, int clipflag);
```

【参数】 参数 left 和 top 是左上角坐标；参数 right 和 bottom 是右下角坐标，它们都是绝对屏幕坐标。如果参数 clipflag 值为 1，则超出视口的输出图形自动被裁剪掉，即所有作图限制于当前图形视口之内；如果 clipflag 值为 0，则不做裁剪，即作图将无限制地扩展于视口周界之外，直到屏幕边界。若无效参数传入 setviewport()，则 graphresult() 函数返回值 -11，先前视口设置仍有效。

【返回值】 该函数没有返回值。

【例 6-28】 下面的示例演示了 setviewport() 函数的使用，使用该函数为图形输出设置当前视口，代码如下。


```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;  int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    setcolor(getmaxcolor( ));                  //设置颜色
    outtextxy(0, 0, "* <-- (0, 0) in default viewport");
                                                //输出默认视口
    setviewport(50, 50, getmaxx( )-50, getmaxy( )-50, CLIP_ON);
                                                //设置视口
    outtextxy(0, 0, "* <-- (0, 0) in smaller viewport");
                                                //在新视口显示同样内容

    getch( );
    closegraph( );                           //关闭图形系统
}

```

运行上述代码, 首先初始化图形系统; 然后设置颜色, 在默认视口中输出文本, 将当前视口缩小, 再输出同样文本; 最后待按下任意键后关闭图形系统。

6.30 清除图形屏幕函数: cleardevice()

函数 cleardevice()用于清除图形屏幕, 其原型如下。

```
void far cleardevice(void);
```

【参数】该函数没有参数。

【返回值】该函数没有返回值。

【例 6-29】下面的示例演示了cleardevice()函数的使用, 采用该函数清除图形屏幕, 代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;  int radius = 100;
    initgraph(&gdriver, &gmode, "");    //初始化图形系统
    errorcode = graphresult( );
    midx = getmaxx( ) / 2;                //取屏幕坐标
    midy = getmaxy( ) / 2;                //取屏幕坐标
    setcolor(getmaxcolor( ));             //设置颜色
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    outtextxy(midx, midy, "press any key to clear the screen:");
                                           //输出文本至屏幕中央
    getch( );
    cleardevice( );                       //清除图形系统
    outtextxy(midx, midy, "press any key to quit:"); //输出其他消息
    getch( );
    closegraph( );                        //关闭图形系统
}

```

运行上述代码，首先初始化图形系统；然后取屏幕坐标，设置颜色，设置文本对齐方式，待按下任意键后，清除图形，再输出其他文本；最后待按下任意键后关闭图形系统。

6.31 返回屏幕的最大 x 坐标函数：getmaxx()

函数 getmaxx()用于获取屏幕的最大 x 坐标，其原型如下。

```
int far getmaxx(void);
```

【参数】该函数没有参数。

【返回值】该函数返回屏幕的最大x坐标。

【例 6-30】下面的示例演示了getmaxx()函数的使用, 采用该函数返回屏幕的最大x坐标, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;  int radius = 100;
    initgraph(&gdriver, &gmode, "");    //初始化图形系统
    errorcode = graphresult( );
    midx = getmaxx( ) / 2;                //取屏幕中心坐标
    midy = getmaxy( ) / 2;                //取屏幕中心坐标
    sprintf(xrange, "X values range from 0..%d", getmaxx( ));
                                         //转换为字符串
    sprintf(yrange, "Y values range from 0..%d", getmaxy( ));
    settextjustify(CENTER_TEXT, CENTER_TEXT);    //设置对齐方式
    outtextxy(midx, midy, xrange);              //在指定位置输出
    outtextxy(midx, midy+textheight("W"), yrange); //在指定位置输出
    closegraph( );                             //关闭图形系统
}
```

运行上述代码, 首先初始化图形系统; 然后获取当前屏幕的中心坐标, 并将坐标转换为字符串, 并在指定的位置输出; 最后待按下任意键后关闭图形系统。

6.32 返回屏幕的最大 y 坐标函数: getmaxy()

函数 getmaxy()用于获取屏幕的最大 y 坐标, 其原型如下。

```
int far getmaxy(void);
```

【参数】该函数没有参数。

【返回值】该函数返回当前屏幕的最大y坐标。

【例 6-31】下面的示例演示了getmaxy()函数的使用,采用该函数返回屏幕的最大y坐标,代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
#define CLIP_ON 1
void main() {
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy; int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult();
    midx = getmaxx() / 2;           //取屏幕中心坐标
    midy = getmaxy() / 2;           //取屏幕中心坐标
    sprintf(xrange, "X values range from 0..%d", getmaxx());
                                    //转换为字符串
    sprintf(yrange, "Y values range from 0..%d", getmaxy());
    setttextjust(CENTER_TEXT, CENTER_TEXT); //设置对齐方式
    outtextxy(midx, midy, xrange); //在指定位置输出
    outtextxy(midx, midy+textheight("W"), yrange); //在指定位置输出
    closegraph();                  //关闭图形系统
}
```

运行上述代码,首先初始化图形系统;然后获取当前屏幕的中心坐标,并将坐标转换为字符串,并在指定的位置输出;最后待按下任意键后关闭图形系统。

6.33 将当前点移到(x, y)函数: moveto()

函数 moveto()用于将当前点移动指定位置,其原型如下。

```
void far moveto(int x, int y);
```

【参数】参数x和y表示要移动到的坐标位置。

【返回值】该函数没有返回值。

【例 6-32】下面的示例演示了 `moveto()` 函数的使用, 采用该函数将当前点移动到指定位置, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;  int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult( );
    moveto(20, 30);               //移动到坐标为(20,30)的位置
    putpixel(getx( ), gety( ), getmaxcolor( )); //在当前位置画点
    sprintf(msg, " (%d, %d)", getx( ), gety( )); //将坐标转换为字符串
    outtextxy(20, 30, msg);       //在该坐标输出字符串
    moveto(100, 100);            //移动到(100,100)位置
    putpixel(getx( ), gety( ), getmaxcolor( )); //在该位置画点
    sprintf(msg, " (%d, %d)", getx( ), gety( )); //转换为字符串
    outtext(msg);                //在该位置输出字符串
    closegraph( );               //关闭图形系统
}
```

运行上述代码, 首先初始化图形系统; 然后移动当前点到指定位置, 并在该位置画点和输出该位置的坐标信息, 再移动到其他位置, 画点和输出该点的坐标信息; 最后待按下任意键后关闭图形系统。

6.34 将当前点移动一相对距离函数: `moverel()`

函数 `moverel()` 用于将当前点移动一相对距离, 其原型如下。

```
void far moverel(int dx, int dy);
```

【参数】参数 `dx` 为 `x` 方向相对移动距离; 参数 `dy` 为 `y` 方向相对移动距离。

【返回值】该函数没有返回值。

【例 6-33】下面的示例演示了 `moverel()` 函数的使用，采用该函数将当前点移动到一相对位置，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;  int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult( );
    moveto(20, 30);               //移动到屏幕(20,30)的位置
    putpixel(getx( ), gety( ), getmaxcolor( )); //画点
    sprintf(msg, " (%d, %d)", getx( ), gety( )); //转换为字符串
    outtextxy(20, 30, msg);       //在该点输出坐标信息
    moverel(100, 100);            //移动相对位置为100的点
    putpixel(getx( ), gety( ), getmaxcolor( )); //在该位置画点
    sprintf(msg, " (%d, %d)", getx( ), gety( )); //转换为字符串
    outtext(msg);                 //输出
    closegraph( );                //关闭图形系统
}
```

运行上述代码，首先初始化图形系统；然后将当前点移动到指定位置，并在该位置画点和输出坐标信息；接着再移动相对位置，在该位置画点和输出坐标信息；最后待按下任意键后关闭图形系统。

6.35 绘制二维条形图函数：bar()

函数 `bar()` 用于绘制二维条形图，其原型如下。

```
void far bar(int left, int top, int right, int bottom);
```


【参数】参数 left、top 和参数 right、bottom 分别为左上角坐标与右下角坐标。

【返回值】该函数没有返回值。

【例 6-34】下面的示例演示了 bar() 函数的使用, 采用该函数绘制二维条形图, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;  int radius = 100;
    initgraph(&gdriver, &gmode, "");    //初始化图形系统
    errorcode = graphresult( );
    midx = getmaxx( ) / 2;              //取屏幕中心坐标
    midy = getmaxy( ) / 2;              //取屏幕中心坐标
    for (i=SOLID_FILL; i<USER_FILL; i++) {
        setfillstyle(i, getmaxcolor( )); //设置填充风格
        bar(midx-50, midy-50, midx+50, midy+50); //绘制二维条形图
        getch( );
    }
    closegraph( );                    //关闭图形系统
}
```

运行上述代码, 首先初始化图形系统。然后获取当前屏幕的中心坐标, 循环设置填充风格, 绘制二维条形图; 最后待按下任意键后关闭图形系统。

6.36 绘制三维条形图函数: bar3d()

函数 bar3d() 用于绘制三维条形图, 其原型如下。

```
void far bar3d(int left, int top, int right, int bottom, int depth,
int topflag);
```

【参数】参数 left、top 和 right、bottom 分别为左上角坐标与右下角坐标；参数 depth 为条块的深度，以像素为单位，通常按宽度的四分之一计算，深度方向通过屏幕纵横比调节为约 45 度。参数 topflag 相当于设置一个布尔参数，如果设置为 1，那么条块上放一顶面；若设置为 0，则三维条形就没有顶面，这样可使多个三维条形叠加在一起。

【返回值】该函数没有返回值。

【例 6-35】下面的示例演示了 bar3d() 函数的使用，采用该函数绘制三维条形图，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy, i;  int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult();
    midx = getmaxx( ) / 2;                      //取屏幕坐标
    midy = getmaxy( ) / 2;                      //取屏幕坐标
    for (i=EMPTY_FILL; i<USER_FILL; i++) { //循环填充模式
        setfillstyle(i, getmaxcolor( ));      //设置填充风格
        bar3d(midx-50, midy-50, midx+50, midy+50, 10, 1);
                                                //绘制三维条形图

        getch( );
    }
    closegraph( );                          //关闭图形系统
}
```

运行上述代码，首先初始化图形系统；然后获取当前屏幕坐标，按填充模式循环设置填充风格，绘制三维条形图；最后待按下任意键

后关闭图形系统。

6.37 绘制矩形函数: rectangle()

函数 rectangle() 用于绘制矩形, 其原型如下。

```
void far rectangle(int left, int top, int right, int bottom);
```

【参数】 参数 left、top 和参数 right、bottom 分别为矩形的左上角坐标和右下角坐标。

【返回值】 该函数没有返回值。

【例 6-36】 下面的示例演示了 rectangle() 函数的使用, 在程序中采用该函数绘制矩形, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main() {
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy; int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult();
    left = getmaxx() / 2 - 50;        //计算矩形左上角 x 坐标
    top = getmaxy() / 2 - 50;         //计算矩形左上角 y 坐标
    right = getmaxx() / 2 + 50;       //计算右下角 x 坐标
    bottom = getmaxy() / 2 + 50;      //计算右下角 y 坐标
    rectangle(left, top, right, bottom); //绘制矩形
    closegraph();                    //关闭图形系统
}
```

运行上述代码, 首先初始化图形系统; 然后计算矩形的左上角坐标和右下角坐标, 绘制矩形; 最后待按下任意键后关闭图形系统。

6.38 设置当前画线颜色函数: setcolor()

函数 setcolor() 用于设置当前画线颜色, 其原型如下。

```
void far setcolor(int color);
```

【参数】 参数 color 为要设置的颜色。

【返回值】 该函数没有返回值。

【例 6-37】 下面的示例演示了 setcolor() 函数的使用, 使用该函数设置当前画线颜色, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main() {
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy; int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult();
    maxcolor = getmaxcolor();        //取支持的最大颜色
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    x = getmaxx() / 2;               //取屏幕中心坐标
    y = getmaxy() / 2;               //取屏幕中心坐标
    for (color=1; color<=maxcolor; color++) { //循环可用的颜色
        cleardevice();               //清除图形屏幕
        setcolor(color);              //设置颜色
        sprintf(msg, "Color: %d", color); //输出字符串
        outtextxy(x, y, msg);
        getch();
    }
    closegraph();                   //关闭图形系统
}
```

运行上述代码, 首先初始化图形系统; 然后获取当前系统所支持的最大颜色, 设置文本对齐方式, 取屏幕中心坐标, 循环可用颜色,

以不同的颜色输出当前颜色信息;最后待按下任意键后关闭图形系统。

6.39 获取当前画线颜色函数: getcolor()

函数 getcolor()用于获取当前画线颜色,其原型如下。

```
int far getcolor(void);
```

【参数】该函数没有参数值。

【返回值】该函数返回当前画线颜色。

【例 6-38】下面的示例演示了 getcolor()函数的使用,采用该函数获取当前画线颜色,代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    midx = getmaxx( ) / 2;                      //取屏幕中心位置坐标
    midy = getmaxy( ) / 2;
    setcolor(getmaxcolor( ));                  //设置颜色
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    color = getcolor( );                       //取当前画线颜色
    itoa(color, colname, 10);                  //转换为字符串
    strcat(colname,
    " is the current drawing color.");          //字符串拼接
    outtextxy(midx, midy, colname);            //在屏幕中心输出颜色串
    closegraph( );                             //关闭图形系统
}
```

运行上述代码，首先初始化图形系统；然后获取屏幕中心坐标，设置支持的最大颜色和文本对齐方式；接着获取当前画线颜色并将其转换为字符串输出；最后待按下任意键后关闭图形系统。

6.40 获取当前线型、模式和宽度函数：

getlinesettings()

函数 `getlinesettings()` 用于获取当前线型、模式和宽度，其原型如下。

```
void far getlinesettings(struct linesettingstype far *lininfo);
```

【参数】参数 `lininfo` 为暂存线型、模式信息的结构体。

【返回值】该函数没有返回值。

【例 6-39】下面的示例演示了 `getlinesettins()` 函数的使用，采用该函数获取当前线型、模式和宽度，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

char *lname[ ] = { "SOLID_LINE",
                  "DOTTED_LINE",
                  "CENTER_LINE",
                  "DASHED_LINE",
                  "USERBIT_LINE"
                };

#define CLIP_ON 1

void main() {
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
```



```

    errorcode = graphresult( );
    midx = getmaxx( ) / 2; //取屏幕中心坐标
    midy = getmaxy( ) / 2;
    getlinesettings(&lineinfo); //取当前线型号信息
    sprintf(lstyle, "%s is the line style.",
            lname[lineinfo.linestyle]); //将当前线型信息转换为字符串
    sprintf(lpattern, "0x%X is the user-defined line pattern.",
            lineinfo.upattern);
    sprintf(lwidth, "%d is the line thickness.",
            lineinfo.thickness);
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    outtextxy(midx, midy, lstyle); //输出
    outtextxy(midx, midy+2*textheight("W"), lpattern);
    outtextxy(midx, midy+4*textheight("W"), lwidth);
    closegraph( );
}

```

运行上述代码,首先初始化图形系统;然后获取当前屏幕中心坐标,获取当前线型、模式的信息,将其转换为字符串;最后在屏幕中心输出,待按下任意键后关闭图形系统。

6.41 获取当前背景颜色函数: getbkcolor()

函数 getbkcolor()用于获取当前背景颜色,其原型如下。

```
int far getbkcolor(void);
```

【参数】该函数没有参数。

【返回值】该函数返回所取的背景颜色。

【例 6-40】下面的示例演示了 getbkcolor()函数的使用,采用该函数获取当前背景颜色,代码如下。

```

#include <graphics.h> //加入图形处理库
#include <stdlib.h> //加入标准工具库
#include <stdio.h> //加入标准输入输出库

```

```

#include <conio.h> //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int bkcolor, midx, midy;
    int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult();
    midx = getmaxx( ) / 2; //获取屏幕中心坐标
    midy = getmaxy( ) / 2;
    setcolor(getmaxcolor( )); //设置颜色
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置对齐方式
    bkcolor = getbkcolor( ); //获取背景颜色
    itoa(bkcolor, bkname, 10); //转换为字符串
    strcat(bkname, " is the current background color.");
    outtextxy(midx, midy, bkname); //在屏幕中心输出
    closegraph( ); //关闭图形系统
}

```

运行上述代码，首先初始化图形系统。然后获取屏幕中心坐标，设置文本对齐方式；接着获取当前背景颜色并转换为字符串，再在屏幕中心输出；最后待按下任意键后关闭图形系统。

() `getbkcolor` : 获取当前背景颜色函数 14.6

6.42 设置当前背景颜色函数: `setbkcolor()`

函数 `setcolor()` 用于设置当前背景颜色，其原型如下。

```
void far setbkcolor(int color);
```

【参数】 参数 `color` 为要设置的背景颜色。

【返回值】 该函数没有返回值。

【例 6-41】 下面的示例演示了 `setbkcolor()` 函数的使用，采用该函数设置当前背景颜色，代码如下。

```
#include <graphics.h> //加入图形处理库
```



```

#include <stdlib.h>           //加入标准工具库
#include <stdio.h>           //加入标准输入输出库
#include <conio.h>           //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult( );
    maxcolor = getmaxcolor( ); //获取支持的最大颜色
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    x = getmaxx( ) / 2; //取屏幕中心坐标
    y = getmaxy( ) / 2;
    for (bkcol=0; bkcol<=maxcolor; bkcol++) //循环可用颜色
        cleardevice( ); //清除图形
        setbkcolor(bkcol); //设置背景颜色
        if (bkcol == WHITE) setcolor(EGA_BLUE); //设置颜色
        sprintf(msg, "Background color: %d", bkcol); //输出
        outtextxy(x, y, msg);
    }
    closegraph( ); //关闭图形系统
}

```

运行上述代码,首先初始化图形系统;然后获取所支持的最大颜色,设置文本对齐方式;接着循环颜色变量,依次将其设置为背景颜色,再输出颜色信息;最后待按下任意键后关闭图形系统。

6.43 获取可以传给函数 setcolor 的最大颜色

函数: getmaxcolor()

函数 getmaxcolor()用于获取系统支持的最大颜色,其原型如下。

```
int far getmaxcolor(void);
```


【参数】该函数没有参数。

【返回值】该函数返回当前系统所支持的最大颜色值。

【例 6-42】下面的示例演示了 getmaxcolor() 函数的使用, 采用该函数获取最大颜色值, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy; char colstr[80];
    int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult();
    midx = getmaxx() / 2;           //获取屏幕中心坐标
    midy = getmaxy() / 2;
    sprintf(colstr, "This mode supports colors 0..%d",
getmaxcolor()); //获取颜色并转换为字符串
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    outtextxy(midx, midy, colstr); //输出
    getch();
    closegraph(); //关闭图形系统
}
```

运行上述代码, 首先初始化图形系统。然后获取屏幕中心坐标, 获取当前图形系统所支持的最大颜色并转换为字符串输出; 最后待按下任意键后关闭图形系统。

6.44 获取有关当前调色板的信息函数: getpalette()

函数 getpalette() 用于获取有关当前调色板信息, 其原型如下。

```
void far getpalette(struct palettetype far *palette);
```

【参数】 参数palette为palettetype型的结构。该函数调用执行后, 当前调色板的值被装入palette所指向的结构中, 从而可以从该结构中获得这个调色板的设置信息。palette结构如下。

```
#define MAXCOLORS 15
struct palettetype {
    unsigned char size;
    signed char colors[MAXCOLORS+1];
};
```

其中, size 用于存放当前调色板的有效颜色个数; colors 数组的每一元素中装入调色板的一个相应颜色值。

【返回值】 该函数没有返回值。

【例 6-43】 下面的示例演示了getpalette()函数的使用, 在程序中获取当前调色板信息, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    getpalette(&pal);                           //获取调色板信息
    sprintf(psize, "The palette has %d \
        modifiable entries.", pal.size); //转换为字符串
    outtextxy(0, y, psize);                     //输出
    if (pal.size != 0) {
        ht = textheight("W");
        y += 2*ht;
        outtextxy(0, y, "Here are the current \ values:"); //输出
        y += 2*ht;
        for (i=0; i<pal.size; i++, y+=ht) {
```



```

    sprintf(pval, "palette[%02d]: 0x%02X", i, pal.colors[i]);
    outtextxy(0, y, pval);           //输出调色板颜色
    }
}
closegraph();                       //关闭图形系统
}

```

运行上述代码，首先初始化图形系统；然后获取当前调色板并将调色板信息转换为字符串，循环输出调色板颜色；最后待按下任意键后关闭图形系统。

6.45 按指定方式改变所有的调色板颜色

函数：setallpalette()

函数 setallpalette() 是按指定方式改变所有的调色板颜色，其原型如下。

```
void setallpalette(struct palettetype *p);
```

【参数】 参数 p 为调色板结构体。

【返回值】 在低分辨率图形显示模式下，该函数无效。若该函数调用成功，则把 p 指向的 palettetype 结构中的色彩值设置为当前调色板所有颜色的新色彩值。

【例 6-44】 下面的示例演示了 setallpalette() 函数的使用，使用该按指定方式改变调色板颜色，代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
#define CLIP_ON 1
void main() {
    int gdriver = DETECT, gmode, errorcode;
    struct palettetype *palette;

```



```

int errorcode,color,maxcolor;
initgraph(&graphdriver,&graphmode,"");           //初始化图形系统
errorcode = graphresult( );
if(errorcode!=grOk){
    printf("Graphics error %s\n",grapherrormsg(errorcode));
    exit(1);
}
palette=getdefaultpalette( );                   //获取默认调色板
printf("palette->size %d\n",palette->size);      //输出
getch( ); maxcolor = getmaxcolor();
for(color=0;color<maxcolor;color++) {
    setfillstyle(LINE_FILL,color);               //设置填充风格
    bar(20*(color-1),0,20*color,20);            //绘制二维图形码
    getch( );
}
if(palette->size>1){
    do
        setpalette(random(palette->size),random(palette->size));
        //设置调色板
    while(!kbhit());getch( );
}
setallpalette(palette);                          //设置所有调色板
closegraph( );
}

```

运行上述代码,首先初始化图形系统;然后获取默认调色板信息并输出,循环采用调色板颜色设置填充风格,并绘制二维图形码;最后设置所有调色板,待按下任意键后关闭图形系统。

6.46 设置有关当前调色板的信息

函数: setpalette()

函数 setpalette()用于设置有关当前调色板的信息,其原型如下。

```
void far setpalette(int index, int actual_color);
```

【参数】参数index为色彩值；参数actual_color为色彩值相应的颜色名。

【返回值】该函数没有返回值。

【例 6-45】下面的示例演示了setpalette()函数的使用，采用该函数设置当前调色板信息，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    maxcolor = getmaxcolor( );                 //取最大颜色
    ht = 2 * textheight("W");                 //获取字符串高度
    for (color=1; color<=maxcolor; color++) { //循环以不同颜色输出
        setcolor(color);                       //设置颜色
        sprintf(msg, "Color: %d", color);
        outtextxy(1, y, msg);                  //输出
        y += ht;
    }
    getch( );
    for (color=1; color<=maxcolor; color++) {
        setpalette(color, BLACK);               //设置调色板
        getch( );
    }
    closegraph( );                             //关闭图形系统
}
```

运行上述代码，首先初始化图形系统；然后取最大颜色，循环以不同颜色输出颜色信息，再循环设置调色板；最后待按下任意键后关闭图形系统。

6.47 关闭图形系统函数: closegraph()

函数 closegraph()用于关闭图形系统, 其原型如下。

```
void far closegraph(void);
```

【参数】该函数没有参数。

【返回值】该函数没有返回值。

【例 6-46】下面的示例演示了 closegraph() 函数的使用, 采用该函数关闭图形系统, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int x, y;
    int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    x = getmaxx( ) / 2;                         //获取屏幕中心位置
    y = getmaxy( ) / 2;
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    outtextxy(x, y, "Press a key to close the graphics system:");
                                                    //输出
    getch( );                                   //等待按任意键
    closegraph( );                             //关闭图形系统
    printf("We're now back in text mode.\n");   //输出提示信息
    printf("Press any key to halt:");
    getch( );
}
```

运行上述代码, 首先初始化图形系统; 然后获取屏幕中心位置, 设置文本对齐方式, 并在屏幕中心输出提示信息, 然后关闭图形系统, 再输出提示信息, 待按下任意键后关闭图形系统。

6.48 改变内部图形缓冲区的大小函数:

setgraphbufsize()

函数 setgraphbufsize()用于改变内部图形缓冲区大小, 其原型如下。

```
unsigned far setgraphbufsize(unsigned bufsize);
```

【参数】 参数bufsize为要改变的缓冲区大小。

【返回值】 该函数返回先前定义的图形缓冲区字节数。

【例 6-47】 下面的示例演示了setgraphbufsize()函数的使用, 采用该函数改变内部函数缓冲区大小, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define BUFSIZE 1000           //内部图形缓冲区

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int x, y, oldsize;
    char msg[80];
    oldsize = setgraphbufsize(BUFSIZE); //设置内部图形缓冲区大小
    initgraph(&gdriver, &gmode, ""); //初始化
    errorcode = graphresult(); //获取初始结果
    x = getmaxx() / 2; //获取屏幕中心位置
    y = getmaxy() / 2;
    sprintf(msg, "Graphics buffer size: %d", BUFSIZE);
    //输出缓冲区大小
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    outtextxy(x, y, msg);
    sprintf(msg, "Old graphics buffer size: %d", oldsize);
    outtextxy(x, y+textheight("W"), msg);
    getch();
    closegraph(); //关闭图形系统
```

}

运行上述代码, 首先设置内部图形缓冲区, 初始化图形系统; 然后获取屏幕中心位置, 设置文本对齐方式, 再输出其他文本; 最后待按下任意键后关闭图形系统。

注意: 必须在调用 `initgraph()` 函数之前调用该函数。

6.49 通过检测硬件确定图形驱动程序和模式

函数: `detectgraph()`

函数 `detectgraph()` 通过检测硬件确定图形驱动程序和模式, 其原型如下。

```
void far detectgraph(int far *graphdriver, int far *graphmode);
```

【参数】 函数把 `graphdriver` 所指向的整型变量设置为图形驱动程序的代码, 把 `graphmode` 所指向的整型变量设置为显卡支持的最高有效模式 (即该显卡能支持的最高分辨率)。

【返回值】 该函数返回适合于该显卡的图形驱动程序的代码 (也称等价值), 并存放在 `driver` 指向的变量中。若计算机系统中无图形硬件, 则由 `driver` 指向的变量设置为 -2。

【例 6-48】 下面的示例演示了 `detectgraph()` 函数的使用, 采用该函数检测硬件, 确定图形驱动程序和模式, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
char *dname[ ] = { "requests detection",
    "a CGA",
    "an MCGA",
    "an EGA",
```



```

    "a 64K EGA",
    "a monochrome EGA",
    "an IBM 8514",
    "a Hercules monochrome",
    "an AT&T 6300 PC",
    "a VGA",
    "an IBM 3270 PC"
};

void main( ){
    int gdriver, gmode, errorcode;
    detectgraph(&gdriver, &gmode);          //检测硬件
    errorcode = graphresult( );              //获取结果
    printf("You have %s video display \card.\n", dname[gdriver]);
                                           //输出
    printf("Press any key to halt:");
    getch( );
}

```

运行上述代码，首先检测硬件，并根据结果判断是否有误，若有误，输出提示信息，退出；然后清除屏幕，输出提示信息；最后待按下任意键后关闭图形系统。

6.50 绘制多边形函数：drawpoly()

函数 drawpoly() 用于绘制多边形，其原型如下。

```
void far drawpoly(int numpoints, int far *polypoints);
```

【参数】 参数 numpoints 为多边形的顶点数；参数 polypoints 指向整型数组，该数组中是多边形所有顶点(x, y)坐标值，即一系列整数对，x 坐标值在前。

【返回值】 该函数没有返回值。

【例 6-49】 下面的示例演示了 drawpoly() 函数的使用，采用该函数绘制多边形，代码如下。


```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int maxx, maxy;
    int poly[10];
    initgraph(&gdriver, &gmode, "");           //初始化
    errorcode = graphresult( );                 //读取初始结果
    maxx = getmaxx( );                         //获取最大坐标
    maxy = getmaxy( );
    poly[0] = 20;                              //第一个点坐标
    poly[1] = maxy / 2;
    poly[2] = maxx - 20;                       //第二个点坐标
    poly[3] = 20;
    poly[4] = maxx - 50;                       //第三个点坐标
    poly[5] = maxy - 20;
    poly[6] = maxx / 2;                       //第四个点坐标
    poly[7] = maxy / 2;
    poly[8] = poly[0];                         //绘制时不会自动关闭, 因此需要手动关闭
    poly[9] = poly[1];
    drawpoly(5, poly);                         //绘制多边形
    getch( );
    closegraph( );                             //关闭图形系统
}

```

运行上述代码, 首先初始化图形系统; 然后取屏幕最大坐标, 分别设置多边形的各个顶点坐标, 最后手动关闭顶头坐标, 绘制多边形, 待按下任意键后关闭图形系统。

6.51 绘制多边形并填充函数: fillpoly()

函数 fillpoly() 用于绘制多边形并填充, 其原型如下。

```
void far fillpoly(int numpoints, int far *polypoints);
```

【参数】 参数numpoints为多边形的顶点数；参数polypoints指向整型数组，该数组中是多边形所有顶点(x, y)坐标值，即一系列整数对，x坐标值在前。

【返回值】 该函数没有返回值。

【例 6-50】 下面的示例演示了fillpoly()函数的使用，采用该函数绘制多边形并填充，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int i, maxx, maxy;
    int poly[8];
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );                 //读取初始化结果
    maxx = getmaxx( );                          //获取屏幕最大坐标
    maxy = getmaxy( );
    poly[0] = 20;                               //第一个点
    poly[1] = maxy / 2;
    poly[2] = maxx - 20;                       //第二个点
    poly[3] = 20;
    poly[4] = maxx - 50;                       //第三个点
    poly[5] = maxy - 20;
    poly[6] = maxx / 2;                       //第四个点，关闭顶点
    poly[7] = maxy / 2;
    for (i=EMPTY_FILL; i<USER_FILL; i++) { //循环填充模式
        setfillstyle(i, getmaxcolor( )); //设置填充模式
        fillpoly(4, poly);               //绘制并填充多边形
        getch( );
    }
    closegraph( );                          //关闭图形系统
}
```


运行上述代码, 首先初始化图形系统; 然后获取屏幕最大坐标; 接着设置多边形各顶点, 以填充模式循环, 绘制并填充多边形; 最后待按下任意键后关闭图形系统。

6.52 填充有界区域函数: floodfill()

函数 floodfill() 用于填充有界区域, 其原型如下。

```
void far floodfill(int x, int y, int border);
```

【参数】 参数x和y为封闭图形内的任意一点; 参数border为边界的颜色, 也就是封闭图形轮廓的颜色。

【返回值】 该函数没有返回值。

【例 6-51】 下面的示例演示了floodfill()函数的使用, 采用该函数填充有界区域, 代码如下。

```
#include <graphics.h> //加入图形处理库
#include <stdlib.h> //加入标准工具库
#include <stdio.h> //加入标准输入输出库
#include <conio.h> //加入控制台输入输出库
#define CLIP_ON 1
void main() {
    int gdriver = DETECT, gmode, errorcode;
    int maxx, maxy;
    int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult();
    maxx = getmaxx(); //获取屏幕最大坐标
    maxy = getmaxy();
    setcolor(getmaxcolor()); //设置颜色
    setfillstyle(SOLID_FILL, getmaxcolor()); //设置填充风格
    rectangle(0, 0, maxx, maxy); //画屏幕边框
    circle(maxx / 3, maxy / 2, 50); //画圆
    circle(maxx / 2, 20, 100);
```



```

    circle(maxx-20, maxy-50, 75);
    circle(20, maxy-20, 25);
    getch();
    floodfill(2, 2, getmaxcolor());           //填充有界区域
    getch();
    closegraph();                             //关闭图形系统
}

```

运行上述代码，首先初始化图形系统；然后获取屏幕最大坐标，设置颜色和填充风格；接着绘制屏幕边框和圆，填充空白区域；最后待按下任意键后关闭图形系统。

6.53 获取当前图形模式函数：getgraphmode()

函数 `getgraphmode()` 用于获取当前图形模式，其原型如下。

```
int far getgraphmode(void);
```

【参数】 该函数没有参数。

【返回值】 该函数执行后返回当前图形模式。

【例 6-52】 下面的示例演示了 `getgraphmode()` 函数的使用，在程序中采用该函数获取当前图形模式，代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main() {
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult();
    midx = getmaxx() / 2;                       //获取屏幕中心坐标
}

```

```

midy = getmaxy( ) / 2;
mode = getgraphmode( );           //获取图形模式
sprintf(numname,
        "%d is the current mode number.",
        mode);                     //转换为字符串
sprintf(modename,
        "%s is the current graphics mode",
        getmodename(mode));
settextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
outtextxy(midx, midy, numname);         //输出
outtextxy(midx, midy+2*textheight("W"),
        modename);
closegraph( );                       //关闭图形系统
)

```

运行上述代码,首先初始化图形系统;然后获取屏幕中心坐标和图形模式,并将图形模式转换为字符串输出;最后待按下任意键后关闭图形系统。

6.54 设置当前图形模式函数: setgraphmode()

函数 setgraphmode()用于设置当前图形模式,其原型如下。

```
void far setgraphmode(int mode);
```

【参数】 参数mode为要设置的图形模式。

【返回值】 该函数没有返回值。

【例 6-53】 下面的示例演示了 setgraphmode()函数的使用,使用该函数设置当前图形模式,代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
#define CLIP_ON 1

```



```

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int x, y;
    int radius = 100;
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    x = getmaxx( ) / 2;                         //取当前屏幕中心位置
    y = getmaxy( ) / 2;
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    outtextxy(x, y, "Press any key to exit graphics:"); //输出
    getch( );
    restorecrtmode( );                         //恢复文本模式
    printf("We're now in text mode.\n");        //输出提示信息
    printf("Press any key to return to graphics mode:");
    getch( );
    setgraphmode(getgraphmode( ));             //设置图形模式
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    outtextxy(x, y, "We're back in graphics mode."); //在屏幕中心输出
    outtextxy(x, y+textheight("W"), "Press any key to halt:");
    getch( );
    closegraph( );                             //关闭图形系统
}

```

运行上述代码，首先初始化图形系统；然后获取当前屏幕中心位置，设置文本对齐方式并输出；接着恢复文本模式，输出提示信息；最后再设置图形模式，并输出提示信息，待按下任意键后关闭图形系统。

6.55 将指定区域的位图调入内存

函数：getimage()

函数 getimage()用于将指定区域的位图调入内存，其原型如下。


```
void far getimage(int left, int top, int right, int bottom, void far
*bitmap);
```

【参数】参数left和top表示左上角坐标；参数right和bottom为右下角坐标；参数bitmap为位图缓冲区。

【返回值】该函数没有返回值。

【例 6-54】下面的示例演示了getimage()函数的使用，采用该函数将指定区域的位图调入内存，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

void save_screen(void far *buf[4]);
void restore_screen(void far *buf[4]);
int maxx, maxy;
void main( ){
    int gdriver=DETECT, gmode, errorcode;
    void far *ptr[4];
    initgraph(&gdriver, &gmode, "");           //初始化
    errorcode = graphresult( );                 //检查结果
    maxx = getmaxx( );                          //获取屏幕最大坐标
    maxy = getmaxy( );
    rectangle(0, 0, maxx, maxy);               //画一矩形
    line(0, 0, maxx, maxy);                    //画一直线
    line(0, maxy, maxx, 0);
    save_screen(ptr);                           //保存屏幕
    getch( );
    cleardevice( );                             //清除图形
    restore_screen(ptr);                        //恢复屏幕
    getch( );
    closegraph( );                             //关闭图形系统
}

void save_screen(void far *buf[4]) {           //自定义函数保存屏幕
    unsigned size;
    int ystart=0, yend, yincr, block;
    yincr = (maxy+1) / 4;
```

```

yend = yincr;
size = imagesize(0, ystart, maxx, yend); //取图像尺寸
for (block=0; block<=3; block++) {
    if ((buf[block] = farmalloc(size)) == NULL) {
        closegraph( ); //清除图形
        printf("Error: not enough heap space in save_screen().\n");
        exit(1);
    }
    getimage(0, ystart, maxx, yend, buf[block]); //调入内存
    ystart = yend + 1;
    yend += yincr + 1;
}
}

void restore_screen(void far *buf[4]) { //自定义恢复屏幕
    int ystart=0, yend, yincr, block;
    yincr = (maxy+1) / 4;
    yend = yincr;
    for (block=0; block<=3; block++) {
        putimage(0, ystart, buf[block], COPY_PUT); //输出位图
        farfree(buf[block]); //释放
        ystart = yend + 1;
        yend += yincr + 1;
    }
}

```

运行上述代码，首先初始化图形系统；然后获取屏幕最大坐标，在屏幕上绘制矩形和直线，调用自定义函数保存当前屏幕，待按下任意键后恢复屏幕；最后待按下任意键后关闭图形系统。

6.56 在屏幕上输出位图函数：putimage()

函数 putimage() 用于在屏幕上输出位图，其原型如下。

```
void far putimage(int x, int y, void far *bitmap, int op);
```

【参数】参数x和y为输出屏幕图像的左上角，即输出图像的起始位置；

bitmap指向要输出的内存中图像; 参数op用于控制图像以何种方式输出到屏幕上。

【返回值】该函数没有返回值。

【例 6-55】下面的示例演示了putimage()函数的使用, 采用该函数在屏幕上输出位图, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define ARROW_SIZE 10
void draw_arrow(int x, int y);
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    void *arrow;
    int x, y, maxx;
    unsigned int size;
    initgraph(&gdriver, &gmode, "");           //初始化
    errorcode = graphresult( );                 //读取初始结果
    maxx = getmaxx( );                          //取屏幕最大坐标
    x = 0;
    y = getmaxy( ) / 2;
    draw_arrow(x, y);                           //调用自定义函数
    size = imagesize(x, y-ARROW_SIZE, x+(4*ARROW_SIZE),
y+ARROW_SIZE);                                //计算大小
    arrow = malloc(size);                       //分配内存
    getimage(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE,
arrow);                                         //调入内存
    while (!kbhit( )) {                         //直到有键按下结束
        putimage(x, y-ARROW_SIZE, arrow, XOR_PUT); //输出图像
        x += ARROW_SIZE;
        if (x >= maxx) x = 0;
        putimage(x, y-ARROW_SIZE, arrow, XOR_PUT); //输出图像
    }
    free(arrow);
    closegraph( );                             //关闭图形系统
}
```



```

void draw_arrow(int x, int y)    {           //自定义函数绘制箭头
    moveto(x, y);
    linerel(4*ARROW_SIZE, 0);
    linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
    linerel(0, 2*ARROW_SIZE);
    linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}

```

运行上述代码，首先初始化图形系统；然后获取最大坐标，调用自定义函数绘制箭头，待按下任意键后关闭图形系统。

6.57 获取给定图形驱动程序的模式范围

函数：getmoderange()

函数 getmoderange() 用于获取给定图形驱动程序的模式，其原型如下。

```

void far getmoderange(int graphdriver, int far *lomode, int far
*himode);

```

【参数】 参数 graphdriver 是用来调用此函数时，指定图形驱动程序等价值或宏名；返回的最小和最大有效值存放在参数指针 lomode 和 himode 分别指向的整型变量中。

【返回值】 该函数没有返回值。

【例 6-56】 下面的示例演示了 getmoderange() 函数的使用，获取给定图形驱动程序的模式，代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
#define CLIP_ON 1
void main( ){

```

```

int gdriver = DETECT, gmode, errorcode;
int midx, midy;
int radius = 100;
initgraph(&gdriver, &gmode, ""); //初始化图形系统
errorcode = graphresult();
midx = getmaxx() / 2; //获取屏幕中心位置
midy = getmaxy() / 2;
getmoderange(gdriver, &low, &high); //获取给定图形驱动程序的模式范围
sprintf(mrange, "This driver supports modes %d..%d", low, high);
//转换为字符串
settextjustify(CENTER_TEXT, CENTER_TEXT);
//设置文本对齐方式
outtextxy(midx, midy, mrange); //输出文本
closegraph(); //关闭图形系统
}

```

运行上述代码, 首先初始化图形系统; 然后获取屏幕中心位置, 获取给定图形驱动程序的模式范围并转换为字符串输出; 最后待按下任意键后关闭图形系统。

6.58 获取指定像素的颜色函数: getpixel()

函数 getpixel() 用于获取指定像素的颜色, 其原型如下。

```
int far getpixel(int x, int y);
```

【参数】 参数 x 和 y 为要取的坐标位置。

【返回值】 该函数返回指定位置的颜色, 如果点为清除状态, 返回 0; 否则, 返回非零值。

【例 6-57】 下面的示例演示了 getpixel() 函数的使用, 采用该函数获取指定像素的颜色, 代码如下。

```

#include <stdio.h> //加入标准输入输出库
#include <graphics.h> //加入图形库
void main() {

```

```

int i,j;
printf("V");
gotoxy(10,10);           //隐藏光标
for(i=0;i<8;i++)
for(j=0;j<16;j++) {
    if(getpixel(i,j))    //获取指定像素颜色
        putpixel(10+i,10+j,1); //绘制像素
    else putpixel(10+i,10+j,0); //清除该点
}
getchar( );
}

```

运行上述程序，首先清除屏幕，隐藏光标；然后循环获取屏幕上位置的颜色。若该点为清除状态，则在该点绘制像素；否则，清除该点。

6.59 在指定位置绘制像素函数：putpixel()

函数 putpixel() 用于在指定位置绘制像素，其原型如下。

```
void far putpixel (int x, int y, int pixelcolor);
```

【参数】参数x和y为屏幕上的坐标；参数pixelcolor的值为0表示清除指定位置的点，值为1表示在指定位置处画点，值为2表示将指定位置的点的状态取反。

【返回值】该函数没有返回值。

【例 6-58】下面的示例演示了putpixel()函数的使用，采用该函数在指定位置绘制像素，代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#include <dos.h>                //加入系统接口库
#define PIXEL_COUNT 1000

```



```

#define DELAY_TIME 100
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int i, x, y, color, maxx, maxy, maxcolor, seed;
    initgraph(&gdriver, &gmode, "");           //初始化
    errorcode = graphresult( );                 //读取初始化结果
    maxx = getmaxx( ) + 1;                     //取最大坐标
    maxy = getmaxy( ) + 1;
    maxcolor = getmaxcolor( ) + 1;             //取最大颜色
    while (!kbhit( )) {
        seed = random(32767);                  //随机数
        srand(seed);
        for (i=0; i<PIXEL_COUNT; i++) {
            x = random(maxx);
            y = random(maxy);
            color = random(maxcolor);
            putpixel(x, y, color);              //绘制像素
        }
        delay(DELAY_TIME);                     //暂停
        srand(seed);
        for (i=0; i<PIXEL_COUNT; i++) {
            x = random(maxx);
            y = random(maxy);
            color = random(maxcolor);
            if (color == getpixel(x, y))
                putpixel(x, y, 0);             //绘制像素
        }
    }
    getch( );
    closegraph( );                             //关闭图形系统
}

```

运行上述代码,首先初始化图形系统;然后获取当前屏幕最大坐标和最大颜色,随机绘制屏幕上某点的像素;最后待按下任意键后关闭图形系统。

6.60 将所有图形设置复位为它们的默认值

函数: graphdefaults()

函数 graphdefaults()用于将所有图形设置复位为默认值,其原型如下。

```
void far graphdefaults(void);
```

【参数】该函数没有参数。

【返回值】该函数没有返回值。

【例 6-59】下面的示例演示了 graphdefaults()函数的使用,将所有图形设置为默认值,代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int maxx, maxy;
    int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult( );
    maxx = getmaxx( );              //取最大坐标
    maxy = getmaxy( );
    setlinestyle(DOTTED_LINE, 0, 3); //设置线风格
    line(0, 0, maxx, maxy);         //画线
    outtextxy(maxx/2, maxy/3, "Before default values are restored.");
                                     //输出
    getch( );
    graphdefaults( );               //复位
    cleardevice( );                 //清除图形
    line(0, 0, maxx, maxy);         //画线
```

```

    outtextxy(maxx/2, maxy/3, "After restoring default values.");
    //输出
    getch( );
    closegraph( );
}

```

运行上述代码, 首先初始化图形系统; 然后获取最大坐标, 设置直线风格, 画直线和输出字符串; 接着再将所有图形设置复位, 重新画一直线和输出字符串; 最后待按下任意键后关闭图形系统。

6.61 返回一个错误信息串的指针函数:

grapherrormsg()

函数 `grapherrormsg()` 用于返回一个错误信息串的指针, 其原型如下。

```
char *far grapherrormsg(int errorcode);
```

【参数】 参数 `errorcode` 为错误编号。

【返回值】 该函数返回错误信息串指针。

【例 6-60】 下面的示例演示了 `grapherrormsg()` 函数的使用, 采用该函数返回错误信息串, 代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define NONSENSE -50
int main(void)
{
    int gdriver = NONSENSE, gmode, errorcode; //强制产生一个错误
    initgraph(&gdriver, &gmode, "");         //初始化图形系统
    errorcode = graphresult();                //读取初始化结果
    if (errorcode != grOk)                     //判断结果, 输出并解释出错信息
    {

```



```

    printf("Graphics error: %s/n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);                                //退出
}
line(0, 0, getmaxx(), getmaxy());          //画线
getch();                                    //清空
closegraph();                              //关闭图形系统
return 0;
}

```

运行上述代码, 首先初始化图形系统, 读取初始结果。若有错误, 根据错误编号返回错误信息, 退出程序; 否则, 画一直线, 待按下任意键后关闭图形系统。

6.62 用户可修改的图形存储区释放函数:

graphfreemem()

函数 graphfreemem() 用于释放用户可修改图形存储区, 其原型如下。

```
void far _graphfreemem(void far *ptr, unsigned size);
```

【参数】 参数 `ptr` 为释放的存储区; 参数 `size` 为要释放的存储区大小。

【返回值】 该函数没有返回值。

【例 6-61】 下面的示例演示了 graphfreemem() 函数的使用, 采用该函数释放用户可修改的图形存储区, 代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
void main() {

```

```

    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    clrscr( );           //清屏
    printf("Press any key to initialize graphics mode:");
    getch( );
    clrscr( );
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );               //读取初始结果
    midx = getmaxx( ) / 2;                     //获取屏幕中心位置
    midy = getmaxy( ) / 2;
    settextjustify(CENTER_TEXT, CENTER_TEXT); //设置文本对齐方式
    outtextxy(midx, midy, "Press any key to exit graphics mode:");
                                           //输出
    closegraph( );
}

void far * far _graphgetmem(unsigned size){
    printf("_graphgetmem called to allocate %d bytes.\n", size);
    printf("hit any key:");
    getch( );
    printf("\n");
    return farmalloc(size);
}

void far _graphfreemem(void far *ptr, unsigned size){
    printf("_graphfreemem called to free %d bytes.\n", size);
    printf("hit any key:");
    getch( );
    printf("\n");
    farfree(ptr);
}

```

运行上述程序, 首先清除屏幕, 输出提示消息, 初始化图形系统; 然后获取屏幕中心位置, 输出提示信息, 待按下任意键后关闭图形系统。

6.63 用户可修改的图形存储区分配函数:

`_graphgetmem()`

函数 `_graphgetmem()` 为用户可修改的图形存储区分配函数, 其原型如下。

```
void far *far _graphgetmem(unsigned size);
```

【参数】 该函数没有参数。

【返回值】 该函数执行后, 将返回一个未使用内存区大小, 返回值的类型为无符号数。

【例 6-62】 下面的示例演示了 `_graphgetmem()` 函数的使用, 使用该函数获取当前环境中未使用内存的大小, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    clrscr( );                  //清屏
    printf("Press any key to initialize graphics mode:");
    getch( );
    clrscr( );
    initgraph(&gdriver, &gmode, ""); //初始化
    errorcode = graphresult( );      //获取初始结果
    midx = getmaxx( ) / 2;          //获取屏幕中心位置
    midy = getmaxy( ) / 2;
    setttextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(midx, midy, "Press any key to exit graphics mode:");
    //输出
    getch( );
    closegraph( );               //关闭图形系统
}
```



```

void far * far _graphgetmem(unsigned size){
    printf("_graphgetmem called to allocate %d bytes.\n", size);
    printf("hit any key:");
    getch( );
    printf("\n");
    return farmalloc(size);
}

void far _graphfreemem(void far *ptr, unsigned size){
    printf("_graphfreemem called to free %d bytes.\n", size);
    printf("hit any key:");
    getch( );
    printf("\n");
    farfree(ptr);
}

```

运行上述代码, 首先初始化图形系统; 然后获取屏幕中心位置, 且输出相应信息; 最后待按下任意键后关闭图形系统。

6.64 返回保存位图像所需的字节数

函数: imagesize()

函数 imagesize() 返回保存位图所需的字节数, 其原型如下。

```
unsigned far imagesize(int left, int top, int right, int bottom);
```

【参数】 参数left和top为左上角坐标; 参数right和bottom为右下角坐标。

【返回值】 该函数返回字节数。

【例 6-63】 下面的示例演示了imagesize()函数的使用, 采用该函数返回位置所需要的字节, 代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库

```

```

#include <conio.h> //加入控制台输入输出库
#define ARROW_SIZE 10
void draw_arrow(int x, int y);
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    void *arrow;
    int x, y, maxx;
    unsigned int size;
    initgraph(&gdriver, &gmode, ""); //初始化
    errorcode = graphresult( ); //读取初始结果
    maxx = getmaxx( ); //最大光标
    x = 0;
    y = getmaxy( ) / 2;
    draw_arrow(x, y); //自定义函数
    size = imagesize(x, y-ARROW_SIZE, x+(4*ARROW_SIZE),
y+ARROW_SIZE); //计算大小
    arrow = malloc(size); //分配
    getimage(x, y-ARROW_SIZE, x+(4*ARROW_SIZE), y+ARROW_SIZE,
arrow); //保存图像
    while (!kbhit( )) { //有键按下结束
        putimage(x, y-ARROW_SIZE, arrow, XOR_PUT); //输出位图
        x += ARROW_SIZE;
        if (x >= maxx)
            x = 0;
        putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
    }
    free(arrow);
    closegraph( );
}

void draw_arrow(int x, int y){
    moveto(x, y);
    linerel(4*ARROW_SIZE, 0);
    linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
    linerel(0, 2*ARROW_SIZE);
    linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}

```

运行上述代码，首先初始化图形系统，然后获取屏幕中心位置，

调用自定义函数画箭头, 将当前图像保存到内存, 待按下任意键后关闭图形系统。

6.65 在指定两点间画直线函数: line()

函数 line() 用于在指定两点间画直线, 其原型如下。

```
void far line(int x0, int y0, int x1, int y1);
```

【参数】 参数x0 和y0 为第一个点的坐标; 参数x1 和y1 为第二个点的坐标。

【返回值】 该函数没有返回值。

【例 6-64】 下面的示例演示了line()函数的使用, 采用该函数绘制直线, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int xmax, ymax;
    int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult();
    setcolor(getmaxcolor());         //设置颜色
    xmax = getmaxx();                //取最大坐标
    ymax = getmaxy();
    line(0, 0, xmax, ymax);          //画直线
    getch();
    closegraph();                    //关闭图形系统
}
```

运行上述代码, 首先初始化图形系统; 然后设置颜色, 获取当前

屏幕最大坐标；接着再画一直线，待按下任意键后关闭图形系统。

6.66 在指定两点间画直线函数：lineto()

函数 lineto() 用于在指定两点间画直线，其原型如下。

```
void lineto(int x,int y);
```

【参数】 参数x、y为指定点的坐标。

【返回值】 该函数没有返回值。

【例 6-65】 下面的示例演示了 lineto() 函数的使用，采用该函数在指定两点间画直线，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;  int radius = 100;
    char msg[80];
    initgraph(&gdriver, &gmode, "");           //初始化图形系统
    errorcode = graphresult( );
    moveto(20, 30);                             //移到指定位置
    sprintf(msg, " (%d, %d)", getx( ), gety( ));
    outtextxy(20, 30, msg);                      //输出
    lineto(100, 100);                            //画直线
    sprintf(msg, " (%d, %d)", getx( ), gety( ));
    outtext(msg);                                //输出
    closegraph( );                              //关闭图形系统
}
```

运行上述程序，首先初始化图形系统；然后移动光标，输出信息；接着从当前位置到指定位置间绘制直线，待按下任意键后关闭图形系统。

6.67 从当前位置点到当前点绘制直线

函数: linerel()

函数 linerel() 用于从当前位置到当前点绘制直线, 其原型如下。

```
void far linerel(int dx, int dy);
```

【参数】 参数 dx、dy 分别为水平偏移距离和垂直偏移距离。该函数调用后, 当前位置变为增加偏移距离后的位置。

【返回值】 该函数没有返回值。

【例 6-66】 下面的示例演示了 linerel() 函数的使用, 采用该函数从当前位置到当前点绘制直线, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库
#define CLIP_ON 1
void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int midx, midy;
    int radius = 100;
    initgraph(&gdriver, &gmode, ""); //初始化图形系统
    errorcode = graphresult( );
    moveto(20, 30);                //移到指定位置
    sprintf(msg, " (%d, %d)", getx( ), gety( ));
    outtextxy(20, 30, msg);        //输出
    linerel(100, 100);             //画直线
    sprintf(msg, " (%d, %d)", getx( ), gety( ));
    outtext(msg);                  //输出
    closegraph( );                 //关闭图形系统
}
```

运行上述代码, 首先初始化图形系统; 然后移动光标, 输出信息; 接着再从当前位置到指定位置间绘制直线, 待按下任意键后关闭图形系统。

6.68 选择低亮度字符函数: lowvideo()

函数 lowvideo() 用于选择低亮度字符, 其原型如下。

```
void lowvideo(void);
```

【参数】该函数没有参数。

【返回值】该函数没有返回值。

【例 6-67】下面的示例演示了 lowvideo() 函数的使用, 采用该函数选择低亮度字符, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库

void main( ){
    clrscr( );                  //清屏
    highvideo( );               //高亮
    cprintf("High Intesity Text\r\n"); //输出
    lowvideo( );                //低亮
    gotoxy(1,2);                //移动光标
    cprintf("Low Intensity Text\r\n");
}
```

运行上述程序, 首先清除屏幕, 高亮显示字符, 输出字符串; 然后低亮度显示字符, 输出字符串。

6.69 登录已连接进来的图形驱动程序代码

函数: registerbgidriver()

函数 registerbgidriver() 用于登录已连接进来的图形驱动器程序代码, 其原型如下。

```
int registerbgidriver(void(*driver)(void));
```


【参数】参数driver为要连接的图形驱动程序。

【返回值】若该函数执行成功，则返回1；否则，返回0。

【例6-68】下面的示例演示了registerbgidriver()函数的使用，采用该函数登录已连接进来的图形驱动程序，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>             //加入标准输入输出库
#include <conio.h>             //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    errorcode = registerbgidriver(EGAVGA_driver); //登录连接驱动
    if (errorcode < 0) { //判断错误
        printf("Graphics error: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);
    }
    initgraph(&gdriver, &gmode, ""); //初始化图形
    errorcode = graphresult(); //读取初始结果
    line(0, 0, getmaxx(), getmaxy()); //画直线
    getch();
    closegraph(); //关闭图形系统
}
```

运行上述程序，首先登录已连接的图形驱动程序，若有错，输出提示信息，并退出；否则，初始化图形系统，并对初始化结果进行判断，最后画一直线，关闭图形系统。

6.70 将屏幕模式恢复为先前的 imitgraph 设置

函数: restorecrtmode()

函数 restorecrtmode() 用于将屏幕模式恢复为 imitgraph，其原型如下。

```
void far restorecrtmode(void);
```

【参数】 该函数没有参数。

【返回值】 该函数没有返回值。

【例 6-69】 下面的示例演示了 `restorecrtmode()` 函数的使用，采用该函数将屏幕模式恢复，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int x, y;
    initgraph(&gdriver, &gmode, ""); //初始化
    errorcode = graphresult();        //读取结果
    x = getmaxx() / 2;                //取屏幕中心位置
    y = getmaxy() / 2;
    setttextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x, y, "Press any key to exit graphics:"); //输出
    getch();
    restorecrtmode();                 //恢复
    printf("We're now in text mode.\n"); //输出消息
    printf("Press any key to return to graphics mode:");
    getch();
    setgraphmode(getgraphmode());     //设置
    setttextjustify(CENTER_TEXT, CENTER_TEXT);
    outtextxy(x, y, "We're back in graphics mode."); //输出
    outtextxy(x, y+textheight("W"), "Press any key to halt:");
    getch();
    closegraph();                     //关闭图形系统
}
```

运行上述代码，首先初始化图形系统；然后获取屏幕中心位置，并在该位置输出信息；接着恢复设置，再次输出信息，待按下任意键后关闭图形系统。

6.71 设置图形输出活动页函数:

setactivepage()

函数 setactivepage() 设置图形输出活动页, 其原型如下。

```
void far setactivepage(int pagenum);
```

【参数】参数 pagenum 为要设置的页号。

【返回值】该函数没有返回值。

【例 6-70】下面的示例演示了 setactivepage() 函数的使用, 在程序中设置图形输出活动, 代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>             //加入标准工具库
#include <stdio.h>              //加入标准输入输出库
#include <conio.h>              //加入控制台输入输出库

void main( ){
    int gdriver = EGA, gmode = EGAHI, errorcode;
    int x, y, ht;
    initgraph(&gdriver, &gmode, "");           //初始化图形
    errorcode = graphresult( );                 //判断初始结果
    x = getmaxx( ) / 2;                         //获取屏幕中心位置
    y = getmaxy( ) / 2;
    ht = textheight("W");
    setactivepage(1);                           //设置活动页
    line(0, 0, getmaxx( ), getmaxy( ));         //画直线
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //输出
    outtextxy(x, y, "This is page #1:");
    outtextxy(x, y+ht, "Press any key to halt:");
    setactivepage(0);                           //设置活动页
    outtextxy(x, y, "This is page #0.");         //输出
    outtextxy(x, y+ht, "Press any key to view page #1:");
    getch( );
    setvisualpage(1);                           //设置为可见
    getch( );
```



```
closegraph( );           //关闭图形
}
```

运行上述代码，首先初始化图形系统；然后获取屏幕中心位置，设置活动页，画直线输出信息，接着设置活动页为可见；最后待按下任意键后关闭图形系统。

6.72 设置可见图形页号函数：setvisualpage()

函数 setvisualpage() 用于设置可见图形页号，其原型如下。

```
void far setvisualpage(int pagenum);
```

【参数】 参数 pagenum 为要设置的页号。

【返回值】 该函数没有参数。

【例 6-71】 下面的示例演示了 setvisualpage() 函数的使用，使用该函数设置可见图形页号，代码如下。

```
#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库

void main( ){
    int gdriver = EGA, gmode = EGAHI, errorcode;
    int x, y, ht;
    initgraph(&gdriver, &gmode, "");           //初始化
    errorcode = graphresult( );                 //读取结果
    x = getmaxx( ) / 2;                         //获取屏幕中心坐标
    y = getmaxy( ) / 2;
    ht = textheight("W");
    setactivepage(1);                           //设置活动页
    line(0, 0, getmaxx( ), getmaxy( ));        //画直线
    setttextjustify(CENTER_TEXT, CENTER_TEXT); //输出
    outtextxy(x, y, "This is page #1:");
    outtextxy(x, y+ht, "Press any key to halt:");
```

```

    setactivepage(0);           //设置活动页
    outtextxy(x, y, "This is page #0."); //输出
    outtextxy(x, y+ht, "Press any key to view page #1:");
    getch();
    setvisualpage(1);          //设置可见
    getch();
    closegraph();              //关闭
}

```

运行上述代码, 首先初始化图形系统; 然后获取屏中心坐标, 设置活动页画直线, 并输出字符串, 再设置可见; 最后待按下任意键后关闭图形系统。

6.73 设置当前画线宽度和类型函数:

setlinestyle()

函数 setlinestyle ()用于设置当前画线宽度和类型, 其原型如下。

```
void far setlinestyle(int linestyle, unsigned upattern);
```

【参数】 参数linestyle为画线类型; 参数upattern为模式。

【返回值】 该函数没有返回值。

【例 6-72】 下面的示例演示了setlinestyle()函数的使用, 采用该函数设置画线宽度和类型, 代码如下。

```

#include <graphics.h>           //加入图形处理库
#include <stdlib.h>              //加入标准工具库
#include <stdio.h>               //加入标准输入输出库
#include <conio.h>               //加入控制台输入输出库
char *lname[ ] = {
    "SOLID_LINE",
    "DOTTED_LINE",
    "CENTER_LINE",
    "DASHED_LINE",
}

```

```

    "USERBIT_LINE"
};

void main( ){
    int gdriver = DETECT, gmode, errorcode;
    int style, midx, midy, userpat;
    char stylestr[40];
    initgraph(&gdriver, &gmode, "");           //初始化图形
    errorcode = graphresult( );                 //读取初始结果
    midx = getmaxx( ) / 2;                      //获取屏幕中心坐标
    midy = getmaxy( ) / 2;
    userpat = 1;
    for (style=SOLID_LINE; style<=USERBIT_LINE; style++) {
        setlinestyle(style, userpat, 1);        //设置直线风格
        strcpy(stylestr, lname[style]);         //将风格转换为字符串
        line(0, 0, midx-10, midy);             //画直线
        rectangle(0, 0, getmaxx( ), getmaxy( )); //画矩形
        outtextxy(midx, midy, stylestr);        //输出
        getch( );
        cleardevice( );                         //清除图形
    }
    closegraph( );
}

```

运行上述代码，首先初始化图形系统；然后获取屏幕中心坐标，循环设置直线风格，将风格转换为字符串并输出；最后待按下任意键后关闭图形系统。

第7章 动态内存管理函数库: alloc.h

在程序设计中,常常需要对内存进行相应的操作。C语言专门提供一个针对内存操作的函数库——alloc.h,该文件库中存在的常用函数主要包括内存申请、分配、释放等内存管理函数。

7.1 更改数据段空间分配函数: brk()

函数 brk()用于修改数据段的空间分配,其原型如下。

```
extern int brk(void *endds);
```

【参数】参数*endds为需要进行更改的内存空间的分配量,其类型为指针类型。

【返回值】该函数执行成功后将返回 0; 否则, 返回 1。

【例 7-1】下面的示例演示了 brk()函数的使用,在程序中动态地更改内存数据段空间,代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <alloc.h> //加入内存管理库
void main( ) {
    char *ptr; //定义一个字符型指针
    printf("Changing allocation with brk( )\n"); //输出提示信息
    ptr = malloc(1); //分配一个字节长度的内存
    printf("Before brk( ) call: %lu bytes free\n", coreleft( )); //获取未使用的内存大小
    brk(ptr+1000); //更改数据段
    printf(" After brk( ) call: %lu bytes free\n", coreleft( )); //获取分配后的未使用内存大小
}
```

运行上述程序，首先定义一个字符类型的指针；然后在内存数据区分配一个字节长度的内存区；再使用 `prk()` 函数对数据段内存区进行重新分配；最后输出前后的未使用内存区大小，以验证是否分配成功，其输出结果如下。

```
Changing allocation with brk( )
Before brk( ) call: 98629600 bytes free
After brk( ) call: 98628600 bytes free
```

注意：这里内存大小与运行该程序的环境有关，其数据大小因所在平台的内存大小不同而不同。

7.2 获取未使用内存大小函数：`coreleft()`

函数 `coreleft()` 用于获取当前未使用的内存区大小，其原型如下。

```
unsigned coreleft(void);
```

【参数】 该函数没有参数。

【返回值】 该函数执行后将返回一个未使用内存区大小，返回值的类型为无符号数。

【例 7-2】 下面的示例演示了 `coreleft()` 函数的使用，使用该函数获取当前环境中未使用内存的大小，代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <alloc.h>           //加入内存管理库
void main( ) {
    printf("The difference between the highest allocated block
and\n");
    printf("the top of the heap is: %lu bytes\n", (unsigned long)
coreleft( ));                //获取未使用内存区大小
}
```

运行上述代码，将采用 `coreleft()` 函数获取当前未使用内存区大小，并输出如下结果。

The difference between the highest allocated block and the top of the heap is: 63488 bytes

注意: 这里的未使用内存大小会与程序运行平台、物理内存的大小、执行的程序所占内存大小有关, 因此在实际执行时可能不会得到以上数据。

7.3 申请堆栈空间函数: farcalloc()

函数 farcalloc() 用于从远堆栈申请指定大小的空间, 其原型如下。

```
void far *farcalloc(unsigned long units, unsigned long unitsz);
```

【参数】 参数 units 表示要分配的空间大小; 参数 unitsz 表示空间的存储大小。

【返回值】 该函数按所给参数分配空间, 没有返回值。

【例 7-3】 下面的示例演示了 farcalloc() 函数的使用, 根据所给的空间大小参数分配空间, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <alloc.h> //加入内存管理库
#include <string.h> //加入字符串库
#include <dos.h> //加入系统接口库
void main() {
    char far *fptr; //定义指针
    char *str = "Hello"; //定义指针
    fptr = farcalloc(10, sizeof(char)); //给 far 指针分配内存
    movedata(FP_SEG(str), FP_OFF(str),
        FP_SEG(fptr), FP_OFF(fptr),
        strlen(str)); //将 Hello 复制至所分配的内存
    printf("Far string is: %Fs\n", fptr); //输出内容
    farfree(fptr); //释放内存
}
```

运行上述代码, 首先定义指针; 然后给指针分配内存空间, 再将

指定的内容复制到所分配的内存空间并进行输出；最后释放所申请的空间，其输出结果如下。

```
Far string is: Hello
```

7.4 获取空余存储区空间大小函数：

`()` `farcoreleft()`

函数 `farcoreleft()` 用于获取当前空余存储区空间大小，其原型如下。

```
long farcoreleft(void);
```

【参数】 该函数没有参数。

【返回值】 该函数返回一个长整型的空余存储区空间大小。

【例 7-4】 下面的示例演示了 `farcoreleft()` 函数的使用，采用该函数获取当前系统中空余存储区空间大小，代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <alloc.h>           //加入内存管理库
void main( ){
    printf("The difference between the\highest allocated block in
the far\n");                //输出提示信息
    printf("heap and the top of the far heap\is: %lu bytes\n",
farcoreleft( ));            //输出空余存储空间大小
}
```

运行上述程序，首先采用 `farcoreleft()` 函数获取空余存储区空间大小，并采用 `printf()` 函数进行输出，其输出结果如下。

```
The difference between the highest allocated block in the far
heap and the top of the far heap is: 527264 bytes
```

注意：空余存储区空间大小会根据读者所运行的环境不同而得到不同结果。

7.5 释放堆中空间函数: farfree()

函数 farfree() 用于释放将所申请的堆空间, 其原型如下。

```
void farfree(void);
```

【参数】该函数没有参数。

【返回值】该函数直接释放堆中的空间, 没有返回值。

【例 7-5】下面的示例演示了 farfree() 函数的使用, 采用该函数释放已申请的堆空间, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <alloc.h>           //加入内存管理库
#include <string.h>          //加入字符串库
#include <dos.h>             //加入系统接口库

void main() {
    char far *fp;
    char *s = "this is test char.";
    fp = farcalloc(10, sizeof(char)); //申请空间
    movedata(FP_SEG(s), FP_OFF(s),
             FP_SEG(fp), FP_OFF(fp),
             strlen(s)); //将所定义的内容复制至所申请的空间
    printf("Far string is: %Fs\n", fp); //输出内容
    farfree(fp); //释放空间
}
```

运行上述程序, 首先申请指定长度的内存空间; 然后将所定义的变量复制到所申请的空间并进行输出; 最后释放所申请的空间。

7.6 存储块分配空间函数: farmalloc()

函数 farmalloc() 用于分配指定大小的存储块空间, 其原型如下。

```
void far *farmalloc(unsigned long size);
```

【参数】参数size表示要分配的空间大小，其类型为无符号长整型。

【返回值】该函数直接分配所指定大小的空间，没有返回值。

【例 7-6】下面的示例演示了farmalloc()函数的使用，采用该函数分配指定大小的存储块空间，代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <alloc.h> //加入内存管理库
#include <string.h> //加入字符串库
#include <dos.h> //加入系统接口库
int main(void)
{
    char far *fptr;
    char *str = "Hello";
    fptr = farmalloc(10); //分配存储块空间
    movedata(FP_SEG(str), FP_OFF(str),
             FP_SEG(fptr), FP_OFF(fptr),
             strlen(str)); //将所定义的内容复制至该处
    printf("Far string is: %Fs\n", fptr); //输出内容
    farfree(fptr); //释放空间
    return 0;
}
```

运行上述程序，首先申请空间，将定义的内容复制到所申请的内存空间并进行输出，然后释放所申请的内存空间。

7.7 存储块空间调整函数：farrealloc()

函数 farrealloc()按指定空间大小进行存储块空间的调整，其原型如下。

```
void far *farrealloc(void far *block, unsigned long newsize);
```

【参数】参数block表示要进行调整的块空间；参数newsize表示要进行调整的空间大小。

【返回值】该函数返回调整后的内存指针。

【例 7-7】下面的示例演示了 `farrealloc()` 函数的使用, 采用该函数对已分配的内存空间进行重新分配, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <alloc.h>           //加入内存管理库
void main( ){
    char far *fptr;
    fptr = farmalloc(10);     //申请空间
    printf("First address: %Fp\n", fptr); //输出空间地址信息
    fptr = farrealloc(fptr,20); //调整空间
    printf("New address : %Fp\n", fptr); //输出空间信息
    farfree(fptr);           //释放空间
}
```

运行上述程序, 首先申请指定大小的内存空间, 输出其地址信息; 然后对该空间按指定大小进行调整, 并输出其地址信息, 其输出结果如下。

```
First address: 1FA2:0004
New address : 1FA3:0004
```

7.8 释放已分配 DOS 内存块函数: `freemem()`

函数 `freemem()` 用于释放指定的已分配的 DOS 内存块, 其原型如下。

```
int freemem(unsigned seg);
```

【参数】参数 `seg` 表示要释放的内存块。

【返回值】该函数释放 DOS 内存块后, 将返回表示成功与失败的整数, 即 0 和 1。

【例 7-8】下面的示例演示了 `freemem()` 函数的使用, 采用该函数释放已分配的 DOS 内存块, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <alloc.h>           //加入内存管理库
```

```

#include <dos.h>                                //加入系统接口库
void main( ){
    unsigned int size, segp;                      //定义变量
    int stat; size = 64;
    stat = allocmem(size, &segp);                //分配内存块
    if (stat < 0)                                //分配成功
        printf("Allocated memory at segment:\ %x\n", segp);
    else                                          //分配失败
        printf("Failed: maximum number of\paragraphs available is
%u\n", stat);
    freemem(segp);                               //释放内存块空间
}

```

运行上述程序，首先分配指定大小的 DOS 内存块，判断分配是否成功，并输出相应的提示信息；然后释放所申请的 DOS 内存块，其输出结果如下。

```
Allocated memory at segment: 1f43
```

7.9 改变数据段空间位置函数：sbrk()

函数 sbrk() 用于改变数据段空间的位置，其原型如下。

```
char *sbrk(int incr);
```

【参数】 参数 incr 表示要修改的数据段空间位置值。

【返回值】 该函数根据所给的位置值改变数据段空间位置，并返回一个 char 型的指向该位置的指针。

【例 7-9】 下面的示例演示了 sbrk() 函数的使用，采用该函数改变已分配的内存数据段空间位置，代码如下。

```

#include <stdio.h>                                //加入标准输入输出库
#include <alloc.h>                                //加入内存管理库
void main( ){
    printf("Changing allocation with sbrk( )\n"); //显示提示信息
}

```

```

printf("Before sbrk( ) call: %lu bytes free\n",
//显示改变前的内存信息
(unsigned long) coreleft( ));
sbrk(1000); //改变数据段空间位置
printf(" After sbrk( ) call: %lu bytes free\n", (unsigned long)
coreleft( )); //输出改变后的空间位置
}

```

运行上述程序, 首先输出当前的内存信息, 然后改变数据段空间位置, 再输出改变后的内存地址信息, 其输出结果如下。

```

Changing allocation with sbrk( )
Before sbrk( ) call: 63472 bytes free
After sbrk( ) call: 62464 bytes free

```


第 8 章 目录操作函数库: dir.h

在编程时,很多时候需要对文件的目录进行各种操作。为了方便编程,C语言给编程人员提供了一个专门针对目录操作的函数库——dir.h,该文件库中存在的常用函数包含工作目录的创建、删除及搜索等函数。

8.1 更改工作目录函数: chdir()

函数 chdir()用于改变当前工作目录为指定目录,其原型如下。

```
int chdir(char *path)
```

【参数】参数path表示新设置的路径名称。

【返回值】返回值为 0 表示改变工作目录操作成功;为-1 表示改变路径操作失败。

【例 8-1】下面的示例演示了 chdir()函数的使用,代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <stdlib.h>          //加入标准工具库
#include <dir.h>              //加入目标操作函数库
void main( ) {
    if (!chdir("C:\\photo")) //更改当前路径为 C: photo, 返回 0 表示更改
成功, 否则失败
        printf("路径修改成功");
}
```

运行上述程序,将当前路径更改为 C:\photo 目录,编译并运行程序。如果成功,输出“路径修改成功”,main()函数返回值为 0,输出结果如下。

路径修改成功

注意: 在路径表示的时候, 由于“\”为转义字符的标识, 因此路径的分隔符需要使用“\\”表示。

8.2 搜索磁盘目录函数: findfirst()

函数 findfirst() 用于寻找与 fname 相匹配的第一个文件名称, 其原型如下。

```
int findfirst(char *fname, struct ffblk *ptr, int attrib)
```

【参数】参数 fname 表示要匹配的文件名称, 可包含通配符“*”或“?”, 其中, “*”表示任意多个字符, “?”表示一个任意字符; 参数 struct ffblk 表示系统定义的 ffblk 类型结构变量名, 其原型如下。

```
struct ffblk{
    char ff_reserved[21];           //DOS 保留字
    char ff_attrib;                 //被检索的文件属性
    unsigned ff_fctime;            //文件的最后修改时间
    unsigned ff_fdate;            //文件的最后修改日期
    long ff_fsize;                 //文件大小
    char ff_name[13];              //文件名
}
```

其中, attrib 表示要查找的文件属性, 属性常数可选值有 6 个, 如 fa_ronly 表示只读文件, fa_label 表示卷标号, fa_hidden 表示隐藏文件, fa_dirac 表示目录, fa_system 表示系统文件, fa_arch 表示档案。

【返回值】返回值为 0 表示寻找匹配文件名成功; 为 -1 表示寻找匹配文件名失败。

【例 8-2】下面的示例演示了 findfirst() 函数的使用, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <dir.h>              //加入目标操作函数库
void main( )
{
```

```

    struct ffblk ffblk;           //定义 ffblk 类型的结构体变量 ffblk
    int done;
    printf("Directory listing of *.*");
    done=findfirst("D:\\*.doc",&ffblk,0); //查找文件扩展名为 doc 文
件, 保存在 ffblk 结构体变量中, 默认属性
    printf("%s\n", ffblk.ff_name); //查找成功, 则输出文件信息
}

```

运行上述代码后, 将在 D 盘根目录下查找到的第一个 doc 类型的文件名称输出到屏幕。

8.3 搜索磁盘目录函数: findnext()

函数 findnext() 用于寻找与 fname 相匹配的下一个文件名称, 其原型如下。

```
int findnext(struct ffblk *ptr)
```

【参数】参数 fname 表示要匹配的文件名称, 可包含通配符“*”或“?”, 其中“*”表示任意多个字符, “?”表示一个任意字符; 参数 struct ffblk 表示系统定义的 ffblk 类型结构变量名, 其原型参考 8.2 节。

【返回值】返回值为 0 表示寻找匹配文件名成功; 为-1 表示寻找匹配文件名失败。

【例 8-3】下面的示例演示了 findnext() 函数的使用, 代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <dir.h>             //加入目标操作函数库
void main( )
{
    struct ffblk ffblk;      //定义 ffblk 类型的结构体变量 ffblk
    int done;
    printf("Directory listing of *.*");
    done=findfirst("D:\\*.doc",&ffblk,0); //查找文件扩展名为 doc 的文
件, 保存在 ffblk 结构体变量中, 默认属性
    while (!done) {

```



```

        printf("%s\n", ffblk.ff_name);        //查找成功,则输出文件信息
        done=findnext(&ffblk);                //下一个匹配项赋值给 done
    }
}

```

运行上述代码后,将在 D 盘根目录下查找到的 doc 类型的文件名称输出到屏幕,结果如下。

```

C 语言作业.doc
工作计划.doc
材料.doc

```

注意: 上述运行结果与所使用计算机 D 盘下的文件有关。

8.4 指定当前目录函数: getcurdir()

函数 `getcurdir()` 用于获取指定驱动器上的当前工作路径,其原型如下。

```
int getcurdir(int drive, char *dir)
```

【参数】 参数 `drive` 表示驱动器名,取值为 0 表示默认驱动器;值为 1 表示 A 驱动器;值为 2 表示 B 驱动器,以此类推。参数 `dir` 表示路径字符串,用来存放目录名,不包括驱动器名,不以反斜杠开始。

【返回值】 返回值为 0 表示获取路径操作成功;为 1 表示获取路径操作失败。

【例 8-4】 下面的示例演示了 `getcurdir()` 函数的使用,代码如下。

```

#include <dir.h>                                //加入目标操作函数库
#include<stdio.h>                                //加入标准输入输出库
void main( ){
    char buf[50];                                //定义长度为 50 的字符数组
    getcurdir(0,buf); //调用函数 getcurdir( ),获取当前路径,驱动器为默认
    printf("The current directory is: %s",&buf); //输出获取到的当前工
作目录
}

```

运行上述程序，得到当前的路径，输出结果如下。

```
The current directory is:PROGRA~1\winYES\TCPP30H\PROJECT
```

注意：上述程序中，drive 参数的值为 0 时，获取的是当前程序所在的驱动器；获取的路径名称不包含驱动器名称，并且不以反斜杠开始。

8.5 获取当前工作目录函数：getcwd()

函数 getcwd() 用于获取当前工作目录的绝对路径，其原型如下。

```
char *getcwd(char *dir,int len)
```

【参数】参数 dir 表示指定的路径字符串；参数 len 表示路径的最大长度。

【返回值】如果 dir 指定的空间长度大于等于当前路径的长度，将当前路径复制到参数 dir 所指的内存空间中；如果 dir 指定的长度小于路径长度，返回指向已经分配的内存缓冲区地址。

【例 8-5】下面的示例演示了 getcwd() 函数的使用，代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <dir.h>              //加入目标操作函数库
void main( ){
    char buffer[50];          //定义 buffer 长度为 50
    getcwd(buffer,50);        //获取当前工作目录
    printf("The current directory is: %s",&buffer);
                               //输出获取到的当前工作目录
}
```

运行上述程序，获取当前路径，输出结果如下。

```
The current directory is:C:\PROGRA~1\winYES\TCPP30H\PROJECT
```

注意：上述程序的运行结果与当前路径有关；而且 buffer 的长度可以与 len 参数的长度不一致，如果 len 指定的长度小于路径的长度，

不能正确输出路径。

8.6 获取磁盘驱动器号函数: getdisk()

函数 getdisk() 用于获取当前工作目录的驱动器号, 其原型如下。

```
int getdisk(void)
```

【参数】 该函数没有参数。

【返回值】 返回值为 0 表示驱动器A; 为 1 表示驱动器B; 为 2 表示驱动器C, 以此类推。

【例 8-6】 下面的示例演示了 getdisk() 函数的使用, 代码如下。

```
#include<stdio.h>                                //加入标准输入输出库
#include<dir.h>                                    //加入目标操作函数库
void main( ){
    int a=getdisk( );
    printf("the current disk is %d",a);
}
```

运行上述程序, 获取当前目录的驱动器号, 输出结果如下。

```
the current disk is 2
```

注意: 上述程序的运行结果与当前路径有关。

8.7 建立目录函数: mkdir()

函数 mkdir() 用于在当前目录下创建一个新的目录, 其原型如下。

```
int mkdir(const char *path)
```

【参数】 参数 path 表示创建的新目录的名称, 最大有效长度为 8 个字符。

【返回值】 返回值为 0 表示创建新目录成功；为-1 表示创建失败。

【例 8-7】 下面的示例演示了 `mkdir()` 函数的使用，代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <dir.h>              //加入目标操作函数库
void main( ){
    int status;               //定义整型变量，存放函数结果
    status=mkdir("abc");      //创建新目录，名称为 ABC
    (!status)?(printf("Directory created")):(printf("Unable to
create directory"));
    //创建成功，输出 Directory created，否则输出 Unable to create directory
}
```

运行上述程序，在当前目录下创建名称为 ABC 的新目录，输出结果如下。

```
Directory created
```

注意：上述程序创建的新目录存放在当前目录下；如果当前目录下存在同名的目录名称，则函数返回值为-1，表示创建失败；创建的目录名称都是大写字母，`path` 参数的值如为小写，在目录创建时会自动转换为相应的大写字母；如果 `path` 参数字符超过 8 个，则有效字符数只能为 8 位。

8.8 建立文件名函数：mktemp()

函数 `mktemp()` 用于创建唯一文件名，其原型如下。

```
char mktemp(char *fname)
```

【参数】 参数 `fname` 表示产生的文件名称，输入时，该指针必须先赋值为 6 个 X 的字符串。

【返回值】 该函数返回创建的唯一文件名。

【例 8-8】 下面的示例演示了 `mktemp()` 函数的使用，代码如下。

```
#include <stdio.h>           //加入标准输入输出库
```

```

#include <dir.h>                                //加入目标操作函数库
void main( ){
    char *fname="TXXXXXX",*ptr;                //定义字符型指针变量
    ptr=mktemp(fname);                          //执行 mktemp( ) 函
数, 创建唯一文件名
    printf("the temp fname is %s",ptr);          //输出创建的文件名
}

```

运行上述程序, 创建一个唯一文件名, 输出结果如下。

```
the temp fname is TAA.AAA
```

注意: 上述程序创建的文件名为大写英文字母。

8.9 删除文件目录函数: rmdir()

函数 rmdir() 用于在当前目录下删除已存在的目录, 其原型如下。

```
int rmdir(const char *path)
```

【参数】 参数 path 表示删除目录的名称。

【返回值】 返回值为 0 表示删除目录成功; 为 -1 表示删除失败。

【例 8-9】 下面的示例演示了 rmdir() 函数的使用, 代码如下。

```

#include <stdio.h>                                //加入标准输入输出库
#include <dir.h>                                    //加入目标操作函数库
void main( ){
    int status;                                    //定义整型变量, 存放函数结果
    status=rmdir("abc");                          //删除目录, 名称为 abc
    (!status)?(printf("Directory deleted")):(printf("Unable to
deleted directory"));
    //删除成功, 输出 Directory deleted; 否则, 输出 Unable to deleted directory
}

```

运行上述程序, 删除当前目录中名称为 abc 的目录, 输出结果如下。

```
Directory deleted
```

注意：上述程序中，当前目录下存在名称为 abc 的目录，故运行结果如上。在删除目录时不区别英文字母的大小写。

8.10 设置当前工作驱动器函数：setdisk()

函数 setdisk() 实现当前工作驱动器的设置，其原型如下。

```
int setdisk(int drive)
```

【参数】 参数 drive 表示设置的驱动器号，取值为 0 代表 A 盘；值为 1 代表 B 盘，以此类推。

【返回值】 该函数返回驱动器总数。

【例 8-10】 下面的示例演示了 setdisk() 函数的使用，代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <dir.h>              //加入目标操作函数库
void main() {
    int currentDisk, disks, i; //currentDisk 代表当前使用的驱动器
    //号, disks 表示计算机支持的驱动器总数
    currentDisk = getdisk();    //获取当前工作的驱动器号
    disks = setdisk(currentDisk); //改变当前工作的驱动器号
    printf("%d logical drives on the system\n", disks);
    //输出系统支持的驱动器总数
    printf("Available drives:\n");
    for (i=0; i<26; i++) {
        setdisk(i);
        if (i==getdisk()) //判断系统支持的驱动器是否为当前驱动器
            printf("%c: drive is available\n", i+'a');
        //将数字驱动器号按字母表示的驱动器号输出
    }
    setdisk(currentDisk);
}
```

运行上述程序，输出结果如下。


```

26 logical drives on the system
Available drives:
c: drive is available
d: drive is available
e: drive is available
f: drive is available

```

注意: 上述程序的运行结果与当前工作目录、计算机硬盘驱动器的数目有关。一旦使用 `setdisk()` 函数改变当前工作驱动器号, 再编译其他 C 语言源程序时会出错, 所以请重启编译环境。

8.11 查找文件函数: `searchpath()`

函数 `searchpath()` 用于在当前工作目录下查找指定文件, 其原型如下。

```
char *searchpath(const char *file)
```

【参数】 参数 `file` 表示要寻找其路径的文件名称。

【返回值】 如果查找到指定的文件, 返回文件的路径; 否则, 返回 `NULL`。

【例 8-11】 下面的示例演示了 `searchpath()` 函数的使用, 代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <dir.h>             //加入目标操作函数库
void main( ){
    int disks=setdisk(4);    //改变当前驱动器为 E
    char *p=searchpath("a.c"); //查找 A.C 文件
    if (p!=NULL)             //判断文件是否存在
        printf("file path is %s",p); //输出文件路径
    else printf("file is nonexit!");
}

```

运行上述程序, 输出结果如下。

```
file path is E:\A.C
```

注意：上述程序的运行结果与计算机 E 盘上的文件结构有关。

8.12 构造文件名函数：fnmerge()

函数 fnmerge() 按指定的驱动器、目录名称、文件名称和扩展名称来构造文件名，其原型如下。

```
void fnmerge(char *path, char *drive, char *dir, char *fname, char *ext)
```

【参数】 参数 path 表示存储指定选项构造的文件的完整路径；参数 drive 表示驱动器名；参数 dir 表示目录名；参数 fname 表示文件名；参数 ext 表示扩展名。

【返回值】 该参数没有返回值。

【例 8-12】 下面的示例演示了 fnmerge() 函数的使用，代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
#include <dir.h>                                    //加入目标操作函数库
void main( ){
    char *s;                                       //字符型指针文件
    fnmerge(s, "e", "\\aa", "c", ".txt");        //构造文件: e:\aa\c.txt
    printf("%s\n", s);
}
```

运行上述程序，输出结果如下。

```
e:\aa\c.txt
```

第9章 系统接口函数库: dos.h

在程序设计中,常常需要进行针对硬件的相应操作。C语言专门提供了一个针对系统接口操作的函数库——dos.h,该文件库中存在的常用函数主要包括硬件读取、存储设置、中断设置等操作函数。

9.1 磁盘读数据函数: absread()

函数 absread()用于绝对磁盘扇区读取,其原型如下。

```
int absread(int drive, int nsects, int sectno, void *buffer);
```

【参数】参数drive为需要读取的磁盘;参数nsects为扇区号;参数sectno为逻辑扇区号;参数buffer为暂存数据的缓冲区。

【返回值】该函数执行成功后,将返回非零值;否则,返回NULL。

【例9-1】下面的示例演示了absread()函数的使用,采用该函数直接读取绝对磁盘扇区,代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <conio.h> //加入控制台输入输出库
#include <process.h> //加入进程管理库
#include <dos.h> //加入系统接口库

void main( ){
    int i, strt, ch_out, sector; char buf[512];
    printf("Insert a diskette into drive A and press any key\n");
    //输出提示信息

    sector = 0;
    if (absread(0, 1, sector, &buf) != 0) { //读取扇区数据
        perror("Disk problem");
        exit(1);
    }
```



```

printf("Read OK\n");           //输出提示信息
strt = 3;
for (i=0; i<80; i++) {         //输出所取的数据
    ch_out = buf[strt+i];
    putchar(ch_out);
}
printf("\n");
return(0);
}

```

运行上述程序，首先输出提示信息，读取指定扇区的数据到缓冲区，若读取失败输出提示信息；若读取成功，将缓冲区的数据显示到控制台。

9.2 磁盘写数据函数：abswrite()

函数 abswrite() 用于向磁盘指定扇区写数据，其原型如下。

```
int abswrite(int drive, int nsects, int tsectno, void *buffer);
```

【参数】 参数 drive 为磁盘号；参数 nsects 为扇区号；参数 tsectno 为逻辑扇区号；参数 buffer 为暂存要写入数据的缓冲区。

【返回值】 该函数执行成功后，将返回 0；否则，返回非零值。

【例 9-2】 下面示例演示了 abswrite() 函数的使用，使用该函数向指定磁盘特定扇区写入数据，代码如下。

```

#include <stdio.h>               //加入标准输入输出库
#include <dos.h>                 //加入系统接口库
#include <stdlib.h>              //加入工具库
#include <bios.h>               //加入 bios 相关函数库

void main( ){
    unsigned char buffer[512];  //声明变量
    clrscr( );                 //清屏
    printf("Insert a blank disk in the A: drive , then press any
key.\n");                      //输出提示信息
}

```

```

    getch( );                                //等待输入
    if(abswrite(0,1,1,&buffer) != 0)          //写入数据,并输出提示信息
        printf("Cannot write to the A drive.\n");
    else    printf("Drive A , Sector 1 written to\n");
}

```

运行上述代码,首先声明要写入的数据,然后输出提示信息等待输入,写入缓冲区数据,并根据返回数据输出相应的提示信息。

9.3 DOS 分配存储段函数: allocmem()

函数 allocmem()用于分配 DOS 存储段,其原型如下。

```
int allocmem(unsigned size, unsigned *seg);
```

【参数】 参数size为要分配的大小; 参数seg为返回的存储段位置。

【返回值】 该函数分配成功后,返回-1; 否则,返回0。

【例 9-3】 下面的示例演示了allocmem()函数的使用,采用该函数分配DOS存储段,代码如下。

```

#include <dos.h>                                //加入系统接口库
#include <alloc.h>                              //加入动态内存管理库
#include <stdio.h>                              //加入标准输入输出库
void main( ){
    unsigned int size, segp;
    int stat;
    size = 64;                                //分配大小
    stat = allocmem(size, &sepg);              //分配内存存储段
    if (stat == -1)                            //根据返回输出提示信息
        printf("Allocated memory at segment: %x\n", segp);
    else printf("Failed: maximum number of paragraphs available is
    %u\n",stat);
}

```

运行上述代码,首先定义要分配的存储段大小,然后分配存储段,并根据返回值输出相应的提示信息和所分配的存储段的位置。

9.4 DOS 系统调用函数: bdos()

函数 bdos() 用于 DOS 功能系统调用, 其原型如下。

```
int bdos(int dosfun, unsigned dosdx, unsigned dosal);
```

【参数】 参数 dosfun 是系统调用号; 参数 dosdx 是传给寄存器 DX 的值; 参数 dosal 是传给寄存器 AL 的值。

【返回值】 当该函数调用成功后, 返回 AX 中的值。

【例 9-4】 下面的示例演示了 bdos() 函数的使用, 采用该函数调用系统调用号, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
#include <dos.h>                                    //加入系统接口库
char current_drive(void) {                          //取当前驱动器
    char curdrive;                                  //声明变量
    curdrive = bdos(0x19, 0, 0);                    //系统调用
    return('A' + curdrive);
}

void main( ){
    printf("The current drive is %c:\n", current_drive());
    //输出当前驱动器
}
```

运行上述程序, 首先自定义函数, 通过调用 DOS 系统调用号的方式取当前驱动器号, 然后在主程序中调用该自定义函数, 并输出当前驱动器号。

9.5 返回国家相关信息函数: country()

函数 country() 返回指定国家代码的国家相关信息, 其原型如下。

```
struct COUNTRY *country(int countrycode, struct country *country);
```

该函数返回指定国家代码的国家相关信息, 其原型如下。

【参数】参数countrycode为国家代码; 参数country为返回的国家信息结构体。

【返回值】该函数没有返回值。

【例 9-5】下面的示例演示了country()函数的使用, 采用该函数返回国家相关信息, 代码如下。

```
#include <dos.h> //加入系统接口库
#include <stdio.h> //加入标准输入输出库
#define USA 0 //定义国家代码

void main() {
    struct COUNTRY country_info; //定义国家信息结构体
    country(USA, &country_info); //取国家信息
    printf("The currency symbol for the USA is: %s\n", country_info.
co_curr); //输出国家信息
}
```

运行上述程序, 首先声明国家信息代码, 然后将国家信息从国家信息结构体中取出, 最后将所取的国家信息输出至控制台。

9.6 设置 Ctrl_Break 处理程序函数: ctrlbrk()

函数 ctrlbrk()用于设置同时按下【Ctrl】和【Break】键的处理程序, 其原型如下。

```
void ctrlbrk(*fptr)(void);
```

【参数】参数fptr为所设置的处理程序名称。

【返回值】该函数没有返回值。

【例 9-6】下面的示例演示了ctrlbrk()函数的使用, 采用该函数设置Ctrl_Break处理程序, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <dos.h> //加入系统接口库
#define ABORT 0
```

```

int c_break(void) { //自定义处理函数
    printf("Control-Break pressed. Program aborting ...\n");
    //输出提示信息

    return (ABORT);
}

void main( ){
    ctrlbrk(c_break); //设置处理程序
    for(;;) { //循环等待中止
        printf("Looping... Press <Ctrl-Break> to quit:\n");
        //输出提示信息
    }
}

```

运行上述程序，首先声明用于设置自定义处理程序，然后在主程序中设置 Ctrl_Break 处理程序，输出提示信息，当同时按下【Ctrl】+【Break】键时，将执行所设置的处理程序，程序结束运行。

9.7 获取扩展 DOS 错误信息函数: dosexterr()

函数 dosexterr() 用于获取扩展 DOS 错误信息，其原型如下。

```
int dosexterr(struct DOSERR *dblkp);
```

【参数】 参数 dblkp 为保存错误信息的结构体。

【返回值】 该函数获取错误信息后，返回非零值；否则，返回 0。

【例 9-7】 下面的示例演示了 dosexterr() 函数的使用，采用该函数在程序中获取扩展 DOS 错误信息，代码如下。

```

#include <stdio.h> //加入标准输入输出库
#include <dos.h> //加入系统接口库

void main( ){
    FILE *fp;
    struct DOSERROR info; //声明结构体
    fp = fopen("perror.dat","r"); //打开文件
}

```

```

if (!fp) perror("Unable to open file for reading");
//根据打开结果输出提示信息
dosexterr(&info);
//获取扩展 DOS 错误信息
printf("Extended DOS error \information:\n"); //输出错误信息
printf("    Extended error:  \d\n",info.exterror); //输出错误信息
printf("        Class:  \x\n",info.class); //输出错误信息
printf("        Action:  \x\n",info.action); //输出错误信息
printf("        Error Locus:  \x\n",info.locus); //输出错误信息
}

```

运行上述程序,首先声明用于暂存错误信息的结构体;然后打开一个指定的文件,根据打开结果输出提示信息;接着获取 DOS 扩展错误信息,并输出错误信息中的每一项。

9.8 获取 Control_break 设置函数: getcbrk()

函数 getcbrk()用于获取 Control_break 设置,其原型如下。

```
int getcbrk(void);
```

【参数】该函数没有参数。

【返回值】若设置了Control_break处理函数,返回0;否则,返回-1。

【例 9-8】下面的示例演示了getcbrk()函数的使用,采用该函数获取Control_break设置,代码如下。

```

#include <stdio.h> //加入标准输入输出库
#include <dos.h> //加入系统接口库
void main( )
{
    if (getcbrk( )) //获取设置函数,并输出提示信息
        printf("Cntrl-brk flag is on\n");
    else printf("Cntrl-brk flag is off\n");
}

```

运行上述程序,首先通过 getcbrk()函数判断是否设置 Control_break 处理函数,并根据该处理函数输出相应的提示信息。

9.9 获取 DOS 日期函数: getdate()

函数 getdate()用于获取 DOS 日期, 其原型如下。

```
void getdate(struct *dateblk);
```

【参数】 参数dateblk为保存日期的日期结构体。

【返回值】 该函数所获取的日期暂存于参数dateblk中, 返回日期结构体。

【例 9-9】 下面的示例演示了getdate()函数的使用, 采用该函数获取 DOS日期, 代码如下。

```
#include <dos.h> //加入系统接口库
#include <stdio.h> //加入标准输入输出库
void main( ){
    struct date d; //声明日期结构体
    getdate(&d); //获取 DOS 日期
    printf("The current year is: %d\n",
        d.da_year); //输出日期项
    printf("The current day is: %d\n",
        d.da_day); //输出日期项
    printf("The current month is: %d\n",
        d.da_mon); //输出日期项
}
```

运行上述程序, 首先声明用于保存 DOS 日期的日期结构体; 然后获取 DOS 日期, 再分别输出 DOS 日期中的年、月、日。

9.10 设置 DOS 日期函数: setdate()

函数 setdate()用于设置当前系统的 DOS 日期, 其原型如下。

```
void setdate(struct date *dateblk);
```

【参数】 参数dateblk为要设置的DOS日期。

【返回值】该函数没有返回值。

【例 9-10】下面的示例演示了setdate()函数的使用，在程序中动态地设置DOS日期，代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <process.h> //加入进程管理库
#include <dos.h> //加入系统接口库
void main( ){
    struct date reset; //声明结构体
    struct date save_date; //声明日期结构体
    getdate(&save_date); //取 DOS 日期
    printf("Original date:\n"); //输出日期
    system("date"); //显示日期
    reset.da_year = 2001; //给日期结构体赋值
    reset.da_day = 1; //给日期结构体赋值
    reset.da_mon = 1; //给日期结构体赋值
    setdate(&reset); //设置 DOS 日期
    printf("Date after setting:\n"); //输出日期
    system("date");
    setdate(&save_date); //设置原始日期
    printf("Back to original date:\n"); //输出日期
    system("date");
}
```

运行上述程序，首先声明用于暂存当前日期和设置日期的日期结构体，获取并输出当前 DOS 日期；然后设置并输出自定义的日期；最后还原并输出原来的日期。

9.11 设置 DOS 时间函数: settime()

函数 settime()用于设置 DOS 时间，其原型如下。

```
void settime(struct time *timep);
```

【参数】参数timep为时间结构体。

【返回值】该函数没有返回值。

【例 9-11】下面的示例演示了 `settime()` 函数的使用, 使用该函数设置 DOS 时间, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <dos.h>             //加入系统接口库
void main( ){
    struct time t;           //声明结构体
    gettime(&t);             //取当前 DOS 时间
    printf("The current minute is: %d\n", t.ti_min); //输出时间
    printf("The current hour is: %d\n", t.ti_hour); //输出时间
    printf("The current hundredth of a second is: %d\n", t.ti_hund); //输出时间
    printf("The current second is: %d\n", t.ti_sec); //输出时间
    t.ti_min++;              //当前时间分钟加 1
    settime(&t);             //设置时间
}
```

运行上述代码, 首先声明用于暂存时间的时间结构体, 获取并输出当前时间; 然后将当前时间的分钟加 1, 再以该时间设置为 DOS 时间。

9.12 从硬件端口中输入函数: `inport()`

函数 `inport()` 用于从硬件端口中输入数据, 其原型如下。

```
int inport(int portid);
```

【参数】参数 `portid` 为硬件端口号。

【返回值】该函数返回从硬件端口读取的数据。

【例 9-12】下面的示例演示了 `inport()` 函数的使用, 使用该函数读取指定硬件端口的数据, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <dos.h>             //加入系统接口库
```



```

void main( ){
    int result;                //声明变量
    int port = 0;              //声明端口号变量
    result = inport(port);     //读取端口内容
    printf("Word read from port %d = 0x%X\n", port, result);
                                //输出内容
}

```

运行上述代码，首先定义用于暂存端口内容的变量和端口号变量；然后从指定硬件端口读取数据，再将其以十六进制输出到控制台。

9.13 从硬件端口中输入函数: inportb()

函数 inportb() 从硬件端口中输入一个字节的的数据，其原型如下。

```
int inportb(int port);
```

【参数】 参数 port 表示要硬件端口号。

【返回值】 该函数返回从指定硬件端口读入的一个 8 位二进制（一个字节）数据。

【例 9-13】 下面的示例演示了 inportb() 函数的使用，从硬件端口中读取一个字节的输入数据，代码如下。

```

#include <stdio.h>                //加入标准输入输出库
#include <dos.h>                  //加入系统接口库
void main( ){
    int result;                    //声明暂存读取内容变量
    int port = 0x210;              //声明端口号
    result = inportb(port);        //读取一字节数据
    printf("port 0x%x sent Value is%d\n ", port, result); //输出
}

```

运行上述程序，首先声明用于暂存读取内容的变量和端口号变量；然后从指定的端口号读取一字节数据到指定变量中；最后再将所读取的内容输出到控制台。

9.14 从硬件端口中输出函数: output()

函数 output() 用于从硬件端口中输出数据, 其原型如下。

```
void output(int port, int value);
```

【参数】 参数 port 为要输出的硬件端口号; 参数 value 为要输出的数据。

【返回值】 该函数没有返回值。

【例 9-14】 下面的示例演示了 output() 函数的使用, 在程序中向指定硬件端口中输出数据, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
#include <dos.h>                                    //加入系统接口库
int main(void)
{
    int value = 64;                                //声明需要输出数据
    int port = 0;                                  //声明硬件端口
    output(port, value);                           //向指定端口输出数据
    printf("Value %d sent to port number %d\n", value, port); //输出提示信息
    return 0;
}
```

运行上述程序, 首先声明要输出的数据和输出的硬件端口号; 然后再将该数据输出到指定的端口号; 最后输出提示信息。

9.15 从硬件端口中输出函数: outportb()

函数 outportb() 用于向硬件端口输出一字节的数据, 其原型如下。

```
void outportb(int port, char byte);
```

【参数】 参数 port 为硬件端口号; 参数 byte 为要输出的字节内容。

【返回值】该函数没有返回值。

【例 9-15】下面的示例演示了outportb()函数的使用, 采用该函数在程序中向指定硬件端口中输出字节数据, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <dos.h>             //加入系统接口库

void main( ){
    int value = 64;          //声明输出数据
    int port = 0;            //声明端口号
    outportb(port, value);   //输出字节数据
    printf("Value %d sent to port number %d\n", value, port);
                               //输出提示信息
}
```

运行上述程序, 首先声明需要输出的数据和硬件端口号; 然后将指定的数据输出到特定的硬件端口, 再将输出提示信息。

9.16 通用 DOS 接口函数: intdos()

函数 intdos()用于调用通用 DOS 接口, 其原型如下。

```
int intdos(union REGS *inregs, union REGS *outregs);
```

【参数】用户定义的寄存器值存于寄存器结构 inregs 中, 运行完后参数将返回的寄存器值存于寄存器结构 outregs 中。

【返回值】若该函数执行成功, 返回 0; 否则, 返回非零值。

【例 9-16】下面的示例演示了intdos()函数的使用, 采用该函数调用通用DOS接口, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <dos.h>             //加入系统接口库

int delete_file(char near *filename) { //自定义函数删除文件
    union REGS regs;
    int ret;
    regs.h.ah = 0x41;          //调用中断删除文件
```



```

regs.x.dx = (unsigned) filename; //要删除的文件名
ret = intdos(&regs, &regs);      //调用 DOS 接口
return(regs.x.cflag ? ret : 0);  //删除成功返回 0, 失败返回非零值
}

void main( ){
int err;
err = delete_file("NOTEXIST.$$$"); //调用自定义函数删除指定文件
if (!err)                          //根据删除结果输出提示信息
    printf("Able to delete NOTEXIST.$$$\n");
else printf("Not Able to delete NOTEXIST.$$$\n");
}

```

运行上述程序, 首先声明采用通用 DOS 接口实现删除指定文件的自定义函数; 然后调用该函数删除指定文件, 并根据返回结果输出相应的提示信息。

9.17 通用 DOS 中断接口函数: intdosx()

函数 intdosx() 用于调用通用 DOS 中断接口, 其原型如下。

```
int intdosx(union REGS *inregs, union REGS *outregs, struct SREGS *segregs);
```

【参数】 用户定义的寄存器值存于寄存器结构 inregs 和 segregs 中; 运行完后将返回的寄存器值保存于寄存器结构 outregs 中。

【返回值】 若该函数执行成功, 返回 0; 否则, 返回非零值。

【例 9-17】 下面的示例演示了 isdosx() 函数的使用, 采用该函数调用通用 DOS 中断接口, 代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <dos.h>             //加入系统接口库
int delete_file(char far *filename) { //自定义函数删除文件
    union REGS regs; struct SREGS sregs;
    int ret;
    regs.h.ah = 0x41;        //调用中断删除文件
}

```

```

regs.x.dx = FP_OFF(filename);           //文件名
sregs.ds = FP_SEG(filename);           //调用时的段寄存器值
ret = intdosx(&regs, &regs, &sregs); //调用 DOS 中断删除文件
return(regs.x.cflag ? ret : 0);         //若删除成功返回0; 否则返回非零值
)

void main( ){
int err;
err = delete_file("NOTEXIST.$$$"); //调用自定义函数删除文件
if (!err)                          //根据返回结果输出相应提示信息
    printf("Able to delete NOTEXIST.$$$\n");
else printf("Not Able to delete NOTEXIST.$$$\n");
}

```

运行上述程序, 首先声明采用调用 DOS 中断接口实现删除指定文件的自定义函数; 然后调用该函数删除指定文件, 并根据返回结果输出相应的提示信息。

9.18 通用 8086 软中断接口函数: int86()

函数 int86() 用于调用通用 8086 软中断接口, 其原型如下。

```
int int86(int intr_num, union REGS *inregs, union REGS *outregs);
```

【参数】 参数 intr_num 指定中断号; 参数 inregs 与参数 outregs 分别是入口参数寄存器和出口参数寄存器。

【返回值】 该函数返回中断结束的返回值, 并暂存于 outregs 中。

【例 9-18】 下面的示例演示了 int86() 函数的使用, 采用该函数在程序中调用 8086 软中断, 代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <conio.h>           //加入控制台输入输出库
#include <dos.h>             //加入系统接口库
#define VIDEO 0x10
void movetoxy(int x, int y) { //自定义函数
    union REGS regs;         //声明结果体

```

```

    regs.h.ah = 2;                //设置光标位置
    regs.h.dh = y;
    regs.h.dl = x;
    regs.h.bh = 0;                //显示页号
    int86(VIDEO, &regs, &regs);  //调用显示中断
}

void main( ){
    movetoxy(35, 10);             //调用自定义函数
    printf("Hello\n");           //输出
}

```

运行上述程序，首先声明调用 8086 软中断的自定义函数；然后清除屏幕，调用该自定义函数，再在屏幕指定位置输出提示信息。

9.19 通用 8086 软中断接口函数：int86x()

函数 int86x() 用于调用通用 8086 软中断接口，其原型如下。

```
int int86x(int intr_num, union REGS *inregs, union REGS *outregs,
struct SREGS *segregs);
```

【参数】参数 intr_num 指定中断号；参数 inregs 与参数 outregs 分别是入口参数寄存器和出口参数寄存器；参数 segregs 为调用时的段寄存器的值和设置返回时的值。

【返回值】函数执行成功后，将返回的寄存器值暂存于存储器结构 outregs 中。

【例 9-19】下面的示例演示了 int86x() 函数的使用，采用该函数调用 8086 软中断接口，代码如下。

```

#include <dos.h>                //加入系统接口库
#include <process.h>             //加入进程管理库
#include <stdio.h>               //加入标准输入输出库
void main( ){
    char filename[80];           //声明字符数组
    union REGS inregs, outregs;
}

```



```

struct SREGS segregs;
printf("Enter filename: ");
gets(filename); //获取文件名
inregs.h.ah = 0x43;
inregs.h.al = 0x21;
inregs.x.dx = FP_OFF(filename);
segregs.ds = FP_SEG(filename);
int86x(0x21, &inregs, &outregs, &segregs); //调用 21 中断
printf("File attribute: %X\n", outregs.x.cx); //输出信息
}

```

运行上述程序, 首先声明用于暂存文件名的字符数组; 然后获取输入的文件名, 调用通用 8086 软中断; 最后再输出调用结果。

9.20 改变软中断接口函数: intr()

函数 intr() 用于改变软中断接口, 其原型如下。

```
void intr(int intr_num, struct REGPACK *preg);
```

【参数】 参数 intr_num 为中断号; 参数 preg 为结构变量名。

【返回值】 该函数没有返回值。

【例 9-20】 下面的示例演示了 intr() 函数的使用, 使用该函数改变软中断接口, 其代码如下。

```

#include <stdio.h> //加入标准输入输出库
#include <string.h> //加入字符串库
#include <dir.h> //加入目录管理库
#include <dos.h> //加入系统接口库
#define CF 1 /* Carry flag */
void main() {
    char directory[80];
    struct REGPACK reg;
    printf("Enter directory to change to: "); //输出提示信息
    gets(directory); //获取目录
}

```

```

    reg.r_ax = 0x3B << 8;                //赋值
    reg.r_dx = FP_OFF(directory);        //赋值
    reg.r_ds = FP_SEG(directory);        //赋值
    intr(0x21, &reg);                    //改变软中断接口
    if (reg.r_flags & CF)                 //根据返回结果输出提示信息
        printf("Directory change failed\n");
    getcwd(directory, 80);               //获取当前工作目录
    printf("The current directory is: %s\n", directory);
                                        //输出当前目录
}

```

运行上述代码，首先声明用于暂存目录的字符数组和结构体；然后获取要改变的目录，改变软中断接口，并根据返回结果输出提示信息；最后获取并输出当前工作目录。

9.21 退出并继续驻留函数：keep()

函数 keep() 用于退出并继续驻留，其原型如下。

```
void keep(int status, int size);
```

【参数】 参数 status 为出口状态；参数 size 为程序占用存储空间。

【返回值】 该函数没有返回值。

【例 9-21】 下面的示例演示了 keep() 函数的使用，在程序中设置为内存常驻程序，代码如下。

```

#include <dos.h>                        //加入系统接口库
#define INTR 0x1C                       //声明中断
#define ATTR 0x7900                    //声明屏幕属性
extern unsigned _heaplen = 1024;
extern unsigned _stklen = 512;
void interrupt ( *oldhandler)(void);    //声明中断函数
void interrupt handler(void) {
    unsigned int (far *screen)[80];
    static int count;
}

```

```

    screen = MK_FP(0xB800, 0);
    count++; //计数器增加, 并保持在0~9
    count %= 10;
    screen[0][79] = count + '0' + ATTR; //将数字输出
    oldhandler(); //调用旧中断
}

void main() {
    oldhandler = getvect(INTR); //取中断地址
    setvect(INTR, handler); //设置新中断函数
    keep(0, (_SS + (_SP/16) - _psp)); //退出并继续驻留
}

```

运行上述代码, 首先声明中断处理函数; 然后取中断地址, 设置新的中断函数, 将当前程序常驻内存, 待执行自定义中断处理函数后, 调用原中断。

9.22 分析文件名函数: parsfnm()

函数 `parsfnm()` 用于分析文件名, 其原型如下。

```
char *parsfnm (char *cmdline, struct fcb *fcbptr, int option);
```

【参数】 参数 `cmdline` 为要分析的文件名; 参数 `fcbptr` 为文件控制块结构体; 参数 `option` 为分析文件名的参数。

【返回值】 若该函数执行成功, 则返回文件控制块结构; 否则, 返回 `NULL`。

【例 9-22】 下面的示例演示了 `parsfnm()` 函数的使用, 在程序中采用该函数分析文件名, 代码如下。

```

#include <process.h> //加入进程管理库
#include <string.h> //加入字符串库
#include <stdio.h> //加入标准输入输出库
#include <dos.h> //加入系统接口库

void main() {
    char line[80];
}

```



```

    struct fcb blk;
    printf("Enter drive and file name (no path - ie. a:file.dat)\n");
                                                //取文件名
    gets(line);
    if (parsfnm(line, &blk, 1) == NULL)      //将文件名放到文件控制块
中, 并根据返回结果输出提示信息
        printf("Error in parsfm call\n");
    else printf("Drive #%d Name: %11s\n", blk.fcb_drive, blk.fcb_
name);
                                                //输出文件控制块内容
}

```

运行上述程序, 首先声明用于暂存文件名的字符数组和文件控制块结构体; 然后输出提示信息, 等待用户输入文件名, 待用户输入用户名后, 分析所输入的文件名, 并根据分析结果输出提示信息。

9.23 检查存储单元函数: peek()

函数 `peek()` 用于检查存储单元, 其原型如下。

```
int peek(int segment, unsigned offset);
```

【参数】 参数 `segment` 为要检查的存储单元地址; 参数 `offset` 为地址偏移量。

【返回值】 该函数返回指定的要检查单元的值。

【例 9-23】 下面的示例演示了 `peek()` 函数的使用, 采用该函数检查指定的存储单元, 代码如下。

```

#include <stdio.h>                                //加入标准输入输出库
#include <conio.h>                                //加入控制台输入输出库
#include <dos.h>                                  //加入系统接口库
void main( ){
    int value = 0;
    printf("The current status of your keyboard is:\n");
                                                //输出提示信息
    value = peek(0x0040, 0x0017);                //检查存储单元
}

```

```

if (value & 1)          //判断存储单元的值, 并根据存储结果输出提示信息
    printf("Right shift on\n");
else    printf("Right shift off\n");
if (value & 2)          printf("Left shift on\n");
else    printf("Left shift off\n");
if (value & 4)          printf("Control key on\n");
else    printf("Control key off\n");
if (value & 8)          printf("Alt key on\n");
else    printf("Alt key off\n");
}

```

运行上述程序, 检查指定存储单元的值, 并根据该存储单元的值判断当前键盘状态, 判断键盘哪一功能键正在使用。

9.24 检查存储单元函数: peekb()

函数 peekb() 用于检查存储单元, 其原型如下。

```
char peekb (int segment, unsigned offset);
```

【参数】 参数 segment 为要检查的存储单元地址; 参数 offset 为存储单元偏移量。

【返回值】 该函数返回指定存储单元一字节的数据。

【例 9-24】 下面的示例演示了 peekb() 函数的使用, 采用该函数检查指定的存储单元, 代码如下。

```

#include <stdio.h>                //加入标准输入输出库
#include <conio.h>                //加入控制台输入输出库
#include <dos.h>                  //加入系统接口库
void main( ){
    int value = 0;
    printf("The current status of your keyboard is:\n");
                                //输出提示信息
    value = peekb(0x0040, 0x0017); //检查储存单元, 并根据结果输出提示信息
    if (value & 1)    printf("Right shift on\n");
}

```

```

else    printf("Right shift off\n");
if (value & 2)    printf("Left shift on\n");
else    printf("Left shift off\n");
if (value & 4)    printf("Control key on\n");
else    printf("Control key off\n");
if (value & 8)    printf("Alt key on\n");
else    printf("Alt key off\n");
}

```

运行上述程序,首先输出提示信息;然后读取指定存储单元的值,并根据该值判断当前键盘状态。

9.25 存值到给定存储单元函数: poke()

函数 `poke()` 用于存值到指定的存储单元,其原型如下。

```
void poke(int segment, int offset, int value);
```

【参数】 参数 `segment` 为指定的存储单元; 参数 `offset` 为存储器的偏移量; 参数 `value` 为要存储的值。

【返回值】 该函数没有返回值。

【例 9-25】 下面的示例演示了 `poke()` 函数的使用,在程序中将给定的值存储到指定存储单元,代码如下。

```

#include <dos.h>                //加入系统接口库
#include <conio.h>              //加入控制台输入输出库
void main( ){
    clrscr();
    cprintf("Make sure the scroll lock key is off and press any
key\r\n");                      //输出提示信息
    getch( );                  //等待输入
    poke(0x0000,0x0417,16);    //存储给定值到指定单元
    cprintf("The scroll lock is now on\r\n"); //输出提示信息
}

```

运行上述程序,首先清除屏幕,输出提示信息,等待输入任意字

符;然后再将给定的数据存储到指定的存储单元中,再输出提示信息。

9.26 存值到给定存储单元函数: pokeb()

函数 `pokeb()` 用于向指定存储单元存储一字节数据,其原型如下。

```
void pokeb(int segment, int offset, char value);
```

【参数】 参数 `segment` 为指定的存储单元; 参数 `offset` 为存储器的偏移量; 参数 `value` 为要存储的值。

【返回值】 该函数没有返回值。

【例 9-26】 下面的示例演示了 `pokeb()` 函数的使用, 在程序中向指定的存储单元存储一字节数据, 代码如下。

```
#include <dos.h> //加入系统接口库
#include <conio.h> //加入控制台输入输出库
void main( ){
    clrscr( );
    printf("Make sure the scroll lock key is off and press any
key\r\n"); //输出提示信息
    getch( ); //输入字符
    pokeb(0x0000,0x0417,16); //向存储单元写入值
    printf("The scroll lock is now on\r\n"); //输出提示信息
}
```

运行上述程序, 首先清除屏幕, 输出提示信息, 等待输入任意字符; 然后向指定的存储单元写入字节数据, 再在控制台输出提示信息。

9.27 随机块读函数: randbrd()

函数 `randbrd()` 用于实现随机块读, 采用 DOS 0x27 中断将文件内容读入磁盘缓冲区, 其原型如下。

```
int randbrd(struct fcb *fcbptr, int reccnt);
```

【参数】参数fcbptr为要读取的块；参数recnt为记录条数。

【返回值】该函数返回 0 表示读取所有记录；1 表示文件结束；2 表示循环读取；3 表示文件结束，但最后一条记录未完成。

【例 9-27】下面的示例演示了randbrd()函数的使用，在程序中采用该函数实现随机块读，代码如下。

```
#include <process.h>           //加入进程管理库
#include <string.h>             //加入字符串库
#include <stdio.h>              //加入标准输入输出库
#include <dos.h>                //加入系统接口库

void main( ){
    char far *save_dta;
    char line[80], buffer[256];
    struct fcb blk;
    int i, result;
    printf("Enter drive and file name (no path - i.e. a:file.dat)\n");
    //读取文件名
    gets(line);
    if (!parsfnm(line, &blk, 1)) { //分析文件名
        printf("Error in call to parsfnm\n");
        exit(1);
    }
    printf("Drive #%d File: %s\n\n", blk.fcb_drive, blk.fcb_name);
    //输出文件控制块内容
    bdosptr(0x0F, &blk, 0); //采用 DOS 文件控制块打开文件
    save_dta = getdta( );    //保存旧数据
    setdta(buffer);          //设置新数据
    blk.fcb_recsz = 128;     //设置新文件数据
    blk.fcb_random = 0L;     //设置新文件数据
    result = randbrd(&blk, 1); //随机块读
    if (!result)              //根据结果输出相应提示信息
        printf("Read OK\n\n");
    else {
        perror("Error during read");
        exit(1);
    }
}
```

```

    }
    printf("The first 128 characters are:\n"); //输出新数据信息
    for (i=0; i<128; i++)    putchar(buffer[i]);
    setdta(save_dta);        //恢复旧数据
}

```

运行上述程序，首先输出提示信息，等待用户输入文件，并以该文件名打开文件；然后保存旧的数据，设置新的数据，并随机读取指定块，再根据读取结果输出相应的提示信息；最后恢复旧数据。

9.28 随机块写函数: randbwr()

函数 randbwr()用于实现随机块写，其原型如下。

```
int randbwr(struct fcb *fcbptr, int reccnt);
```

【参数】 参数fcbptr为要写的块；参数recnt为记录条数。

【返回值】 该函数返回 0 表示写完所有记录；1 表示磁盘空间不足，未操作；2 表示写记录返回 0xFFFF。

【例 9-28】 下面的示例演示了randbwr()函数的使用，在程序中动态地更改内存数据段空间，代码如下。

```

#include <process.h>           //加入进程管理库
#include <string.h>            //加入字符串库
#include <stdio.h>             //加入标准输入输出库
#include <dos.h>               //加入系统接口库
void main( ){
    char far *save_dta;
    char line[80];
    char buffer[256] = "RANDBWR test!";
    struct fcb blk;
    int result;
    printf("Enter a file name to create (no path - ie. a:file.dat\n");
                                //输出提示信息
    gets(line);                 //获取输入的文件名
}

```



```

parsfnm(line, &blk, 1);           //分析文件名
printf("Drive #%d File: %s\n", blk.fcb_drive, blk.fcb_name);
                                   //输出文件控制块内容
if (bdosptr(0x16, &blk, 0) == -1) { //创建文件, 并输出提示信息
    perror("Error creating file");
    exit(1);
}
save_dta = getdta( );             //保存文件内容
setdta(buffer);                   //设置文件新内容
blk.fcb_recsz = 256;              //修改文件控制块内容
blk.fcb_random = 0L;              //修改文件控制块内容
result = randbwr(&blk, 1);        //随机块写
if (!result)                      //根据结果输出提示信息
    printf("Write OK\n");
else {
    perror("Disk error");
    exit(1);
}
if (bdosptr(0x10, &blk, 0) == -1) { //关闭文件
    perror("Error closing file");
    exit(1);
}
setdta(save_dta);                 //还原文件原内容
}

```

运行上述程序, 首先获取输入的文件名, 分析并输出文件控制块内容; 然后将文件原内容暂存, 修改文件控制块内容, 以随机块的方式写入方式; 最后关闭文件, 还原文件原始内容。

9.29 读段寄存器值函数: segread()

函数 segread() 用于读取寄存器值, 其原型如下。

```
void segread(struct SREGS *segtbl);
```

【参数】 参数 segtbl 为暂存寄存器值的结构体。

【返回值】该函数没有返回值。

【例 9-29】下面的示例演示了 `segread()` 函数的使用, 使用该函数获取寄存器中的值, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <dos.h> //加入系统接口库
void main() {
    struct SREGS segs; //声明寄存器值结构体
    segread(&segs); //读取寄存器值
    printf("Current segment register settings\n\n"); //输出寄存器值
    printf("CS: %X DS: %X\n", segs.cs, segs.ds);
    printf("ES: %X SS: %X\n", segs.es, segs.ss);
}
```

运行上述代码, 首先声明用于暂存寄存器值的结构体, 然后读取寄存器中的值, 最后输出每个寄存器中的值至控制台。

9.30 执行挂起函数: `sleep()`

函数 `sleep()` 用于执行挂起操作, 其原型如下。

```
unsigned sleep(unsigned seconds);
```

【参数】参数 `seconds` 为要挂起的时间。

【返回值】当已经过了参数 `seconds` 所指定的时间, 返回值为 0; 当由于捕捉到某个信号 `sleep` 提前返回时, 返回值是未睡够的时间。

【例 9-30】下面的示例演示了 `sleep()` 函数的使用, 采用该函数使程序挂起, 代码如下。

```
#include <dos.h> //加入系统接口库
#include <stdio.h> //加入标准输入输出库
void main() {
    int i; //声明变量
    for (i=1; i<5; i++) { //循环输出挂起, 并输出挂起时间
        printf("Sleeping for %d seconds\n", i);
    }
```

```

        sleep(i);
    }
}

```

运行上述代码，首先声明用于循环的变量；然后依次进行循环，将程序挂起并输出挂起时间，待循环完成结束程序。

9.31 修改 DOS 分配函数：setblock()

函数 setblock() 用于修改先前已分配的 DOS 存储段大小，其原型如下。

```
int setblock(int seg, int newsize);
```

【参数】 参数 seg 为要修改的存储段地址；参数 newsize 为要修改的大小。

【返回值】 该函数分配成功返回非零值；否则，返回 0。

【例 9-31】 下面的示例演示了 setblock() 函数的使用，在程序中修改分配的 DOS 存储段大小，代码如下。

```

#include <dos.h>           //加入系统接口库
#include <alloc.h>          //加入动态内存管理库
#include <stdio.h>          //加入标准输入输出库
#include <stdlib.h>         //加入标准工具库
void main( ){
    unsigned int size, segp;
    int stat;
    size = 64;              //分配的空间大小
    stat = allocmem(size, &sepg); //分配 DOS 存储段
    if (stat == -1)         //根据分配结果输出提示信息
        printf("Allocated memory at segment: %X\n", sepg);
    else {
        printf("Failed: maximum number of paragraphs available is
%d\n", stat);
        exit(1);
    }
}

```



```

    }
    stat = setblock(segp, size * 2);           //修改存储段大小
    if (stat == -1)                             //根据分配结果输出提示信息
        printf("Expanded memory block at segment: %X\n", segp);
    else printf("Failed: maximum number of paragraphs available is
%d\n", stat);
    freemem(segp);                             //释放分配的内存
}

```

运行上述程序, 首先分配 DOS 存储段, 并根据分配结果输出相应的提示信息。若分配成功, 修改 DOS 存储段分配大小, 并根据分配结果输出提示信息。

9.32 UNIX 时间格式转换函数: dostounix()

函数 dostounix() 用于实现 UNIX 时间格式的转换, 其原型如下。

```
long dostounix(struct date *dateptr, struct time *timeptr);
```

【参数】 参数 dateptr 为转换的日期结构体; 参数 timeptr 为时间结构体。

【返回值】 该函数执行后返回 UNIX 时间。

【例 9-32】 下面的示例演示了 dostounix() 函数的使用, 将当前系统日期时间转换为 UNIX 时间格式, 代码如下。

```

#include <time.h>           //加入时间函数库
#include <dos.h>             //加入系统接口库
#include <stdio.h>          //加入标准输入输出库

void main() {
    time_t t;
    struct time d_time;
    struct date d_date;
    struct tm *local;
    getdate(&d_date);       //取系统日期
    gettime(&d_time);       //取系统时间
    t = dostounix(&d_date, &d_time); //转换为 UNIX 时间
}

```

```

    local = localtime(&t);           //转换为结构体
    printf("Time and Date: %s\n", \
    asctime(local));                 //输出 ASCII 码时间
}

```

运行上述程序,首先声明暂存日期和时间的结构体;然后获取当前系统日期和时间,并将其转换为 UNIX 时间,并输出转换后的 UNIX 时间。

9.33 获取系统时间函数: `gettime()`

函数 `gettime()` 用于获取系统时间,其数原型如下。

```
void gettime(struct time *timep);
```

【参数】 参数 `timep` 为该函数获取的时间。

【返回值】 该函数没有返回值,其获取的时间保存于 `tm` 结构变量 `timep` 中。

【例 9-33】 下面的示例演示了 `gettime()` 函数的使用,采用该函数获取系统时间,代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <dos.h>              //加入系统接口库
void main( )
{
    struct time t;           //声明结构体
    gettime(&t);             //获取系统时间
    printf("The current time is: %2d:%02d:%02d.%02d\n", t.ti_hour,
    t.ti_min, t.ti_sec, t.ti_hund); //输出
}

```

运行上述程序,首先声明用于保存系统时间的 `tm` 结构体变量;然后通过该函数获取系统时间,再将暂存于该变量中的 `tm` 结构的系统时间输出。

第 10 章 输入输出函数库: io.h

在程序设计中,常常需要与外部文件进行交互操作。C 语言专门提供了一个针对输入输出操作的函数库——io.h,该文件库中存在的常用函数主要包括文件及文件流的读取、输出、写入等输入输出操作函数。

10.1 打开文件函数: open()

函数 open() 用于在程序中打开指定文件,其原型如下。

```
int open(char *pathname, int access[, int permiss]);
```

【参数】 参数 pathname 为要打开的包含路径的文件名; 参数 access 为该方式; 参数 permiss 为访问权限。

【返回值】 该函数执行成功后将返回该文件句柄; 否则, 将返回 -1。

【例 10-1】 下面的示例演示了 open() 函数的使用, 采用该函数在程序中打开指定的文件, 代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>             //加入标准输入输出库
#include <io.h>                //加入输入输出库
void main( ){
    int handle;                //定义句柄变量
    char msg[ ] = "Hello world"; //定义字符型的数组
    if ((handle = open("test.txt", O_CREAT | O_TEXT)) == -1) {
//打开指定文件
        perror("Error:");
        return 1;
    }
```



```

    write(handle, msg, strlen(msg));           //写入字符
    close(handle);                             //关闭文件
}

```

运行上述程序，首先定义一个整型类型的句柄变量；然后打开指定的文件。若打开失败，则输出提示信息；若成功打开文件，写入指定的内容，然后关闭所打开的文件。

10.2 关闭文件函数：close()

函数 close() 关闭由 open() 函数所打开的文件，其原型如下。

```
int close(int handle);
```

【参数】 参数 handle 为打开文件时所返回的文件句柄。

【返回值】 该函数成功关闭文件后将返回 0；否则，将返回 -1。

【例 10-2】 下面的示例演示了 close() 函数的使用，使用该函数关闭指定的文件，代码如下。

```

#include <string.h>           //加入字符串库
#include <stdio.h>            //加入标准输入输出库
#include <io.h>               //加入输入输出库
main() {
    int handle;               //定义文件句柄
    char buf[11] = "0123456789"; //定义字符数组
    handle = open("test.txt ", O_CREAT); //打开指定文件
    if (handle > -1) {        //判断打开成功否
        write(handle, buf, strlen(buf)); //写入指定字符
        close(handle);         //关闭文件
    } else {
        printf("Error opening file\n"); //输出失败提示信息
    }
}

```

运行上述代码，首先采用打开指定的文件，若打开文件成功，则

写入指定的内容; 若打开失败, 则输出提示信息。

【返回值】该函数创建文件后, 将返回所创建文件的句柄。

10.3 创建文件函数: creat()

函数 creat() 用于创建指定文件名的文件, 其原型如下。

```
int creat (const char *filename, int permiss);
```

【参数】参数 filename 表示文件名, 参数 permiss 表示操作权限。

【返回值】该函数创建文件后, 将返回所创建文件的句柄。

【例 10-3】下面的示例演示了 creat() 函数的使用, 使用该函数创建一个文件并写入内容, 代码如下。

```
#include <string.h>           //加入字符串库
#include <io.h>                //加入输入输出库
void main( ){
    int handle;                //定义句柄变量
    char buf[11] = "0123456789"; //定义字符数组
    _fmode = O_BINARY;        //文件使用模式
    handle = creat("test.txt ", S_IREAD | S_IWRITE);
                                //创建二进制文件, 并可以进行读写
    write(handle, buf, strlen(buf)); //写入内容
    close(handle);              //关闭文件
}
```

运行上述代码后, 首先设置创建文件的模式; 然后根据文件创建指定的文件, 再写入所定义的字符数组内容; 最后关闭创建的文件。

10.4 文件检测结束函数: eof()

函数 eof() 用于检测指定文件是否结束, 其原型如下。

```
int eof(int *handle);
```

【参数】 参数handle表示要检测的文件句柄。

【返回值】 该函数判断给定文件是否结束，若结束则返回 0；否则，返回-1。

【例 10-4】 下面的示例演示了eof()函数的使用，采用该函数判断一个文件是否结束并进行输出，代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
#include <io.h>               //加入输入输出库

void main( ){
    int handle;
    char msg[ ] = "This is a test";
    char ch;
    handle = open("test.txt ",
        O_CREAT | O_RDWR,
        S_IREAD | S_IWRITE);    //创建文件
    write(handle, msg, strlen(msg)); //写入内容
    lseek(handle, 0L, SEEK_SET); //移动读写指针
    do {
        read(handle, &ch, 1);    //读取文件内容
        printf("%c", ch);        //输出
    } while (!eof(handle));       //判断文件是否结束
    close(handle);               //关闭文件
}
```

运行上述程序，首先创建一个文件并写入指定的内容；然后判断文件是否结束，读取并输出其内容。

10.5 读文件函数：read()

函数 read() 用于读取由 open() 函数所打开文件的内容，其原型如下。

```
int read(int handle, void *buf, int nbyte);
```


【参数】参数handle表示要读取的文件;参数buf表示要将读取的内容保存的位置;参数nbyte表示要读取的长度。

【返回值】若该函数成功读取指定长度的文件内容,将返回 0;否则,返回-1。

【例 10-5】下面的示例演示了read()函数的使用,从给定的文件中读取指定的长度到缓冲区,代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <io.h>               //加入输入输出库
#include <alloc.h>            //加入动态内存管理函数库
void main( ){
    void *buf;
    int handle, bytes;
    buf = malloc(10);
    if ((handle =open("test.txt ", O_RDONLY | O_BINARY, S_IWRITE |
S_IREAD)) == -1)
    {
        //打开文件,若失败则输出提示信息
        printf("Error Opening File\n");
        exit(1);
    }
    if ((bytes = read(handle, buf, 10)) == -1) {
        //读取内容,若失败输出提示信息
        printf("Read Failed.\n");
        exit(1);
    }else printf("Read: %d bytes read.\n", bytes);
        //成功则输出所读取的内容
}
```

运行上述程序,首先打开指定文件,若失败则输出提示信息;然后读取文件内容,若失败输出提示信息,否则输出所读取的内容。

10.6 写文件函数: write()

函数 write()用于将指定的内容写入由 open()所打开的文件,其

原型如下。

```
int write(int handle, void *buf, int nbyte);
```

【参数】 参数handle为要写入的文件句柄；参数buf为要写入的内容；参数nbyte表示要写入的长度。

【返回值】 若该函数将指定长度的内容成功写入文件，则返回所写入的内容长度；否则，返回-1。

【例 10-6】 下面的示例演示了write()函数的使用，采用该函数向文件写入指定长度的内容，代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <stdlib.h>          //加入标准工具库
#include <io.h>              //加入输入输出库
#include <string.h>          //加入字符串库

void main( ){
    int handle;
    char string[40];
    int length, res;
    if ((handle = open("test.txt", O_WRONLY | O_CREAT | O_TRUNC,
        S_IREAD | S_IWRITE)) == -1) //打开文件
    {
        printf("Error opening file.\n");
        exit(1);
    }
    strcpy(string, "Hello, world!\n"); //将变量复制到数组中
    length = strlen(string);          //计算长度
    if ((res = write(handle, string, length)) != length) { //写入文件
        printf("Error writing to the file.\n"); //若失败输出提示信息
        exit(1);
    }
    printf("Wrote %d bytes to the file.\n", res); //输出结果
    close(handle); //关闭文件
}
```

运行上述程序，首先打开指定的文件，若失败则输出提示信息；然后写入指定长度的内容，判断是否写入成功。若写入成功，输出所写入的内容大小；若写入失败，则输出提示信息。

10.7 文件字节数统计函数: filelength()

函数 filelength() 用于统计由 open() 所打开文件的字节数, 其原型如下。

```
long filelength(int handle);
```

【参数】 参数 handle 表示要统计的文件句柄。

【返回值】 该函数返回指定文件的字节数。

【例 10-7】 下面的示例演示了 filelength() 函数的使用, 采用该函数统计指定文件的字节数, 代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>             //加入标准输入输出库
#include <io.h>                //加入输入输出库
void main( ){
    int handle;
    char buf[11] = "0123456789";
    handle = open("test.txt ", O_CREAT); //打开文件
    write(handle, buf, strlen(buf));     //写入内容
    printf("file length in bytes: %ld\n", //输出文件大小
        filelength(handle));
    close(handle);                      //关闭文件
}
```

运行上述程序, 首先打开指定文件 DUMMY.FIL, 并写入指定的内容, 然后输出所打开文件的字节数。

10.8 获取文件指针位置函数: tell()

函数 tell() 用于获取由 open() 函数所打开文件的指针位置, 其原型如下。

```
long tell(int handle);
```

【参数】 参数 handle 为获取文件指针的文件句柄。

【返回值】 该函数执行后将返回给定文件的文件指针位置。

【例 10-8】 下面的示例演示了 `tell()` 函数的使用，采用该函数获取所给定文件的文件指针位置，代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
#include <io.h>               //加入输入输出库
void main( ){
    int handle;
    char msg[ ] = "Hello world";
    if ((handle = open("test.txt", O_CREAT | O_TEXT | O_APPEND))
== -1)                        //打开文件
    {
        perror("Error:");
        return 1;
    }
    write(handle, msg, strlen(msg));           //写入内容
    printf("The file pointer is at byte %ld\n", tell(handle));
                                           //输出文件指针位置
    close(handle);                             //关闭文件
}
```

运行上述程序，首先打开指定的文件，打开失败则输出提示信息；然后写入指定的内容，并输出文件指针的位置。

10.9 移动文件指针函数：lseek()

函数 `lseek()` 用于移动由 `open()` 函数所开文件的指针，其原型如下。

```
long lseek(int handle, long offset, int fromwhere);
```

【参数】 参数 `handle` 为要移动文件指针的文件句柄；参数 `offset` 表示要移动的偏移量；参数 `fromwhere` 表示文件指针的当前位置。

【返回值】 该函数将返回移动文件指针后的文件指针位置。

【例 10-9】下面的示例演示了 lseek() 函数的使用, 将文件指针移动指定偏移量的位置, 代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>             //加入标准输入输出库
#include <io.h>                //加入输入输出库
void main( ){
    int handle;
    char msg[ ] = "This is a test";
    char ch;
    handle = open("test.txt ", O_CREAT | O_RDWR, S_IREAD | S_IWRITE);
                                     //打开文件
    write(handle, msg, strlen(msg)); //写入内容
    lseek(handle, 0L, SEEK_SET);     //将文件指针移动到文件开始
    do{                             //读取文件内容直到文件结束并输出
        read(handle, &ch, 1);
        printf("%c", ch);
    } while (!eof(handle));
    close(handle);                 //关闭文件
}
```

运行上述程序, 首先打开指定文件并写入内容; 然后将文件指针移动到文件开始, 再读取文件内容并进行输出直到文件结束。

10.10 获取文件时间函数: getftime()

函数 getftime() 用于获取给定打开文件的时间, 其原型如下。

```
int getftime(int handle, struct ftime *ftimep);
```

【参数】参数 handle 为要获取文件的句柄; 参数 ftimep 为存储文件时间的结构体。

【返回值】函数将给定文件的文件时间读取到时间结构体中, 并返回 0。

【例 10-10】下面的示例演示了 getftime() 函数的使用, 采用该函数获

取指定文件的文件时间，代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <io.h> //加入输入输出库
void main( ){
    FILE *stream;
    struct ftime ft; //文件时间结构体
    if ((stream = fopen("test.txt ",
        "wt")) == NULL) { //打开文件
        fprintf(stderr, "Cannot open output file.\n");//输出
        return 1;
    }
    getftime(fileno(stream), &ft); //获取文件时间
    printf("File time: %u:%u:%u\n",
        ft.ft_hour, ft.ft_min,
        ft.ft_tsec * 2); //输出文件时间
    printf("File date: %u/%u/%u\n",
        ft.ft_month, ft.ft_day,
        ft.ft_year+1980); //输出文件日期
    fclose(stream); //关闭文件
}
```

运行上述程序，首先定义一个用于保存文件时间的结构体；然后打开文件并获取文件时间保存到结构体变量，再输出结构体中的详细信息。

10.11 改变文件访问方式函数：chmod()

函数 chmod() 用于改变给定文件的访问方式，其原型如下。

```
int chmod(const char *filename, int permiss);
```

【参数】 参数 filename 为要修改的文件名；参数 permiss 为要修改的权限。

【返回值】 该函数将指定文件的访问方式修改为给定的方式，修改成

功后返回0。

【例 10-11】下面的示例演示了chmod()函数的使用,使用该函数改变给定文件的访问方式,代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <io.h> //加入输入输出库
void make_read_only(char *filename); //声明函数
void main( ){
    make_read_only("NOTEXIST.FIL"); //改变文件属性为只读
    make_read_only("test.txt "); //改变文件属性为只读
}
void make_read_only(char *filename){
    int stat;
    stat = chmod(filename, S_IREAD); //修改为只读属性
    if (stat) //判断修改结果
        printf("Couldn't make %s read-only\n", filename); //若修改
        没成功,输出提示信息
    else printf("Made %s read-only\n", filename);
}
```

运行上述代码后,调用所声明的函数将指定文件的访问方式修改为只读。

10.12 设置文件打开方式函数: setmode()

函数 setmode()用于设置已打开文件的打开方式,其原型如下。

```
int setmode(int handle, unsigned mode);
```

【参数】参数handle为要设置的文件句柄;参数mode为要设置的方式。

【返回值】该函数执行成功后将返回0;否则,返回-1。

【例 10-12】下面的示例演示了setmode()函数的使用,采用该函数将指定文件的打开方式修改为所给定的方式,代码如下。

```

#include <stdio.h> //加入标准输入输出库
#include <io.h> //加入输入输出库
void main( ){
    int result;
    result = setmode(fileno(stdprn), O_TEXT); //设置文件的打开方式
    if (result == -1) //根据设置的结果输出相应信息
        perror("Mode not available\n");
    else printf("Mode successfully switched\n");
}

```

运行上述代码,首先声明用于保存设置文件打开方式结果的变量result;然后设置文件的打开方式,并根据打开文件结果输出相应的提示信息。

10.13 复制文件句柄函数: dup()

函数dup()用于复制已打开文件的文件句柄,其原型如下。

```
int dup(int handle);
```

【参数】参数handle为要复制的文件句柄。

【返回值】该函数执行后将返回一个文件句柄。

【例 10-13】下面的示例演示了dup()函数的使用,代码如下。

```

#include <string.h> //加入字符串库
#include <stdio.h> //加入标准输入输出库
#include <conio.h> //加入控制台输入输出库
#include <io.h> //加入输入输出库
void flush(FILE *stream); //声明一个函数
void main( ){
    FILE *fp; //定义一个文件类型指针
    char msg[ ] = "This is a test"; //定义一个字符类型的数组
    fp = fopen("test.txt ", "w"); //打开文件
    fwrite(msg, strlen(msg), 1, fp); //写入内容
    printf("Press any key to flush \ DUMMY.FIL:"); //输出提示信息
}

```



```

    flush(fp); //调用所定义的函数
    printf("\nFile was flushed, Press any \ key to quit:");
    //输出提示信息
}
void flush(FILE *stream){
    int duphandle;
    fflush(stream); //清除缓冲区
    duphandle = dup(fileno(stream)); //复制句柄
    close(duphandle); //关闭文件
}

```

运行上述程序, 首先声明一个函数, 打开一个文件并写入相应内容; 然后调用自定义函数复制文件句柄, 清除该文件内容。

10.14 设备类型检查函数: isatty()

函数 isatty() 用于检查给定设备类型, 其原型如下。

```
int isatty(int handle);
```

【参数】 handle 表示要检查的设备文件句柄。

【返回值】 该函数用来检查设备文件句柄, 若成功则返回 true; 否则, 返回 NULL (0)。

【例 10-14】 下面的示例演示了 isatty() 函数的使用, 检测给定的句柄是否为设备类型, 代码如下、

```

#include <stdio.h> //加入标准输入输出库
#include <io.h> //加入输入输出库
void main( ){
    int handle;
    handle = fileno(stdprn); //设备句柄
    if (isatty(handle)) //类型检查
        printf("Handle %d is a device type\n", handle); //输出提示信息
    else printf("Handle %d isn't a device type\n", handle);
}

```


运行上述程序，首先获取一个设备句柄；然后根据该句柄判断是否为设备类型，并输出相应的提示信息。

10.15 文件共享锁设置函数：lock()

函数 lock() 用于设置指定文件的文件共享锁，其原型如下。

```
int lock(int handle, long offset, long length);
```

【参数】 参数 handle 为要设置共享锁的文件句柄；参数 offset 为要设置的位置；参数 length 为要设置的长度。

【返回值】 若该函数设置成功，则返回 true；若失败，则返回 false。

【例 10-15】 下面的示例演示了 lock() 函数的使用，采用该函数给指定的文件设置共享锁，代码如下。

```
#include <io.h> //加入输入输出库
#include <process.h> //加入进程管理库
#include <stdio.h> //加入标准输入输出库
void main( ){
    int handle, status;
    long length;
    handle = sopen("c:\\autoexec.bat",
        O_RDONLY, SH_DENYNO, S_IREAD); //打开文件
    if (handle < 0) { //判断打开成功与否
        printf("sopen failed\n");
        exit(1);
    }
    length = filelength(handle); //计算文件长度
    status = lock(handle, 0L, length/2); //设置共享锁
    if (status == 0) //判断设置成功与否
        printf("lock succeeded\n");
    else printf("lock failed\n");
    status = unlock(handle, 0L, length/2); //解除共享锁
    if (status == 0) //判断解除成功与否
        printf("unlock succeeded\n");
}
```

```

    else printf("unlock failed\n");
    close(handle); //关闭文件
}

```

运行上述程序, 首先打开文件, 计算文件长度; 然后设置文件共享锁, 根据返回结果判断是否设置成功, 并输出相应的提示信息; 最后解除共享锁, 关闭文件。

10.16 连接文件句柄函数: fdopen()

函数 fdopen() 用于将流与文件句柄连接, 其原型如下。

```
FILE *fdopen(int handle, char *type);
```

【参数】 参数 handle 为要进行操作的文件句柄; 参数 type 指定流打开的方式。

【返回值】 该函数执行成功后将返回流; 否则, 将返回 NULL。

【例 10-16】 下面的示例演示了 fdopen() 函数的使用, 采用该函数将流与文件句柄连接, 代码如下。

```

#include <stdio.h> //加入标准输入输出库
#include <io.h> //加入输入输出库
void main() {
    int handle;
    FILE *stream; //定义文件流
    handle = open("test.txt ", O_CREAT,
        S_IREAD | S_IWRITE); //打开文件
    stream = fdopen(handle, "w"); //将流与文件句柄连接
    if (stream == NULL) //判断是否连接成功
        printf("fdopen failed\n");
    else {
        fprintf(stream, "Hello world\n"); //输出提示信息
        fclose(stream); //关闭流
    }
}

```

运行上述程序，首先定义一个文件流指针；然后打开文件，将流与文件句柄连接，根据连接结果输出相应的提示信息。

10.17 文件访问权限设置函数：access()

函数 access() 用于设置给定文件的访问权限，其原型如下。

```
int access(const char *filename, int amode);
```

【参数】 参数 filename 为要设置的文件名；参数 amode 为要设置的权限。

【返回值】 该函数执行成功将返回 true；否则，返回 false。

【例 10-17】 下面的示例演示了 access() 函数的使用，通过该函数设置给定文件的文件访问权限，代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <io.h> //加入输入输出库
int file_exists(char *filename); //声明函数
void main( ){
    printf("Does NOTEXIST.FIL exist: %s\n", //输出设置结果
        file_exists("NOTEXIST.FIL") ? "YES" : "NO");
}
int file_exists(char *filename) { //自定义函数设置文件访问权限
    return (access(filename, 0) == 0);
}
```

运行上述程序，通过调用自定义函数设置给定文件的文件访问权限，并根据设置结果输出相应的提示信息。

第 11 章 浮点数据处理库: float.h

在一些涉及数学浮点运算的程序设计中,有时会用到一些浮点数的运算。C 语言专门提供了一个针对浮点数运算的函数库——float.h,该文件库中存在的常用函数主要包括重置浮点运算系统函数和获取浮点数处理器状态值函数。

11.1 重置浮点运算系统函数: _fpreset()

函数 _fpreset() 用于初始化浮点数运算,其原型如下。

```
void _fpreset( )
```

【参数】该函数没有参数。

【返回值】该函数直接重置系统的浮点数运算,没有返回值。

【例 11-1】下面的示例演示了 _fpreset() 函数的使用,采用该函数重置浮点数运算,代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <signal.h>          //加入信号定义函数库
#include <setjmp.h>           //加入函数跳转函数库
#include <stdlib.h>          //加入标准工具库
#include <float.h>           //加入浮点数据处理库
#include <math.h>            //加入数学函数库
#include <string.h>          //加入字符串库
jmp_buf mark;                //跳转地址
int fperr;                   //全局错误号
void __cdecl fphandler( int sig, int num ); //自定义函数声明
void fpcheck( void );        //自定义函数声明
void main( void ){
    double n1, n2, r;
```

```

int jmpret;
_control87( 0, _MCW_EM );
if( signal( SIGFPE, fphandler ) == SIG_ERR ){
    //设置浮点处理错误函数
    fprintf( stderr, "Couldn't set SIGFPE " );
    abort( );
}
jmpret = setjmp( mark );//保存当前环境信息, 若返回 0, 执行相应条件
if( jmpret == 0 ) {
    printf( "Test for invalid operation - " );
    printf( "enter two numbers: " );
    scanf( "%lf %lf", &n1, &n2 );
    r = n1 / n2;
    printf( "%4.3g / %4.3g = %4.3g", n1, n2, r );
    //如果发生错误, 这行不会执行
    r = n1 * n2;
    printf( "%4.3g * %4.3g = %4.3g", n1, n2, r );
    //如果发生错误, 这行不会执行
} else //否则
    fpcheck( );
}

void fphandler( int sig, int num ){
    fperr = num; //赋值
    _fpreset( ); //初始化浮点运算
    longjmp( mark, -1 ); //跳转
}

```

运行上述程序, 首先声明两个错误处理函数, 保存当前环境信息, 并根据错误信息执行相应的错误处理函数。

11.2 获取浮点处理器状态值函数: `_status87()`

函数 `_status87()` 用于获取 CPU 浮点数处理器的状态值, 其原型如下。

```
unsigned int _status87(void);
```

【参数】 该函数没有参数。

【返回值】 该函数将获取浮点处理器状态值，并返回类型为无符号数的系统状态值。

【例 11-2】 下面的示例演示了 `_status87()` 函数的使用，使用该函数获取浮点处理器状态值，代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <float.h> //加入浮点运算库
void main( ){
    float x; //定义浮点型变量
    double y = 1.5e-100; //定义 double 变量
    printf("Status 87 before error: %x\n", _status87( ));
    //输出当前状态
    x = y; //强制产生一个错误
    y = x;
    printf("Status 87 after error : %x\n", _status87( ));
    //输出当前状态信息
}
```

运行上述代码，首先定义两个浮点型的变量；然后输出当前浮点处理状态，强制将 `double` 类型值赋给 `float` 型变量 `x`，这时将产生一个错误；最后输出此时的浮点处理器状态信息。

第 12 章 控制台输入输出函数库:

conio.h

在程序设计中,常常需要进行控制台输入输出等操作。C 语言专门提供了一个针对控制台输入输出操作的函数库——conio.h,该文件库中存在的常用函数主要包括控制台字符的获取、控制台的设置等操作函数。

12.1 获取字符函数: getch()

函数 getch()用于从控制台无回显地取一个字符,其原型如下。

```
int getch(void);
```

【参数】该函数没有参数。

【返回值】该函数执行后将 从控制台取一个字符。

【例 12-1】下面的示例演示了 getch()函数的使用,在程序中采用该函数从控制台获取一个字符,代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <conio.h>           //加入控制台输入输出库
void main( ){
    char ch;                  //定义接收变量
    printf("Input a character:"); //输入提示信息
    ch = getch( );            //从控制台获取一个字符
    printf("\nYou input a '%c'\n", ch); //输出获取的字符
}
```

运行上述程序,首先定义一个用于保存接收字符的指针;然后输出提示信息,读者在屏幕上输入一个字符后,将该字符输出。

12.2 清除字符函数: clreol()

函数 `clreol()` 用于在文本窗口中清除字符到行末, 其原型如下。

```
void clreol(void);
```

【参数】 该函数没有参数。

【返回值】 该函数没有返回值。

【例 12-2】 下面的示例演示了 `clreol()` 函数的使用, 使用该函数将当前所在行的光标后的所有文件清除, 代码如下。

```
#include <conio.h> //加入控制台输入输出库
void main() {
    clrscr(); //清除屏幕
    printf("The function CLREOL clears all characters from\nthe\r\n"); //输出提示信息
    printf("cursor position to the end of the line within the\r\n"); //输出提示信息
    printf("current text window, without moving the cursor.\r\n"); //输出提示信息
    printf("Press any key to continue . . ."); //输出提示信息
    gotoxy(14, 4); //跳转到屏幕指定位置
    getch(); //获取一个字符
    clreol(); //清除当前位置后的所有文本
}
```

运行上述代码, 首先清除屏幕, 然后在屏幕上输出几行提示信息, 再将光标移动到指定行, 读者输入一个字符后, 将清除掉当前行后面的所有文本。

12.3 清除文本函数: clrscr()

函数 `clrscr()` 用于清除文本模式窗口, 其原型如下。

```
void clrscr(void);
```

【参数】该函数没有参数。

【返回值】该函数直接清除当前屏幕，没有返回值。

【例 12-3】下面的示例演示了clrscr()函数的使用，采用该函数直接清除屏幕上的所有文本，代码如下。

```
#include <conio.h>                                //加入控制台输入输出库
void main( ){
    int i;                                          //定义循环变量
    clrscr( );                                     //首先清除屏幕
    for (i = 0; i < 20; i++)
        cprintf("%d\r\n", i);                    //循环输出
    cprintf("\r\nPress any key to clear screen");//输出提示信息
    getch( );                                     //获取输入字符
    clrscr( );                                     //清除屏幕
    cprintf("The screen has been cleared!");      //输出提示信息
}
```

运行上述代码，首先清除当前屏幕，在屏幕上循环输出变量i的值；然后输出提示信息，读者在控制台输入一个字符后，当前屏幕所有文本将会被清除。

12.4 写字符函数：cputs()

函数cputs()用于将给定的字符写到屏幕，其原型如下。

```
void cputs(const char *string);
```

【参数】参数string为要输出的字符。

【返回值】该函数直接将给定的字符输出到控制台，输出后没有返回值。

【例 12-4】下面的示例演示了cputs()函数的使用，代码如下。

```
#include <conio.h>                                //加入控制台输入输出库
void main( ){
    clrscr( );                                     //清除屏幕
```



```

window(10, 10, 80, 25);           //创建一个文本模式窗口
cputs("This is within the window\r\n"); //输出字符到该窗口
}

```

运行上述程序, 首先清屏幕上的所有文本, 然后创建一个文本窗口, 输出一些字符到该窗口, 待读者输入一个字符后退出。

12.5 删除行函数: delline()

函数 `delline()` 用于在文本窗口中删去一行, 其原型如下。

```
void delline(void);
```

【参数】 该函数没有参数。

【返回值】 该函数直接在文本窗口删除一行, 删除后没有返回值。

【例 12-5】 下面的示例演示了 `delline()` 函数的使用, 采用该函数删除光标所在行, 代码如下。

```

#include <conio.h>           //加入控制台输入输出库
void main( )
{
    clrscr( );               //清除屏幕
    cprintf("The function DELLINE deletes \
the line containing the\r\n"); //输出提示信息
    cprintf("cursor and moves all lines \
below it one line up.\r\n"); //输出提示信息
    cprintf("DELLINE operates within the \
currently active text\r\n");
    cprintf("window. Press any key to \
continue . . .");           //输出提示信息
    gotoxy(1,2);             //将光标移动到第二行第一列
    getch( );                //等待输入
    delline( );              //删除光标所在行
}

```

运行上述程序, 首先清除当前屏幕, 输出三行提示信息; 然后将

光标跳转到第二行第一列，删除这一行，待输入字符后结束程序。

12.6 文本复制函数：gettext()

函数 `gettext()` 用于将文本方式屏幕上的文本复制到存储区，其原型如下。

```
int gettext(int left, int top, int right, int bottom, void *destin);
```

【参数】 参数 `left` 表示复制区域的开始列；参数 `top` 表示要复制区域的开始行；参数 `bottom` 表示要复制区域的结束行；参数 `right` 表示为复制区域的结束列；参数 `destin` 表示保存的缓冲区。

【返回值】 该函数将指定区域内的文本复制到缓冲区，若复制成功，则返回 0；否则，返回 -1。

【例 12-6】 下面的示例演示了 `gettext()` 函数的使用，采用该函数将指定区域的文本复制到缓冲区，代码如下。

```
#include <conio.h>           //加入控制台输入输出库
char buffer[4096];          //定义缓冲区
void main() {
    int i;
    clrscr();                //清屏
    for (i = 0; i <= 20; i++)
        cprintf("Line %d\r\n", i); //在屏幕上输出变量 i
    gettext(1, 1, 80, 25, buffer); //将整个 80x25 的屏幕复制到缓冲区
    gotoxy(1, 25);           //光标跳转到 25 行 1 列
    cprintf("Press any key to clear screen..."); //输出提示信息
    getch();                 //获取字符
    clrscr();                //清屏
    gotoxy(1, 25);           //光标跳转到 25 行 1 列
    cprintf("Press any key to restore screen..."); //输出提示信息
    getch();                 //获取字符
    puttext(1, 1, 80, 25, buffer); //将缓冲区中的内容复制到指定区域
    gotoxy(1, 25);           //光标跳转到 25 行 1 列
```

```

    cprintf("Press any key to quit..."); //输出提示信息
    getch( );                             //获取字符
}

```

运行上述程序, 首先在屏幕上循环输出变量 *i*, 然后将整个屏幕复制到缓冲区, 将屏幕上的内容清空, 待输入字符后将缓冲区中的内容输出到文本窗口。

12.7 设置光标函数: gotoxy()

函数 `gotoxy()` 用于在文本窗口中设置光标, 其原型如下。

```
void gotoxy(int x, int y);
```

【参数】 参数 *x* 为文本窗口中的行; 参数 *y* 为文本窗口中的列。

【返回值】 该函数将光标定位于所指定的位置, 没有返回值。

【例 12-7】 下面的示例演示了 `gotoxy()` 函数的使用, 采用该函数将光标自动定位到指定位置, 代码如下。

```

#include <conio.h> //加入控制台输入输出库
void main( ){
    clrscr( );      //清除屏幕
    gotoxy(35, 12); //光标跳转到 12 行 35 列
    cprintf("Hello world"); //输出内容
    getch( );       //获取字符
}

```

运行上述程序, 首先清除屏幕上的所有内容, 将光标定位于指定位置, 再于该位置输出内容。

12.8 高亮显示文本函数: highvideo()

函数 `highvideo()` 用于选择高亮度文本字符, 其原型如下。


```
void highvideo(void);
```

【参数】该函数没有参数。

【返回值】该函数没有返回值。

【例 12-8】下面的示例演示了highvideo()函数的使用,采用该函数高亮度显示文本,代码如下。

```
#include <conio.h> //加入控制台输入输出库
void main( ){
    clrscr( ); //清除屏幕
    lowvideo( ); //选择低亮度字符
    cprintf("Low Intensity text\r\n"); //输出文本
    highvideo( ); //选择高亮度字符
    gotoxy(1,2); //跳转到2行1列
    cprintf("High Intensity Text\r\n"); //输出文本
}
```

运行上述程序,首先清除屏幕内容,设置低亮度显示文本,输出文本,此时屏幕将以低亮度显示文本;然后设置以高亮度显示,再输出文本,此时文本以高亮度显示。

12.9 插入空行函数: insline()

函数 insline()用于在文本窗口中插入一个空行,其原型如下。

```
void insline(void);
```

【参数】该函数没有参数。

【返回值】该函数在文本窗口中插入一个空行,没有返回值。

【例 12-9】下面的示例演示了insline()函数的使用,采用该函数在指定位置插入一个空行,代码如下。

```
#include <conio.h> //加入控制台输入输出库
void main( ){
    clrscr( ); //清除屏幕
```

```

cprintf("INLINE inserts an empty line in the text window\r\n");
//输出提示信息
cprintf("at the cursor position using the current text\r\n");
//输出提示信息
cprintf("background color. All lines below the empty one\r\n");
//输出提示信息
cprintf("move down one line and the bottom line scrolls\r\n");
//输出提示信息
cprintf("off the bottom of the window.\r\n");
//输出提示信息
cprintf("\r\nPress any key to continue:");
//输出提示信息

gotoxy(1, 3); //光标跳转到 3 行 1 列
getch();      //获取字符
insline();    //插入空行
getch();      //获取字符
}

```

运行上述程序, 首先清除屏幕, 在文本窗口中输出几行文本; 然后将光标跳转到 3 行 1 列, 插入一个空行, 待输入字符后退出程序。

12.10 文本复制函数: puttext()

函数 `puttext()` 用于将文本从存储区复制到屏幕, 其原型如下。

```
int puttext(int left, int top, int right, int bottom, void *source);
```

【参数】 参数 `left` 为目标区域的开始列; 参数 `top` 为目标区域的开始行; 参数 `right` 为目标区域的结束列; 参数 `bottom` 为目标区域的结束行; 参数 `source` 为要复制的源缓冲区。

【返回值】 该函数将源缓冲区的内容复制到指定屏幕的区域, 执行成功后将返回 0, 否则返回 -1。

【例 12-10】 下面的示例演示了 `puttext()` 函数的使用, 在程序中将缓冲区的内容复制到指定区域, 代码如下。

```

#include <conio.h>                                //加入控制台输入输出库
void main( ){
    char buffer[512];                              //定义缓冲区大小
    clrscr( );                                     //清除屏幕
    gotoxy(20, 12);                                //跳转到 12 行 20 列
    cprintf("This is a test. Press any key to continue ...");
                                                    //输出提示信息

    getch( );                                       //获取字符
    gettext(20, 12, 36, 21,buffer);               //复制文本
    clrscr( );                                     //清除屏幕
    gotoxy(20, 12);                                //跳转到 12 行 20 列
    puttext(20, 12, 36, 21, buffer);              //输出到屏幕
    getch( );                                       //获取字符
}

```

运行上述程序，首先定义一个缓冲区大小，清除屏幕，在屏幕指定位置输出文本，复制指定区域到缓冲区中；然后清除屏幕，在指定位置将缓冲区中内容复制到屏幕。

12.11 设置文本属性函数：textattr()

() textattr: 设置文本属性

函数 textattr()同时设置文本的字符和背景颜色，其原型如下。

```
void textattr(int attribute);
```

【参数】 参数attribute的值表示颜色形式编码的信息。

【返回值】 该函数没有返回值。

【例 12-11】 下面的示例演示了textattr()函数的使用，使用该函数设置其后文本的显示属性，代码如下。

```

#include <conio.h>                                //加入控制台输入输出库
void main( ){
    int i;
    clrscr( );                                     //清除屏幕
    for (i=0; i<9; i++) {                         //循环输出，以不同的颜色显示

```



```

        textattr(i + ((i+1) << 4));    //设置文本显示属性
        cprintf("This is a test\r\n"); //输出文本
    }
}

```

运行上述代码, 首先清除屏幕, 在屏幕上以不同的字符颜色和背景颜色显示循环输出的文本。

注意: 背景颜色应左移 4 位才能使 3 位背景颜色移到正确位置。

12.12 文本背景色选择函数: textbackground()

函数 textbackground() 用于选择新的文本背景颜色, 其原型如下。

```
void textbackground(int color);
```

【参数】 参数 color 为颜色值。

【返回值】 该函数没有返回值。

【例 12-12】 下面的示例演示了 textbackground() 函数的使用, 采用该函数给文本指定背景颜色, 代码如下。

```

#include <conio.h>                                //加入控制台输入输出库
void main() {
    int i, j;
    clrscr();                                       //清除屏幕
    for (i=0; i<9; i++) {                          //循环输出
        for (j=0; j<80; j++)                      //循环输出字符 "C"
            cprintf("C");
        cprintf("\r\n");                          //换行
        textcolor(i+1);                          //设置文本颜色
        textbackground(i);                        //设置背景颜色
    }
}

```

运行上述代码, 首先清除屏幕, 循环输出字符 “C”, 并对所输出的文本设置前景颜色和背景颜色。

12.13 文本字符颜色选择函数: textcolor()

函数 textcolor()用于在文本模式中选择新的字符颜色,其原型如下。

```
void textcolor(int color);
```

【参数】 参数color为要选择的颜色值。

【返回值】 该函数没有返回值。

【例 12-13】 下面的示例演示了textcolor()函数的使用,使用该函数给文本设置颜色值,代码如下。

```
#include <conio.h> //加入控制台输入输出库
void main( ){
    int i;
    for (i=0; i<15; i++) { //循环输出文本
        textcolor(i); //循环设置文本颜色
        cprintf("Foreground Color\r\n"); //循环输出文本
    }
}
```

运行上述程序,通过循环的方式设置所输出文本的颜色。

12.14 文本模式设置函数: textmode()

函数 textmode()用于将屏幕设置成文本模式,其原型如下。

```
void textmode(int mode);
```

【参数】 参数mode为要设置的模式。

【返回值】 调用该函数后,屏幕复位,并且所有字符的属性恢复其默认值,没有返回值。

【例 12-14】 下面的示例演示了textmode()函数的使用,代码如下。

```
#include <conio.h> //加入控制台输入输出库
```

```

void main( ){
    textmode(BW40);           //以 40 列黑白显示
    cprintf("ABC");          //输出文本
    getch( );                //获取字符
    textmode(C40);           //以 40 列彩色显示
    cprintf("ABC");          //输出文本
    getch( );                //获取字符
    textmode(BW80);          //以 80 列黑白显示
    cprintf("ABC");          //输出文本
    getch( );                //获取字符
    textmode(C80);           //以 80 列彩色显示
    cprintf("ABC");
    getch( );
    textmode(MONO);           //以 80 列单色显示
    cprintf("ABC");
}

```

运行上述程序, 分别设定每一次输出文本的显示模式。

12.15 返回水平光标位置函数: wherex()

函数 wherex() 用于返回窗口内水平光标位置, 其原型如下。

```
int wherex(void);
```

【参数】该函数没有参数。

【返回值】该函数执行后返回窗口内的水平光标位置。

12.16 返回垂直光标位置函数: wherey()

函数 wherey() 用于返回窗口内垂直光标位置, 其原型如下。

```
int wherey(void);
```


【参数】该函数没有参数。

【返回值】该函数返回窗口内光标所在列。

【例 12-15】下面的示例演示了 `wherex()`、`wherey()` 函数的使用，使用该函数返回光标在当前窗口中的垂直位置，代码如下。

```
#include <conio.h> //加入控制台输入输出库
void main( ){
    clrscr( ); //清除屏幕
    gotoxy(10,10); //将光标跳转到10行10列
    cprintf("Current location is X: %d Y: %d\r\n", wherex( ),
wherey( )); //输出光标垂直位置
    getch( ); //获取字符
}
```

运行上述程序，首先清除屏幕，将光标跳转到 10 行 10 列，然后获取并输出当前光标所在列。

12.17 定义活动文本窗口函数：window()

()xerexw : 函数置光标水平位置 21.51

函数 `window()` 用于定义活动文本模式窗口，其原型如下。

```
void window(int left, int top, int right, int bottom);
```

【参数】参数 `left` 为活动窗口的开始列；参数 `top` 为活动窗口的开始行；参数 `right` 为活动窗口的结束列；参数 `bottom` 为活动窗口的结束行。

【返回值】该函数按给定参数设定活动窗口，没有返回值。

【例 12-16】下面的示例演示了 `window()` 函数的使用，使用该函数设定活动窗口，代码如下。

```
#include <conio.h> //加入控制台输入输出库
void main( ){
    window(10,10,40,11); //将指定区域设定为活动窗口
    textcolor(BLACK); //设置文本颜色
}
```

```

textbackground(WHITE);           //设置背景色
cprintf("This is a test\r\n");  //输出字符
)

```

运行上述程序, 首先将指定区域设定为活动窗口, 然后设置该窗口的文本颜色和背景颜色, 再输出测试字符。

第 13 章 DEBUG 相关函数库: assert.h

在程序设计中,常常需要进行程序的调试操作。C 语言专门提供了一个针对调试操作的函数库——assert.h,该文件库中存在一个调试函数 assert(),其用于在程序中判断某些必备条件是否通过,若不符合给定的条件,直接终止程序执行。

函数 assert()用于测试一个条件并可能使程序终止,其原型如下。

```
void assert(int test);
```

【参数】参数test为要测试的条件。

【返回值】该函数没有返回值。

【例 13-1】下面的示例演示了assert()函数的使用,采用该函数在程序中测试给定的条件,并终止程序,代码如下。

```
#include <assert.h>           //加入 DEBUG 相关函数库
#include <stdio.h>             //加入标准输入输出库
#include <stdlib.h>            //加入标准工具库
struct ITEM {                 //定义结构体
    int key;
    int value;
};
void additem(struct ITEM *itemptr) { //添加参数 itemptr 到结构体,并
    确保它不是 NULL
    assert(itemptr != NULL);      //测试参数 itemptr, 并终止程序
}
void main() {
    additem(NULL);                //调用自定义函数 additem
}
```

运行上述程序,首先定义一个自定义函数,检测所给参数是否为空。若为空,则直接终止程序,然后在主程序中调用该函数。

第 14 章 BIOS 相关函数库: bios.h

在程序设计中,有时需要涉及到 BIOS 的相应操作。C 语言专门提供了一个针对 BIOS 操作的函数库——bios.h,该文件库中存在的常用函数主要包括串行通信、键盘接口、设备检查等操作函数。

14.1 串行 IO 通信函数: bioscom()

函数 bioscom()用于实现串行 I/O 通信,其原型如下。

```
int bioscom(int cmd, char abyte, int port);
```

【参数】参数cmd为要执行的命令;参数abyte为命令所需要参数;参数port为通信端口。

【返回值】该函数返回一个 16 位的整数。

【例 14-1】下面的示例演示了bioscom()函数的使用,通过该函数与指定端口进行通信,代码如下。

```
#include <bios.h> //加入 BIOS 通信函数库
#include <conio.h> //加入控制台输入输出库
#define COM1 0
#define DATA_READY 0x100
#define TRUE 1
#define FALSE 0
#define SETTINGS ( 0x80 | 0x02 | 0x00 | 0x00) //设置波特率等
void main( ){
    int in, out, status, DONE = FALSE;
    bioscom(0, SETTINGS, COM1); //初始化该接口
    cprintf("... BIOSCOM [ESC] to exit ...\n"); //输出提示信息
    while (!DONE)
    {
```

```

    status = bioscom(3, 0, COM1);           //返回接口的状态
    if (status & DATA_READY)
        if ((out = bioscom(2, 0, COM1) & 0x7F) != 0)
            //接收一个字符
            putchar(out);
    if (kbhit())
    {
        if ((in = getch()) == '\x1B')
            DONE = TRUE;
        bioscom(1, in, COM1);              //发送一个字符
    }
}

```

运行上述程序, 首先对指定端口进行相关设置, 然后返回接口的状态, 若接口准备好了, 接收一个字符。

14.2 软硬盘 IO 函数: biosdisk()

函数 biosdisk() 用于对驱动器做相关操作, 其原型如下。

```

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

```

【参数】参数 cmd 为功能号; 参数 drive 为驱动器号(0=A, 1=B, 0x80=C, 0x81=D, 0x82=E 等); 参数 head 为磁头号; 参数 track 为磁道号; 参数 nsects 为扇区号; 参数 buffer 为所写的数据。

【返回值】该函数执行后将返回一个组合而成的状态字节。

【例 14-2】下面的示例演示了 biosdisk() 函数的使用, 使用该函数检测 A 驱动器是否就绪, 代码如下。

```

#include <bios.h>           //加入 BIOS 通信函数库
#include <stdio.h>          //加入标准输入输出库
void main() {
    int result;
}

```

```

char buffer[512];
printf("Testing to see if drive a: is ready\n");
result = biosdisk(4,0,0,0,0,1,buffer); //检查 A 驱动器第一扇区
result &= 0x02;
(result) ? (printf("Drive A: Ready\n")) :
    (printf("Drive A: Not Ready\n")); //根据结果判断 A 驱动器是否就绪
}

```

运行上述代码,将采用 biosdisk() 函数检测 A 驱动器是就绪,并根据结果输出相应的提示信息。

14.3 检查设备函数: biosequip()

函数 biosequip() 用于检测处理器设备状态,其原型如下。

```
int biosequip(void);
```

【参数】 该函数没有参数。

【返回值】 该函数返回一个字节,每一位表示一个信息。

【例 14-3】 下面的示例演示了 biosequip() 函数的使用,采用该函数检测协处理器是否存在,并输出相应结果,代码如下。

```

#include <stdio.h> //加入标准输入输出库
#include <dos.h> //加入系统接口函数库
#include <stdlib.h>
#include <bios.h>
void main() {
    unsigned result;
    clrscr(); //清除屏幕
    result = biosequip(); //检测设备
    if(result & 0x0002) //根据结果输出相应的信息
        printf("Math co-processor installed\n");
    else
        printf("Math co-processor is not installed.\n");
}

```


运行上述代码,首先采用 biosequip() 函数检测,并根据检测结果输出相应的提示信息。

14.4 键盘接口函数: bioskey()

函数 bioskey() 直接使用 BIOS 服务的键盘接口,其原型如下。

```
int bioskey(int cmd);
```

【参数】 参数 cmd 为要执行的键盘命令。

【返回值】 根据命令返回不同的结果。

【例 14-4】 下面的示例演示了 bioskey() 函数的使用,采用该函数判断当前键盘状态,代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <bios.h>             //加入 BIOS 通信函数库
#include <ctype.h>            //加入字符函数库
#define RIGHT 0x01
#define LEFT 0x02
#define CTRL 0x04
#define ALT 0x08
void main() {
    int key, modifiers;
    while (bioskey(1) == 0); //测试键盘是否可用于读
    key = bioskey(0);        //返回敲键盘上的下一个键
    modifiers = bioskey(2);  //返回当前的键盘状态
    if (modifiers) {
        printf("[");
        if (modifiers & RIGHT) printf("RIGHT");//判断当前右键是否可用
        if (modifiers & LEFT) printf("LEFT");//判断当前左键是否可用
        if (modifiers & CTRL) printf("CTRL");//判断当前【Ctrl】键是否可用
        if (modifiers & ALT) printf("ALT");//判断当前【Alt】键是否可用
        printf("]");
    }
}
```

运行上述程序, 调用 bioskey() 函数检测键盘是否可用于读, 然后返回键盘上的下一个键, 再返回当前键盘的状态并输出相应信息。

14.5 获取存储块大小函数: biosmemory()

函数 biosmemory() 用于获取存储块大小, 其原型如下。

```
int biosmemory(void);
```

【参数】该函数没有参数。

【返回值】该函数返回存储块的大小。

【例 14-5】下面的示例演示了 biosmemory() 函数的使用, 采用该函数获取存储块的大小, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <bios.h> //加入 BIOS 通信函数库
void main() {
    int memory_size;
    memory_size = biosmemory(); //返回存储块大小
    printf("RAM size = %dK\n", memory_size); //输出
}
```

运行上述程序, 获取当前系统中存储块的大小并输出。

14.6 设置 BIOS 时间函数: biostime()

函数 biostime() 用于读取或设置 BIOS 的时间, 其原型如下。

```
long biostime(int cmd, long newtime);
```

【参数】参数 cmd 为功能号, 其值为 0 和 1; 参数 newtime 为要设置的时间。

【返回值】若参数cmd为1时，返回BIOS时间。当参数cmd为1时，将newtime设置为新的时间。

【例 14-6】下面的示例演示了biostime()函数的使用，使用该函数获取当前BIOS时间，代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <bios.h> //加入 BIOS 通信函数库
#include <time.h> //加入时间函数库
#include <conio.h> //加入控制台输入输出库

void main( ){
    long bios_time;
    cprintf("The number of clock ticks since midnight is:\r\n");
    cprintf("The number of seconds since midnight is:\r\n");
    cprintf("The number of minutes since midnight is:\r\n");
    cprintf("The number of hours since midnight is:\r\n");
    cprintf("\r\nPress any key to quit:");
    while(!kbhit( )) {
        bios_time = biostime(0, 0L); //获取 BIOS 时间
        gotoxy(50, 1);
        cprintf("%lu", bios_time); //输出 BIOS 时间
        gotoxy(50, 2);
        cprintf("%.4f", bios_time / CLK_TCK); //输出秒
        gotoxy(50, 3);
        cprintf("%.4f", bios_time / CLK_TCK / 60); //输出分
        gotoxy(50, 4);
        cprintf("%.4f", bios_time / CLK_TCK / 3600); //输出小时
    }
}
```

运行上述程序，首先使用该函数获取 BIOS 时间，然后依次直接输出 BIOS 时间、秒数、分钟数和小时数。

第 15 章 内存相关函数库: mem.h

在程序设计中,常常需要对内存进行相关的操作。C 语言专门提供了一个针对内存操作的函数库——mem.h,该文件库中存在的常用函数主要包括字节的复制、比较、移动等内存操作函数。

15.1 字节复制函数: memccpy()

函数 memccpy() 用于从源内存区域复制指定个字节到目标内存区域,若遇到特定字符则停止复制,其原型如下。

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

【参数】参数destin为目标区域;参数source为源区域;参数ch为要复制的字符;参数n为要复制的字节数。

【返回值】该函数返回指向字符ch后的第一个字符的指针,如果src的前n个字节中不存在ch,则返回NULL。

【例 15-1】下面的示例演示了memccpy()函数的使用,采用该函数从源内存区域中复制指定字节到目标内存区域中,代码如下。

```
#include <mem.h> //加入内存管理库
#include <stdio.h> //加入标准输入输出库
void main() {
    char *src = "This is the source string";
    char dest[50];
    char *ptr;
    ptr = memccpy(dest, src, 'c', strlen(src)); //将源区域复制指定长度字节到目标区域中
    if (ptr) {
```

```

    *ptr = '\0';
    printf("The character was found: %s\n", dest); //输出结果
} else printf("The character wasn't found\n");
}

```

运行上述程序，将源区域的指定长度字节复制到目标区域中，遇到字符“c”就停止复制，并根据结果输出相应的提示信息。

15.2 字节复制函数：memcpy()

函数 memcpy() 用于从源内存区域中复制 n 个字节到目标内存区域中，其原型如下。

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

【参数】参数 destin 为指向目标区域的指针；参数 source 为指向源区域的指针；参数 n 为要复制的字节数。

【返回值】该函数返回指向目标内存区域的指针。

【例 15-2】下面的示例演示了 memcpy() 函数的使用，使用该函数复制指定字节大小的字符到目标区域中，代码如下。

```

#include <stdio.h> //加入标准输入输出库
#include <mem.h> //加入内存管理库
void main( )
{
    char src[ ] = "*****"; //定义源区域
    char dest[ ] = "abcdefghijklmnopqrstuvwxyz0123456709";
                                //定义目标区域
    char *ptr;
    printf("destination before memcpy: %s\n", dest); //输出源
    ptr = memcpy(dest, src, strlen(src)); //复制源到目标区域
    if (ptr) printf("destination after memcpy: %s\n", dest);
                                //输出信息
}

```

```

    else    printf("memcpy failed\n");
}

```

运行上述代码, 将采用 `memcpy()` 函数从源区域复制指定大小的字节数到目标区域, 并输出提示信息。

15.3 字符搜索函数: `memchr()`

函数 `memchr()` 用于从 `buf` 所指内存区域的前 `count` 个字节查找字符 `ch`, 其原型如下。

```
extern void *memchr(void *buf, char ch, unsigned count);
```

【参数】参数 `buf` 为要搜索的区域; 参数 `ch` 为要搜索的字符; 参数 `count` 为要搜索的区域的前 `count` 个字节。

【返回值】当第一次遇到字符 `ch` 时停止查找。如果成功, 返回指向字符 `ch` 的指针, 否则返回 `NULL`。

【例 15-3】下面的示例演示了 `memchr()` 函数的使用, 从指定的区域中查询特定的字符, 代码如下。

```

#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
#include <mem.h>             //加入内存管理库
void main( )
{
    char str[17];
    char *ptr;
    strcpy(str, "This is a string");           //复制字符
    ptr = memchr(str, 'r', strlen(str));       //搜索字符
    if (ptr)    printf("The character 'r' is at position: %d\n", ptr
- str);           //输出提示信息
    else    printf("The character was not found\n");
}

```

运行上述代码, 首先将字符复制到目标区域, 然后在该区域搜索

指定的字符“r”，搜索到时停止搜索，并根据结果输出提示信息。

15.4 串比较函数：memcmp()

函数 memcmp() 用于比较内存区域 buf1 和 buf2 的前 count 个字节，其原型如下。

```
extern int memcmp(void *buf1, void *buf2, unsigned int count);
```

【参数】 参数 buf1 和 buf2 为要比较的字符串；参数 count 为要比较的前 count 个字节。

【返回值】 当 buf1 小于 buf2 时，返回值小于 0；当 buf1 等于 buf2 时，返回值等于 0；当 buf1 大于 buf2 时，返回值大于 0。

【例 15-4】 下面的示例演示了 memcmp() 函数的使用，采用该函数比较给定字符串，代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
#include <mem.h>              //加入内存管理库
void main( )
{
    char *s1="Hello, Programmers!";
    char *s2="Hello, programmers!";
    int r;
    r=memcmp(s1,s2,strlen(s1)); //比较字符串
    if(!r)                       //输出提示信息
        printf("s1 and s2 are identical");
    else
        if(r<0) printf("s1 less than s2");
        else printf("s1 greater than s2");
}
```

运行上述程序，首先定义两个需要进行比较的字符串，然后采用 memcmp() 函数比较两个字符串，并根据比较结果输出提示信息。

15.5 串比较函数: memicmp()

函数 memicmp() 用于比较 s1 和 s2 两个串的前 n 个字节, 忽略大小写, 其原型如下。

```
int memicmp(void *buf1, void *buf2, unsigned n);
```

【参数】 参数 buf1 和 buf2 为要比较的字符串; 参数 n 为要比较的字节数。

【返回值】 当 buf1 小于 buf2 时, 返回值小于 0; 当 buf1 等于 buf2 时, 返回值等于 0; 当 buf1 大于 buf2 时, 返回值大于 0。

【例 15-5】 下面的示例演示了 memicmp() 函数的使用, 以忽略大小写的方式比较给定的字符串, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <string.h> //加入字符串库
void main( )
{
    char *buf1 = "ABCDE123";
    char *buf2 = "abcde456";
    int stat;
    stat = memicmp(buf1, buf2, 5); //比较字符串
    printf("The strings to position 5 are "); //输出信息
    if (stat) printf("not "); //输出结果
    printf("the same\n");
}
```

运行上述程序, 首先定义两个要进行比较的字符串, 然后采用 memicmp() 函数进行比较, 并根据比较结果输出相应的提示信息。

15.6 字节移动函数: memmove()

函数 memmove() 从源内存区域移动 count 个字节到目标内存区

域，其原型如下。

```
extern void *memmove(void *dest, const void *src, unsigned int
count);
```

【参数】参数dest表示要移动的目标区域；参数src为要移动的源区域；参数count为要移动的字节数。

【返回值】src和dest所指内存区域可以重叠，但复制后src内容会被更改，函数返回指向dest的指针。

【例 15-6】下面的示例演示了memmove()函数的使用，采用该函数从源区域移动指定字节数到目标区域，代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
#include <mem.h>              //加入内存管理库
void main() {
    char *dest = "abcdefghijklmnopqrstuvwxyz0123456789";
                                //定义目标区域
    char *src = "*****";      //定义源区域
    printf("destination prior to memmove: %s\n", dest);
                                //输出移动前的区域
    memmove(dest, src, 26);    //移动
    printf("destination after memmove: %s\n", dest);
                                //输出移动后的区域
}
```

运行上述程序，首先定义两个地址指针，然后从源区域移动指定字节数到目标区域，并输出移动前后的目标区域的字符。

15.7 串设置函数：memset()

函数memset()把buffer所指内存区域的前count个字节设置成字符c，其原型如下。

```
extern void *memset(void *buffer, int c, int count);
```


【参数】参数buffer为要设置的区域；参数c为要设置成的字符；参数count为要设置的前count个字节。

【返回值】该函数返回指向buffer的指针。

【例 15-7】下面的示例演示了memset()函数的使用，将指定区域的指定字节数的区域设置为特定的值，代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
#include <mem.h>             //加入内存管理库

void main( ){
    char buffer[ ] = "Hello world\n";    //定义要设置的变量
    printf("Buffer before memset: %s\n", buffer);    //输出设置前的变量
    memset(buffer, '*', strlen(buffer) - 1);    //将指定区域设置为*
    printf("Buffer after memset: %s\n", buffer);    //输出设置后的变量
}
```

运行上述程序，首先定义需要进行设置的变量，然后将该区域设置成“*”，最后输出其设置前后的内容。

第 16 章 进程管理函数库: process.h

在程序设计中,常常需要对进程进行相应的操作。C 语言专门提供了一个针对进程管理的函数库——process.h,该文件库中存在的常用函数主要包括创建运行其他子程序函数和程序间跳转函数。

16.1 创建并运行子程序函数: spawnl()

函数 spawnl()按指定的模式创建并运行子程序,其原型如下。

```
int spawnl(int mode, char *pathname, char *arg0, arg1, ... argn, NULL);
```

【参数】参数mode为调用模式;参数pathname为被调用程序路径;参数arg为调用参数。其中,调用模式有以下 3 种取值。

- P_WAIT 0: 将父过程挂起,直到子过程执行完毕。
- P_NOWAIT 1: 父子过程同时执行, Turboc 不支持。
- P_OVERLAY 2: 子过程覆盖父过程。

【返回值】若函数调用失败,返回-1;若调用成功,将返回 0。

【例 16-1】下面的示例演示了 spawnl()函数的使用,采用该函数在程序中调用其他子程序,代码如下。

```
#include <process.h>           //调用进程管理库
#include <stdio.h>             //调用输入输出库
#include <conio.h>             //调用控制台输入输出库
void main() {
    int result;
    clrscr();                  //清屏
    result = spawnl(P_WAIT, "tcc.exe", NULL); //调用外部程序
    if (result == -1) {        //根据结果输出提示信息
        perror("Error from spawnl");
    }
}
```

```

        exit(1);
    }
}

```

运行上述程序, 首先清除屏幕, 调用外部程序, 并根据调用返回值输出相应的提示信息。

16.2 创建并运行子程序函数: spawnle()

函数 spawnle() 用于创建并运行子程序, 其原型如下。

```

int spawnle(int mode, char *pathname, char *arg0, arg1, ..., argn,
NULL);

```

【参数】 参数 mode 为调用模式; 参数 pathname 为要调用的程序名称; 参数 arg 为调用程序的参数。

【返回值】 该函数调用子程序成功, 将返回 0; 否则, 将返回 -1。

【例 16-2】 下面的示例演示了 spawnle() 函数的使用, 使用该函数调用外部子程序, 代码如下。

```

#include <process.h>           //加入进程库
#include <stdio.h>             //加入输入输出库
#include <conio.h>             //加入控制台输入输出库
void main( ){
    int result;
    clrscr( );                //清屏
    result = spawnle(P_WAIT, "tcc.exe", NULL, NULL); //调用外部子程序
    if (result == -1) {        //根据返回结果输出提示信息
        perror("Error from spawnle");
        exit(1);
    }
}

```

运行上述代码, 将采用 spawnle() 函数调用外部子程序, 并根据调用返回结果输出相应的提示信息。


```

        if(2) longjmp(mark, 2);    //判断程序执行中, 是否出现错误,
如果有错误, 则跳转
        if(-1) longjmp(mark, -1); //判断程序执行中, 是否出现错误,
如果有错误, 则跳转
    }else{
        switch (jmpret)    {    //错误处理模块
            case 1:
                printf( "Error 1\n");
                break;
            case 2:
                printf( "Error 2\n");
                break;
            case 3:
                printf( "Error 3\n");
                break;
            default :
                printf( "Unknown Error");
                break;
        }
        exit(0);
    }
}

```

运行上述程序, 首先保存当前环境变量, 然后根据程序执行情况, 判断是否出现错误, 有错误时进行程序跳转。

17.2 非局部转移函数: setjmp()

函数 setjmp() 用于设置缓冲区来保存堆栈的内容, 其原型如下。

```
int setjmp(jmp_buf env);
```

【参数】 参数 env 为 jmp_buf 结构的一个变量, 用于保存当前的环境信息。

【返回值】 该函数执行后返回值为 0。

【例 17-2】下面的示例演示了 setjmp() 函数的使用，使用该函数将当前系统堆栈信息保存在 env 变量中，代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <process.h> //加入进程管理函数库
#include <setjmp.h> //加入函数跳转函数库
void subroutine(void);
jmp_buf jumper;
void main() {
    int value;
    value = setjmp(jumper); //保存当前系统堆栈信息
    if (value != 0) {
        printf("Longjmp with value %d\n", value); //输出提示信息
        exit(value);
    }
    printf("About to call subroutine ... \n");
    subroutine(); //调用自定义函数进行跳转
}
void subroutine(void) {
    longjmp(jumper, 1);
}
```

运行上述代码后，首先采用 setjmp() 函数获取当前系统堆栈，并保存在变量 jumper 中；然后根据设置结果输出提示信息；最后程序进行跳转，并根据跳转前的环境信息进行返回。

注意：longjmp() 函数必须在 setjmp() 函数调用之后，而且 longjmp() 函数必须在 setjmp() 函数的作用域之内。

第 18 章 信号定义函数库: signal.h

在程序设计中,有时需要进行信号的相应操作。C 语言专门提供了一个针对信号处理操作的函数库——signal.h,该文件库中存在的常用函数主要包括设置信号对应动作函数和发送信号操作函数。

18.1 设置信号对应动作函数: signal()

函数 signal()用于设置信号所对应的动作,其原型如下。

```
int signal(int sig, sigfun fname);
```

【参数】参数sig为需要设置的信号量;参数fname为信号所对应的动作函数名称。

【返回值】该函数执行成功后将返回所读取的字符。

【例 18-1】下面的示例演示了signal()函数的使用,在程序中动态地设置信号所对应的动作,代码如下。

```
#pragma inline
#include <stdio.h>
#include <signal.h>                                //加入信号定义函数库
void Catcher(int sig, int type, int *reglist){
    printf("Caught it!\n");                          //输出提示信息
    *(reglist + 8) = 3;                              //使用 AX 等于 3
}
void main( ){
    signal(SIGFPE, Catcher);                          //设置信号定义动作函数
    asm    mov    ax,07FFFH                          //赋值
    asm    inc    ax                                  //加 1
    asm    into                                       //激活信号所对应的动作函数
    asm    dec    ax                                  //减 1
```



```
asm into
}
```

运行上述程序，首先定义一个信号处理函数，然后采用汇编语言给 AX 赋最大值，再加 1 导致 AX 寄存器溢出，进而触发前面所定义的信号处理函数。该信号处理函数将 AX 寄存器赋值为 3，再进行其他操作时就不会出现溢出等问题。

18.2 发送信号函数：raise()

函数 raise() 用于发送需要处理的信号，其原型如下。

```
int raise(int sig);
```

【参数】 参数 sig 为需要设置的信号量。

【返回值】 该函数执行成功后返回 true；否则，返回 false。

【例 18-2】 下面的示例演示了 raise() 函数的使用，采用该函数判断除数是否为 0，其代码如下。

```
#include <signal.h> //加入信号定义函数库
void main() {
    int a, b; //声明变量
    a = 10; //赋值
    b = 0; //赋值
    if (b == 0)
        raise(SIGFPE); //发送信号
    a = a / b; //除法计算
}
```

运行上述代码，首先声明除数和被除数两个变量，然后对其赋值，再检查除数是否为 0。若为 0，则发出信号，程序将根据所设置的信号对应动作函数进行相应的处理。

第 19 章 函数参数处理函数库:

stdarg.h

在程序设计中,常常需要采用可变参数进行相应的操作。C 语言专门提供了一个针对函数参数处理的函数库——stdarg.h,该文件库中存在的常用函数主要包括格式化输入输出、可变参数开关等函数参数处理函数。

19.1 格式化输出流函数: vfprintf()

函数 vfprintf() 用于实现格式化输出到流中,其原型如下。

```
int vfprintf(FILE *stream, char *format, va_list param);
```

【参数】 参数*stream为要从中读取字符的文件流;参数format为输出的格式;参数param为处理参数。

【返回值】 若该函数执行成功,则返回实际输出的字符数;若失败,则返回-1,错误原因存于errno中。

【例 19-1】 下面的示例演示了vfprintf()函数的使用,采用该函数格式化输出到流,代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <stdlib.h> //加入标准工具库
#include <stdarg.h> //加入函数参数处理函数库
FILE *fp;
int vfprf(char *fmt, ...) { //自定义函数
    va_list argptr;
    int cnt;
    va_start(argptr, fmt); //开始使用可变参数
```

```

    cnt = vfprintf(fp, fmt, argptr);           //输出到流
    va_end(argptr);                           //停止使用可变参数
    return(cnt);
}

void main( ){
    int inumber = 30;
    float fnumber = 90.0;
    char string[4] = "abc";
    fp = tmpfile( );                          //以二进制打开暂存文件
    if (fp == NULL) {                          //判断结果
        perror("tmpfile( ) call");
        exit(1);
    }
    vfprintf(fp, "%d %f %s", inumber, fnumber, string); //调用自定义函数
    rewind(fp);                                //将文件指针重新指向流开头
    fscanf(fp, "%d %f %s", &inumber, &fnumber, string);
                                                //从流中执行格式化输入
    printf("%d %f %s\n", inumber, fnumber, string); //输出
    fclose(fp);
}

```

运行上述程序，首先自定义一个格式化输出的函数；然后以二进制打开暂存文件，并根据打开结果输出提示信息；接着调用自定义函数输出到流，将文件指针重定向到流开头，再从流中执行格式输入，并将输入结果输出到控制台。

19.2 格式化输出函数：vprintf()

函数 vprintf() 用于实现格式化输出到 stdout 中，其原型如下。

```
int vprintf(char *format, va_list param);
```

【参数】 参数 format 为输出的格式；参数 param 为处理参数。

【返回值】 该函数执行后，若输出成功返回 true；否则返回 NULL (0)。

【例 19-2】 下面的示例演示了 vprintf() 函数的使用，使用该函数格式

化输出stdout中, 代码如下。

```
#include <stdio.h>                                     //加入标准输入输出库
#include <stdarg.h>                                     //加入函数参数处理函数库
int vpf(char *fmt, ...) {                               //声明自定义函数
    va_list argptr;
    int cnt;
    va_start(argptr, fmt);                             //开始使用可变参数
    cnt = vprintf(fmt, argptr);                         //格式化输出到流
    va_end(argptr);                                     //结束使用可变参数
    return(cnt);
}
void main( ){
    int inumber = 30;
    float fnumber = 90.0;
    char *string = "abc";
    vpf("%d %f %s\n", inumber, fnumber, string); //调用自定义参数
}
```

运行上述代码, 首先声明自定义函数, 然后在主程序中调用该自定义函数格式输出到 stdout 中。

19.3 格式化输出串函数: vsprintf()

函数 vsprintf()用于格式化输出, 其原型如下。

```
int vsprintf(char *string, char *format, va_list param);
```

【参数】参数string为要输出的缓冲区; 参数format为输出的格式; 参数param为处理的参数。

【返回值】若该函数执行成功, 则返回实际输出的字符数; 若失败, 则返回-1, 错误原因存于errno中。

【例 19-3】下面的示例演示了vsprintf()函数的使用, 在程序中采用该函数进行格式化输出, 代码如下。

```

#include <stdio.h> //加入标准输入输出库
#include <conio.h> //加入标准工具库
#include <stdarg.h> //加入函数参数处理函数库
char buffer[80];
int vspf(char *fmt, ...) { //声明自定义函数
    va_list argptr;
    int cnt;
    va_start(argptr, fmt); //开始使用可变参数
    cnt = vsprintf(buffer, fmt, argptr); //格式化输出
    va_end(argptr); //结束使用可变参数
    return(cnt);
}
void main( ){
    int inumber = 30;
    float fnumber = 90.0;
    char string[4] = "abc";
    vspf("%d %f %s", inumber, fnumber, string); //调用自定义函数
    printf("%s\n", buffer); //输出缓冲区
}

```

运行上述代码，首先声明用于格式化输出的自定义函数，然后在主程序中调用该自定义函数进行格式化输出，将其结果输出至缓冲区，再将缓冲区输出至控制台。

19.4 执行格式化输入函数：vscanf()

函数 vscanf() 用于实现格式化输入，其原型如下。

```
int vscanf(char *format, va_list param);
```

【参数】 参数 format 为要输入的格式；参数 param 为要处理的参数。

【返回值】 若该函数执行成功，则返回实际输入的字符；若失败，则返回 -1，错误原因存于 errno 中。

【例 19-4】 下面的示例演示了 vscanf() 函数的使用，采用该函数执行

格式化输入, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <conio.h> //加入标准工具库
#include <stdarg.h> //加入函数参数处理函数库
int vscanf(char *fmt, ...) { //声明自定义函数
    va_list argptr;
    int cnt;
    printf("Enter an integer, a float, and a string (e.g.
i,f,s,)\n"); //输出提示信息
    va_start(argptr, fmt); //开始使用可变参数
    cnt = vscanf(fmt, argptr); //格式化输入
    va_end(argptr); //结束使用可变参数
    return(cnt);
}
void main( ){
    int inumber;
    float fnumber;
    char string[80];
    vscanf("%d, %f, %s", &inumber, &fnumber, string); //调用自定义函数
    printf("%d %f %s\n", inumber, fnumber, string); //输出
}
```

运行上述程序, 首先声明自定义函数, 在该自定义函数中采用可变参数进行格式化输入, 然后在主程序中调用该函数执行格式化输入, 并将输入的内容输出至控制台。

19.5 执行流中格式化输入函数: vsscanf()

函数 vsscanf() 执行流中格式化输入, 其原型如下。

```
int vsscanf(char *s, char *format, va_list param);
```

【参数】 参数s为执行的流; 参数format为输入的格式; 参数param为要处理的参数。

【返回值】若该函数执行成功，则返回实际输入的字符；若失败，则返回-1，错误原因存于errno中。

【例 19-5】下面的示例演示了vsscanf()函数的使用，在程序中采用该函数执行流中格式化输入，代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <conio.h> //加入标准工具库
#include <stdarg.h> //加入函数参数处理函数库
char buffer[80] = "30 90.0 abc";
int vssf(char *fmt, ...) { //声明自定义函数
    va_list argptr;
    int cnt;
    fflush(stdin); //清除流
    va_start(argptr, fmt); //开始使用变参
    cnt = vsscanf(buffer, fmt, argptr); //格式化输入
    va_end(argptr); //结束使用变参
    return(cnt);
}
void main() {
    int inumber;
    float fnumber;
    char string[80];
    vssf("%d %f %s", &inumber, &fnumber, string); //调用自定义函数
    printf("%d %f %s\n", inumber, fnumber, string); //输出
}
```

运行上述程序，首先声明自定义函数，在该函数中进行格式化输入，然后在主程序中调用该自定义函数进行格式化输入，并将所输入的内容输出至控制台。

19.6 执行流中格式化输入函数：vfscanf()

函数vfscanf()用于实现从流中执行格式输入，其原型如下。

```
int vfscanf(FILE *stream, char *format, va_list param);
```

【参数】参数stream为要输入的流;参数format为输入格式;参数param为处理参数。

【返回值】若该函数执行成功,则返回实际输入的字符;若失败,则返回-1,错误原因存于errno中。

【例 19-6】下面的示例演示了vfscanf()函数的使用,在程序中采用该函数执行流中格式化输入,代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <stdlib.h>          //加入标准工具库
#include <stdarg.h>          //加入函数参数处理函数库
FILE *fp;
int vvfscanf(char *fmt, ...) { //声明自定义变量
    va_list argptr;
    int cnt;
    va_start(argptr, fmt);    //开始使用可变参数
    cnt = vfscanf(fp, fmt, argptr); //执行流中格式化输入
    va_end(argptr);           //结束使用可变参数
    return(cnt);
}
void main( ){
    int inumber = 30;
    float fnumber = 90.0;
    char string[4] = "abc";
    fp = tmpfile( );          //打开暂存文件
    if (fp == NULL) {         //判断结果
        perror("tmpfile( ) call");
        exit(1);
    }
    fprintf(fp, "%d %f %s\n", inumber, fnumber, string); //格式化输出
    rewind(fp);          //将指针重定向到文件头
    vvfscanf("%d %f %s", &inumber, &fnumber, string); //调用自定义函数
    printf("%d %f %s\n", inumber, fnumber, string); //输出
    fclose(fp);
}
```

运行上述程序,首先声明自定义函数,在该函数执行流中格式化输入;然后在主程序中打开暂存文件,并判断打开结果,进行格式化

输出后将指针重定向到文件头；再调用该自定义函数进行格式输入，并输出所输入的内容；最后关闭所打开的文件。

19.7 开始使用可变参数函数：va_start()

函数 va_start() 用于开始使用可变参数，其原型如下。

```
void va_start ( va_list ap, prev_param );
```

【参数】 参数 ap 表示参数自身；参数 prev_param 为第一个参数。

【返回值】 该函数没有返回值。

【例 19-7】 下面的示例演示了 va_start() 函数的使用，在程序中采用该函数开始使用可变参数，代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <stdarg.h> //加入函数参数处理函数库
void simple_va_fun(int start, ...) { //自定义函数
    va_list arg_ptr;
    int nArgValue = start;
    int nArgCout=0; //可变参数的数目
    va_start(arg_ptr, start); //以固定参数的地址为起点确定变参的内存起始地址
    do {
        ++nArgCout;
        printf("the %d th arg: %d\n", nArgCout, nArgValue); //输出各参数的值
        nArgValue = va_arg(arg_ptr, int); //得到下一个可变参数的值
    } while(nArgValue != -1);
    return;
}
int main(int argc, char* argv[ ]){
    simple_va_fun(100, -1); //以两个参数调用
    simple_va_fun(100, 200, -1); //以三个参数调用
}
```


运行上述程序, 首先声明自定义函数, 在该自定义函数中开始启用可变参数, 循环获取每一个可变参数的值, 然后在主程序中以可变参数的形式调用该自定义函数。

19.8 停止使用可变参数函数: va_end()

函数 va_end() 用于停止使用可变参数, 其原型如下。

```
void va_end ( va_list ap );
```

【参数】 参数 ap 表示参数自身。

【返回值】 该函数没有返回值。

【例 19-8】 下面的示例演示了 va_end() 函数的使用, 在程序中采用该函数停止使用可变参数, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
#include <stdarg.h>                                //加入函数参数处理函数库
void simple_va_fun(int i, ...){
    va_list arg_ptr;
    char *s=NULL;
    va_start(arg_ptr, i); //以固定参数的地址为起点确定变参的内存起始地址
    s=va_arg(arg_ptr, char*); //得到下一个可变参数的值
    va_end(arg_ptr);
    printf("%d %s\n", i, s); //输出各参数的值
    return;
}
int main(int argc, char* argv[ ]){
    simple_va_fun(100,-1); //以两个参数调用
    simple_va_fun(100,200,-1); //以三个参数调用
}
```

运行上述程序, 首先声明自定义函数, 在该自定义函数中开始启用可变参数, 循环获取每一个可变参数的值, 然后在主程序中以可变参数的形式调用该自定义函数。

19.9 调用可变参数列表函数: va_arg()

函数 va_arg() 用于调用可变参数列表, 其原型如下。

```
type va_arg ( va_list ap, type );
```

【参数】 参数ap为可变参数自身; 参数type是要获取的参数的指定类型, 然后返回这个指定类型的值, 并且把ap的位置指向变参表的下一个变量位置。

【返回值】 该函数没有返回值。

【例 19-9】 下面的示例演示了va_arg()函数的使用, 在程序中采用该函数调用可变参数列表, 代码如下。

```
#include <stdio.h>                                //加入标准输入输出库
#include <stdarg.h>                                //加入函数参数处理函数库
void va_arg_fun(int i, ...){
    va_list arg_ptr;
    char *s=NULL;
    va_start(arg_ptr, i);
                                //以固定参数的地址为起点确定变参的内存起始地址
    s=va_arg(arg_ptr, char*); //得到下一个可变参数的值
    va_end(arg_ptr);
    printf("%d %s\n", i, s); //输出各参数的值
    return;
}
int main(int argc, char* argv[ ]){
    va_arg_fun(30,-1);                                //以两个参数调用
    va_arg_fun(60,200,-1);                            //以三个参数调用
    va_arg_fun(100,300,400,-1);                       //以四个参数调用
}
```

运行上述程序, 首先声明自定义函数, 在该自定义函数中开始启用可变参数, 循环获取每一个可变参数的值, 然后在主程序中以可变参数的形式调用该自定义函数。

第 20 章 时间函数库: time.h

在程序设计中,常常需要进行时间方面的操作。C 语言专门提供了一个针对时间操作的函数库——time.h,该文件库中存在的常用函数主要包括系统日期和时间的获取、设置等操作函数。

20.1 获取 tm 结构的系统时间函数: localtime()

函数 localtime()用于获取时间结构体格式的系统时间,其原型如下。

```
struct tm *localtime(long *clock);
```

【参数】 参数clock为需要转换的时间。

【返回值】 该函数执行成功后将返回tm结构的系统时间。

【例 20-1】 下面的示例演示了localtime()函数的使用,采用该函数获取tm结构的系统时间,代码如下。

```
#include <time.h>           //加入时间库
#include <stdio.h>          //加入输入输出库
#include <dos.h>             //加入系统接口库
void main( ){
    time_t timer;           //声明变量
    struct tm *tblock;      //声明用于保存 tm 时间的结构体
    timer = time(NULL);     //取系统时间
    tblock = localtime(&timer); //将系统时间转换为 tm 结构
    printf("Local time is: %s", asctime(tblock)); //输出 tm 结构时间
}
```

运行上述程序,首先获取系统时间,然后将该系统时间转换为 tm 结构的时间,再通过输入输出函数将其输出。

20.2 获取格林尼治结构的时间函数: gmtime()

函数 gmtime() 将日期和时间转换为格林尼治标准时间, 其原型如下。

```
struct tm *gmtime(long *clock);
```

【参数】 参数 clock 为系统时间。

【返回值】 该函数执行后将返回一个格式尼治格式的时间。

【例 20-2】 下面的示例演示了 gmtime() 函数的使用, 使用该函数将给定时间转换为格林尼治标准时间, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <stdlib.h>          //加入标准工具库
#include <time.h>            //加入时间库
#include <dos.h>             //加入系统接口库
char *tzstr = "TZ=PST8PDT"; //指定时区
void main( ){
    time_t t;                //声明变量
    struct tm *gmt, *area;    //声明格林尼治标准时间结构体
    putenv(tzstr);           //设置环境变量
    tzset( );                //时区设置
    t = time(NULL);          //获取系统时间
    area = localtime(&t);     //转换为 tm 结构时间
    printf("Local time is: %s", asctime(area)); //输出
    gmt = gmtime(&t);         //转换为格林尼治时间
    printf("GMT is: %s", asctime(gmt)); //输出
}
```

运行上述代码, 首先获取系统时间, 然后将其转换为 tm 结构时间, 最后转换为格林尼治标准时间, 并将其输出。

20.3 获取系统时间函数: asctime()

函数 asctime() 将给定的日期和时间转换为 ASCII 码, 其原型如下。

```
char *asctime(const struct tm *tblock);
```

【参数】 参数tblock为要进行转换的tm结构时间。

【返回值】 该函数将给定时间转换为ASCII码并返回。

【例 20-3】 下面的示例演示了asctime()函数的使用, 采用该函数将系统时间转换为ASCII码, 代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <string.h> //加入字符串库
#include <time.h> //加入时间库
void main( ){
    struct tm t; //声明结构体变量
    char str[80]; //声明数组变量
    t.tm_sec = 1; //设置秒
    t.tm_min = 30; //设置分
    t.tm_hour = 9; //设置小时
    t.tm_mday = 22; //设置日期
    t.tm_mon = 11; //设置月份
    t.tm_year = 56; //设置年
    t.tm_wday = 4; //设置星期
    t.tm_yday = 0; //不显示ASCII码
    t.tm_isdst = 0;
    strcpy(str, asctime(&t)); //转换ASCII码
    printf("%s\n", str); //输出
}
```

运行上述代码, 首先声明时间结构体变量, 然后给该tm结构体变量赋值, 最后将该结构转换为ASCII码的时间并输出到控制台。

20.4 计算时间差函数: difftime()

函数difftime()用于计算两个时刻之间的时间差, 其原型如下。

```
double difftime(time_t time2, time_t time1);
```

【参数】 参数time1和time2为需要进行比较的时间。

【返回值】该函数对两个时间值进行比较,返回两个时间之间的差异。

【例 20-4】下面的示例演示了 `difftime()` 函数的使用,采用该函数计算两个时刻之间的时间差,代码如下。

```
#include <time.h>           //加入时间函数库
#include <stdio.h>           //加入标准输入输出库
#include <dos.h>             //加入系统接口函数库
#include <conio.h>           //加入控制台输入输出库
#include <system.h>          //加入标准库

void main( ){
    time_t first, second;    //声明 tm 格式的时间变量
    first = time(NULL);      //获取系统时间
    delay(2000);             //等 2 秒钟
    second = time(NULL);     //获取系统时间
    printf("The difference is: %f \seconds\n",difftime(second,first));
                                //计算并输出两个时间之差
}
```

运行上述程序,首先声明两个用于保存时间的 `tm` 结构体变量,然后获取系统时间,并计算这两个时间之间的时间差。

20.5 转换日期时间字符串函数: `ctime()`

函数 `ctime()` 用于把日期和时间转换为字符串,其原型如下。

```
char *ctime(const time_t *time);
```

【参数】参数 `time` 为要转换的日期时间。

【返回值】该函数将给定的日期转换为字符串并返回。

【例 20-5】下面的示例演示了 `ctime()` 函数的使用,采用该函数将给定的日期时间转换为字符串,代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <time.h>             //加入时间库

void main( ){
```



```

time_t t;                //声明暂存时间的变量
time(&t);                //获取系统时间
printf("Today's date and time: %s\n", ctime(&t));
                        //将系统时间转换并输出为字符串
}

```

运行上述程序, 首先声明用于暂存时间的结构体变量, 获取系统时间, 再将系统转换为字符串并输出到控制台。

20.6 获取或设置时间函数: time()

函数 time() 用于获取或设置系统时间, 其原型如下。

```
long time(long *tloc);
```

【参数】 若给定参数 tloc, 则将当前时间保存到该参数中。

【返回值】 该函数返回当前时间, 若出错则返回 0。

【例 20-6】 下面的示例演示了 time() 函数的使用, 使用该函数获取当前系统时间, 代码如下。

```

#include <time.h>          //加入时间库
#include <stdio.h>          //加入标准输入输出库
#include <dos.h>            //加入系统接口库
void main() {
    time_t t;              //声明变量
    t = time(NULL);        //获取时间
    printf("The number of seconds since January 1, 1970 is %ld", t);
                        //输出
}

```

运行上述程序, 首先声明用于保存系统时间的变量, 然后使用 time() 函数获取当前系统时间, 并以特定的格式输出到控制台。

20.7 UNIX 时间兼容函数: tzset()

函数 tzset() 用于实现将 UNIX 时间兼容, 其原型如下。

```
void tzset(void);
```

【参数】 该函数没有参数。

【返回值】 该函数没有返回值。

【例 20-7】 下面的示例演示了 tzset() 函数的使用, 代码如下。

```
#include <time.h>           //加入时间库
#include <stdlib.h>          //加入标准工具库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    time_t td;               //声明变量
    putenv("TZ=PST8PDT");    //设置环境变量
    tzset( );                //设置 UNIX 时间兼容
    time(&td);                //获取系统时间
    printf("Current time = %s\n", asctime(localtime(&td)));
    //输出
}
```

运行上述程序, 首先声明用于暂存时间的变量, 然后设置时区和 UNIX 时间兼容, 再获取当前系统时间再输出到控制台。

第 21 章 标准工具库函数库: stdlib.h

在程序设计中,常常需要用到一些特殊的操作。C 语言专门提供了一个针对标准工具的函数库——stdlib.h,该文件库中存在的常用函数主要包括内存的申请与释放、随机数的操作、字符串的转换等函数。

21.1 分配主存储器函数: calloc()

函数 calloc() 用于分配主存储器,其原型如下。

```
void *calloc(size_t nelem, size_t elsize);
```

【参数】 参数 nelem 为需要分配的长度; 参数 elsize 为分配的类型。

【返回值】 该函数分配指定长度的内存,分配后没有返回值。

【例 21-1】 下面的示例演示了 calloc() 函数的使用,代码如下。

```
#include <stdio.h> //加入标准输入输出库
#include <alloc.h> //加入内存管理库
#include <stdlib.h> //加入工具库
void main() {
    char *str = NULL; //声明变量
    str = calloc(10, sizeof(char)); //分配内存
    strcpy(str, "Hello"); //复制字符串到该内存
    printf("String is %s\n", str); //输出
    free(str); //释放内存
}
```

运行上述程序,首先声明一个字符串类型的指针;然后分配指定长度的内存,将指定的字符串复制到该内存,并输出到控制台。

21.2 释放已分配块函数: free()

函数 free() 用于释放已经分配的内存块, 其原型如下。

```
void free(void *ptr);
```

【参数】 参数 ptr 为已分配的内存块。

【返回值】 该函数没有返回值。

【例 21-2】 下面的示例演示了 free() 函数的使用, 使用该函数释放已申请的内存空间, 代码如下。

```
#include <string.h>           //加入字符串库
#include <stdio.h>           //加入标准输入输出库
#include <alloc.h>           //加入内存管理库
#include <stdlib.h>          //加入工具库
void main( ){
    char *str;
    str = malloc(10);         //为字符串分配内存
    strcpy(str, "Hello");     //复制字符串到该位置
    printf("String is %s\n", str); //输出
    free(str);               //释放内存
}
```

运行上述代码, 首先声明字符串指针; 然后为字符串申请内存, 再将指定字符串复制到该内存并输出到控制台; 最后释放申请的内存空间。

21.3 内存分配函数: malloc()

函数 malloc() 用于分配指定大小的内存空间, 其原型如下。

```
void *malloc(unsigned size);
```

【参数】 参数 size 为要申请的空间长度。

【返回值】该函数没有返回值。

【例 21-3】下面的示例演示了 malloc() 函数的使用, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <string.h>          //加入字符串库
#include <alloc.h>           //加入内存管理库
#include <process.h>         //进程管理库
#include <stdlib.h>          //加入工具库

void main( ){
    char *str;                //声明变量
    if ((str = malloc(10)) == NULL) { //分配内存
        printf("Not enough memory to allocate buffer\n");
                                                //若内存不够, 输出提示信息
        exit(1);                    //退出
    }
    strcpy(str, "Hello");      //将字符串复制到分配的内存
    printf("String is %s\n", str); //输出
    free(str);                //释放内存空间
}
```

运行上述代码, 首先声明变量, 分配指定大小的内存空间, 若空间不够, 输出提示信息并退出程序。若分配成功, 将字符串复制到该内存区域, 输出该区域内容后释放申请的内存空间。

21.4 重新分配主存函数: realloc()

函数 realloc() 用于重新申请指定大小的内存空间, 其原型如下。

```
void *realloc(void *ptr, unsigned newsize);
```

【参数】参数 ptr 为指向内存区的指针; 参数 newsize 为重新分配内存的大小。

【返回值】该函数将给前面分配的内存区重新分配指定大小的内存并返回。

【例 21-4】下面的示例演示了realloc()函数的使用,使用该函数重新分配内存,代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <alloc.h>           //加入内存管理库
#include <string.h>          //加入字符串库
#include <stdlib.h>          //加入工具库

void main( ){
    char *str;                //声明变量
    str = malloc(10);         //分配内存
    strcpy(str, "Hello");     //复制字符串
    printf("String is %s\n Address is %p\n", str, str); //输出
    str = realloc(str, 20);   //重新分配
    printf("String is %s\n New address is %p\n", str, str); //输出
    free(str);               //释放
}
```

运行上述程序,首先分配内存空间,将字符串复制到该内存区;然后重新分配内存,并输出该内存区内容;最后释放所申请的内存空间。

21.5 随机数发生器函数: rand()

函数rand()用于生成随机数,其原型如下。

```
void rand(void);
```

【参数】该函数没有参数。

【返回值】该函数生成一个随机数并返回。

【例 21-5】下面的示例演示了rand()函数的使用,在程序中采用该函数生成随机数,代码如下。

```
#include <stdlib.h>          //加入工具库
#include <stdio.h>          //加入标准输入输出库

void main( ){
    int i;
    printf("Ten random numbers from 0 to 99\n\n"); //输出提示信息
```



```

    for(i=0; i<10; i++)                //循环输出随机数
        printf("%d\n", rand() % 100);
}

```

运行上述程序,首先在控制台输出提示信息,然后在控制台循环输出 10 个随机数。

21.6 随机数发生器函数: random()

函数 random() 按给定的最大值生成随机数,其原型如下。

```
int random(int num);
```

【参数】 参数 num 为要生成的随机数的最大值。

【返回值】 该函数生成并返回一个不大于指定大小的随机数。

【例 21-6】 下面的示例演示了 random() 函数的使用,使用该函数生成指定大小的随机数,代码如下。

```

#include <stdlib.h>                //加入工具库
#include <stdio.h>                //加入标准输入输出库
#include <time.h>                //加入时间库

void main() {
    randomize();                //初始化
    printf("Random number in the 0~99 range: %d\n", random(100));
    //生成 0~99 之间的随机数,并输出到控制台
}

```

运行上述程序,首先初始化随机数发生器,然后生成一个指定大小的随机数,并输出到控制台。

21.7 初始化随机数发生器函数: randomize()

函数 randomize() 用于初始化随机数发生器,其原型如下。

```
void randomize(void);
```

【参数】该函数没有参数。

【返回值】该函数没有返回值。

【例 21-7】下面的示例演示了randomize()函数的使用，在程序中采用该函数初始化随机数发生器，代码如下。

```
#include <stdlib.h>           //加入工具库
#include <stdio.h>           //加入标准输入输出库
#include <time.h>             //加入时间库
void main( ){
    int i; randomize( );      //初始化随机数发生器
    printf("Ten random numbers from 0 to 99\n\n"); //输出提示信息
    for(i=0; i<10; i++)      //循环输出
        printf("%d\n", rand( ) % 100);
}
```

运行上述程序，首先初始化随机数发生器，然后循环并输出 10 个随机数。

21.8 初始化随机数发生器函数：srand()

函数 srand()用于初始化随机数发生器，其原型如下。

```
void srand(unsigned seed);
```

【参数】参数 seed 必须是一个整数，用于设置随机数的种子。

【返回值】该函数用于设置 rand()函数产生随机数的种子，没有返回值。

【例 21-8】下面的示例演示了srand()函数的使用，采用该函数初始化随机数发生器，代码如下。

```
#include <stdlib.h>           //加入工具库
#include <stdio.h>           //加入标准输入输出库
#include <time.h>             //加入时间库
```

```

int main(void) {
    int i;    time_t t;
    srand((unsigned) time(&t));           //初始化随机数发生器
    printf("Ten random numbers from 0 to 99\n\n"); //输出提示信息
    for(i=0; i<10; i++)                  //循环输出随机数
        printf("%d\n", rand() % 100);
    return 0;
}

```

运行上述程序, 首先使用当前时间作为随机数的种子进行初始化, 然后循环并输出 10 个随机数到控制台。

21.9 异常终止进程函数: abort()

函数 abort() 用于实现异常终止进程, 其原型如下。

```
void abort(void);
```

【参数】 该函数没有参数。

【返回值】 该函数没有返回值。

【例 21-9】 下面的示例演示了 abort() 函数的使用, 采用该函数终止进程, 代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <stdlib.h>          //加入工具库
void main() {
    printf("Calling abort() \n"); //输出提示信息
    abort();                  //终止程序
                                //这一行则不会运行
}

```

运行上述程序, 首先在控制台输出提示信息, 然后异常终止进程, 此时最后一行永远不会执行。

21.10 终止程序函数: exit()

函数 exit() 用于终止程序, 其原型如下。

```
void exit(int status);
```

【参数】 参数 status 为终止状态。

【返回值】 该函数直接终止程序, 没有返回值。

【例 21-10】 下面的示例演示了 exit() 函数的使用, 在程序中使用该函数终止程序, 代码如下。

```
#include <stdlib.h>           //加入工具库
#include <conio.h>             //加控制台输入输出库
#include <stdio.h>             //加入标准输入输出库
void main( ){
    int status;
    printf("Enter either 1 or 2\n"); //输出提示信息
    status = getch( );             //获取字符
    exit(status - '0');            //终止程序
                                   //程序已终止, 这一行永远不会执行
}
```

运行上述程序, 输出提示信息, 获取控制台输入, 然后终止程序。因程序已终止, 其最后一行将永远不会执行。

21.11 获取环境中字符串函数: getenv()

函数 getenv() 用于获取当前环境中的字符串, 其原型如下。

```
char *getenv(char *envvar);
```

【参数】 参数 envvar 为环境变量名。

【返回值】 该函数返回一个给定的环境变量值, 环境变量名可大写或小写。如果指定的变量在环境中未定义, 则返回一个空串。

【例 21-11】下面的示例演示了 `getenv()` 函数的使用, 使用该函数获取当前环境中未使用内存的大小, 代码如下。

```
#include <stdlib.h>           //加入工具库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    char *s;
    s=getenv("COMSPEC");      //获取环境变量
    printf("Command processor: %s\n",s); //输出
}
```

运行上述代码, 首先获取环境变量 `COMSPEC` 的值, 若该变量并未定义, 则输出一个空串。

21.12 加载环境中字符串函数: `putenv()`

函数 `putenv()` 用于改变或增加环境变量的内容, 其原型如下。

```
int putenv(char *envvar);
```

【参数】参数 `envvar` 为需要设置的环境变量, 若该值存在, 则会替换原内容, 否则以新值作为环境变量。

【返回值】若该函数执行成功, 则返回 0; 若有错误发生, 则返回 -1。

【例 21-12】下面的示例演示了 `putenv()` 函数的使用, 采用该函数设置环境变量, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <stdlib.h>          //加入工具库
#include <alloc.h>           //加入内存管理库
#include <string.h>          //加入字符串库
#include <dos.h>             //加入系统接口库
void main( ){
    char *path, *ptr;
    int i = 0;
    ptr = getenv("PATH");    //取环境变量
    path = malloc(strlen(ptr)+15); //申请内存
```

```

strcpy(path, "PATH=");           //复制字符串
strcat(path, ptr);                //拼接
strcat(path, ";c:\\temp");        //拼接
putenv(path);                     //设置环境变量
while (environ[i])                //输出
    printf("%s\n", environ[i++]);
}

```

运行上述代码，首先获取环境变量；然后对所获取的环境变量拼接字符串，再将拼接后的环境变量替换原来的环境变量；最后输出设置的环境变量。

21.13 转换字符串函数：atof()

函数 `atof()` 用于把字符串转换成浮点数，其原型如下。

```
double atof(const char *nptr);
```

【参数】 参数 `nptr` 为要转换的字符串。

【返回值】 该函数会扫描参数 `nptr` 字符串，跳过前面的空格字符，直到遇上数字或正负符号才开始做转换，而再遇到非数字或字符串结束时（'\0'）才结束转换，并将结果返回，返回转换后的浮点型数。

【例 21-13】 下面的示例演示了 `atof()` 函数的使用，采用该函数将给定的字符串转换为浮点数，代码如下。

```

#include <stdlib.h>                //加入工具库
#include <stdio.h>                 //加入标准输入输出库
void main( ){
    float f;
    char *str = "12345.67";        //定义要转换的字符串
    f = atof(str);                 //转换为浮点数
    printf("string = %s float = %f\n", str, f); //输出
}

```

运行上述程序，首先声明要转换的字符串，然后将该字符串转换

为浮点数并输出至控制台。

21.14 转换字符串函数: atoi()

函数 `atoi()` 用于把字符串转换成整型数, 其原型如下。

```
int atoi(const char *nptr);
```

【参数】 参数 `nptr` 表示要进行转换的字符。

【返回值】 该函数会扫描参数 `nptr` 字符串, 跳过前面的空格字符, 直到遇上数字或正负符号才开始做转换, 而再遇到非数字或字符串结束时 (`\0`) 才结束转换, 并将结果返回, 返回转换后的整型数。

【例 21-14】 下面的示例演示了 `atoi()` 函数的使用, 采用该函数将给定的字符串转换为整型数, 代码如下。

```
#include <stdlib.h>           //加入工具库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    int n;
    char *str = "12345.67";   //声明变量
    n = atoi(str);            //转换为长整型
    printf("string = %s integer = %d\n", str, n); //输出
}
```

运行上述程序, 首先声明要定义的字符串, 然后将该字符串转换为长整型, 最后将转换后的内容输出到控制台。

21.15 转换字符串函数: atol()

函数 `atol()` 用于把字符串转换为长整型数, 其原型如下。

```
long atol(const char *nptr);
```

【参数】 参数 `nptr` 表示要进行转换的字符。

【返回值】 该函数会扫描参数 `nptr` 字符串，跳过前面的空格字符，直到遇上数字或正负符号才开始做转换，而再遇到非数字或字符串结束时（`'\0'`）才结束转换，并将结果返回，返回转换后的长整型数。

【例 21-15】 下面的示例演示了 `atol()` 函数的使用，采用该函数将指定字符转换为长整型，代码如下。

```
#include <stdlib.h>           //加入工具库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    long l;
    char *str = "98765432";   //声明要转换的变量
    l = atol(str);            //转换为长整型
    printf("string = %s integer = %ld\n", str, l); //输出
}
```

运行上述程序，首先声明要转换的字符串，然后将该字符串转换为长整型，最后将转换后的长整型数输出到控制台。

21.16 转换浮点数函数： `ecvt()`

函数 `ecvt()` 用于把一个浮点数转换为字符串，其原型如下。

```
char ecvt(double value, int ndigits, int *decpt, int *sign);
```

【参数】 参数 `value` 为需要转换的浮点数；参数 `ndigits` 表示显示的位数；参数 `decpt` 指针所指的变量会返回数值中小数点的地址（从左到右算起）；而参数 `sign` 指针所指的变量则代表数值正或负，若数值为正，则返回 0，否则返回 1。

【返回值】 该函数返回字符串指针。

【例 21-16】 下面的示例演示了 `ecvt()` 函数的使用，采用该函数将浮点数转换为字符串，代码如下。

```
#include <stdlib.h>           //加入工具库
```

```

#include <stdio.h> //加入标准输入输出库
#include <conio.h> //加入控制台输入输出库
void main( ){
    char *string;
    double value;
    int dec, sign;
    int ndig = 10;
    value = 9.876;
    string = ecvt(value, ndig, &dec, &sign); //转换
    printf("string = %s      dec = %d \
    sign = %d\n", string, dec, sign); //输出
    value = -123.45;
    ndig= 15;
    string = ecvt(value, ndig, &dec, &sign); //转换
    printf("string = %s dec = %d sign = %d\n",
    string, dec, sign); //输出
}

```

运行上述程序, 首先声明需要进行转换的浮点数变量, 然后将该浮点数转换为字符串并输出到控制台。

21.17 转换浮点数函数: fcvt()

函数 fcvt() 用于把一个浮点数转换为字符串, 其原型如下。

```
char *fcvt(double value, int ndigits, int *decpt, int *sign);
```

【参数】该函数用来将参数 value 转换成 ASCII 码字符串; 参数 ndigits 表示小数点后显示的位数。若转换成功, 参数 decpt 指针所指的变量会返回数值中小数点的地址 (从左到右算起); 而参数 sign 指针所指的变量则代表数值的正或负, 若数值为正, 则该返回值为 0, 否则为 1。

【返回值】该函数返回一个字符串指针。

【例 21-17】下面的示例演示了 fcvt() 函数的使用, 采用该函数将浮

点数转换为字符串，代码如下。

```
#include <stdlib.h> //加入工具库
#include <stdio.h> //加入标准输入输出库
#include <conio.h> //加入控制台输入输出库
void main( ){
    char *string;
    double value;
    int dec, sign;
    int ndig = 10;
    clrscr( ); //清屏
    value = 9.876;
    string = fcvt(value, ndig, &dec, &sign); //转换
    printf("string = %s dec = %d \n", string, dec, sign); //输出
    value = -123.45;
    ndig = 15;
    string = fcvt(value, ndig, &dec, &sign); //转换
    printf("string = %s dec = %d sign = %d\n", string, dec, sign); //输出
}
```

运行上述程序，首先声明需要进行转换的浮点数变量，然后将该变量转换为字符串并输出到控制台。

21.18 转换浮点数函数：gcvt()

函数 gcvt() 用于将浮点型数转换为字符串，取四舍五入，其原型如下。

```
char *gcvt(double value, int ndigits, char *buf);
```

【参数】参数 value 为要转换成的 ASCII 码字符串；参数 ndigits 表示显示的位数；参数 buf 为暂存结果的缓冲区。

【返回值】该函数返回一字符串指针，此地址即为 buf 指针。

【例 21-18】下面的示例演示了 `gcvrt()` 函数的使用, 采用该函数将浮点数转换为字符串, 代码如下。

```
#include <stdlib.h> //加入工具库
#include <stdio.h> //加入标准输入输出库

void main() {
    char str[25];
    double num;
    int sig = 5;
    num = 9.876;

    gcvrt(num, sig, str); //转换
    printf("string = %s\n", str);
    num = -123.4567;

    gcvrt(num, sig, str); //转换
    printf("string = %s\n", str);
    num = 0.678e5;
}
```

运行上述程序, 首先定义需要进行转换的不同类型的浮点数; 然后将这些浮点数转换为字符串, 并输出至控制台。

21.19 转换整数函数: `itoa()`

函数 `itoa()` 用于把一整数转换为字符串, 其原型如下。

```
char *itoa(int value, char *string, int radix);
```

【参数】参数 `value` 为需要转换成字符的数字; 参数 `string` 为转换后保存字符的变量; 参数 `radix` 为转换数字的基数 (进制)。

【返回值】该函数返回指向转换后的字符串的指针。

【例 21-19】下面的示例演示了 `itoa()` 函数的使用, 采用该函数将整数转换为字符串, 代码如下。

```
#include <stdlib.h> //加入工具库
#include <stdio.h> //加入标准输入输出库
```

```

void main( ){
    int number = 12345;
    char string[25];
    itoa(number, string, 10);           //转换
    printf("integer = %d string = %s\n", number, string); //输出
}

```

运行上述程序，首先声明需要进行转换的整数；然后将该整数转换为字符串，并输出到控制台。

21.20 转换字符串函数：strtod()

函数 strtod()用于将字符串转换成浮点数，其原型如下。

```
double strtod(char *str, char **endptr);
```

【参数】函数会扫描参数 str 字符串，跳过前面的空格字符，直到遇上数字或正负符号才开始做转换，出现非数字或字符串结束时 ('\0') 才结束转换，并将结果返回。若 endptr 不为 NULL，则会将遇到不合条件而终止的 nptr 中的字符指针由 endptr 返回。

【返回值】该函数返回转换后的浮点型数。

【例 21-20】下面的示例演示了 strtod() 函数的使用，使用该函数将字符串转换成浮点数，代码如下。

```

#include <stdio.h>           //加入标准输入输出库
#include <stdlib.h>          //加入工具库
void main( ){
    char input[80], *endptr;
    double value;
    printf("Enter a floating point number:"); //输出提示信息
    gets(input);
    value = strtod(input, &endptr);           //转换
    printf("The string is %s the number is %lf\n", input, value);
                                           //输出
}

```


运行上述代码, 首先声明需要进行转换的变量, 输出提示信息, 然后将变量转换为浮点数并输出到控制台。

21.21 转换字符串函数: strtol()

函数 `strtol()` 用于将字符串转换为长整数, 其原型如下。

```
long strtol(char *str, char **endptr, int base);
```

【参数】 函数会将参数 `str` 字符串根据参数 `base` 来转换成长整型数。若参数 `endptr` 不为 `NULL`, 则会将遇到不合条件而终止的 `nptr` 中的字符指针由 `endptr` 返回。

【返回值】 返回转换后的长整型数, 否则返回 `ERANGE` 并将错误代码存入 `errno` 中。

【例 21-21】 下面的示例演示了 `strtol()` 函数的使用, 将指定的字符串转换为长整型数, 其代码如下。

```
#include <stdlib.h>           //加入工具库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    char *string = "87654321", *endptr;
    long lnumber;
    lnumber = strtol(string, &endptr, 10); //转换
    printf("string = %s long = %ld\n", string, lnumber); //输出
}
```

运行上述代码, 首先声明需要转换的字符串; 然后将该字符串转换为长整型数, 并将该长整型数输出到控制台。

21.22 交换字节函数: swab()

函数 `swab()` 将从源和目标区域交换字节, 其原型如下。

```
void swab (char *from, char *to, int nbytes);
```

【参数】参数 from 和 to 为需要交换字节的字符串；参数 nbytes 为需要交换的长度。

【返回值】该函数没有返回值。

【例 21-22】下面的示例演示了 swab() 函数的使用，采用该函数交换指定区域的字节，代码如下。

```
#include <stdlib.h>           //加入工具库
#include <stdio.h>            //加入标准输入输出库
#include <string.h>           //加入字符串库
char source[15] = "rFna koBlrna d"; //声明
char target[15];              //声明
void main( ){
    swab(source, target, strlen(source)); //交换
    printf("This is target: %s\n", target); //输出
}
```

运行上述程序，首先声明两个字符数组，然后采用该函数交换两个字符数组，最后将交换后的数组输出到控制台。

21.23 线性搜索函数：lfind()

函数 lfind() 利用线性搜索在数组中从头至尾一项项查找数据，其原型如下。

```
void *lfind(void *key, void *base, int *nelem, int width, int (*fcmp) ( ));
```

【参数】参数 key 为要查找的关键字；参数 base 为要查找的区域；参数 nelem 为查找后的结果缓冲区；参数 width 为要查找的宽度；参数 fcmp 为是否需要比较。

【返回值】若参数 c 为空格，则返回 true，否则返回 NULL (0)。

【例 21-23】下面的示例演示了 lfind() 函数的使用，在程序中采用该

函数进行线性搜索, 代码如下。

```
#include <stdio.h>           //加入标准输入输出库
#include <stdlib.h>          //加入工具库
int compare(int *x, int *y) { //自定义函数
    return( *x - *y );
}
void main( ){
    int array[5] = {35, 87, 46, 99, 12}; //声明数组
    size_t nelem = 5;
    int key;
    int *result;
    key = 99;
    result = lfind(&key, array, &nelem, sizeof(int), (int (*)(const
void *,const void *))compare);
    if (result) printf("Number %d found\n",key); //输出提示信息
    else      printf("Number %d not found\n",key);
}
```

运行上述程序, 首先声明需要查询的数组; 然后通过使用 `lfind()` 函数进行线性搜索, 并根据搜索结果输出不同的提示信息。

21.24 线性搜索函数: `lsearch()`

函数 `lsearch()` 利用线性搜索在数组中从头至尾一项项查找数据, 其原型如下。

```
void *lsearch(const void *key, void *base, size_t *nelem, size_t
width, int (*compar)(const void *, const void *));
```

【参数】 参数 `key` 指向要查找的关键数据; 参数 `base` 指向要被搜索的数组开头地址; 参数 `nelem` 代表数组中的元素数量, 每一元素的大小则由参数 `size` 决定。最后一项参数 `compar` 为一函数指针, 这个参数用来判断两个元素是否相同。若传给 `compar` 的第一个参数所指的元素数据和第二个参数所指的元素数据相同时则返回 0, 两个元素数

据不相同则返回非零值。如果 lsearch() 函数找不到关键数据时，会主动把该项数据加入数组中。

【返回值】若找到关键数据，则返回找到的该笔元素的四肢；如果在数组中找不到关键数据，则将此关键数据加入数组，再把加入数组后的地址返回。

【例 21-24】下面的示例演示了 lsearch() 函数的使用，采用该函数在程序中执行线性搜索，代码如下。

```
#include<stdio.h>                //加入标准输入输出库
#include<stdlib.h>                //加入工具库
#define NMEMB 50
#define SIZE 10
int compar (const void *a,const void *b) {    //声明自定义变量
    return (strcmp((char *) a, (char *) b));
}
void main( ){
    char
data[NMEMB][SIZE]={ "Linux","freebsd","solzris","sunos","windows"};
                                //声明变量
    char key[80],*base,*offset;
    int i, nmemb=NMEMB,size=SIZE;
    for(i=1;i<5;i++){            //循环搜索
        fgets(key,sizeof(key),stdin);
        key[strlen(key)-1]='\0';
        base = data[0];
        offset = (char *)lfind(key,base,&nmemb,size,compar);
        if(offset ==NULL){
            printf("%s not found!\n",key);
            offset=(char *)
lsearch(key,base,&nmemb,size,compar);
            printf("Add %s to data array\n",offset);
        }else
            printf("found : %s \n",offset);
    }
}
```

运行上述程序，首先声明需要进行搜索的数组变量；然后循环进行线性搜索，并根据搜索结果输出相应的提示信息。

21.25 二分法搜索函数: bsearch()

函数 bsearch() 用于通过二分法实现字符的搜索, 其原型如下。

```
void *bsearch(const void *key, const void *base, size_t *nelem,
size_t width, int(*compar)(const void *, const *));
```

【参数】 参数 base 指向欲排序的数组开头地址; 参数 nelem 代表数组中的元素数量, 每一元素的大小则由参数 size 决定, 最后一项参数 compar 为一函数指针, 这个函数用来判断两个元素间的大小关系。若传给 compar 的第一个参数所指的元素数据大于第二个参数所指的元素数据, 则必须返回大于 0 的值; 若两个元素数据相等, 则返回 0。

【返回值】 该函数没有返回值。

【例 21-25】 下面的示例演示了 bsearch() 函数的使用, 在程序中通过二分法进行字符的搜索, 代码如下。

```
#include <stdlib.h> //加入工具库
#include <stdio.h> //加入标准输入输出库
#define NELEMS(arr) (sizeof(arr) / sizeof(arr[0]))
int numarray[ ] = {123, 145, 512, 627, 800, 933}; //声明变量
int numeric (const int *p1, const int *p2) { //自定义函数
    return(*p1 - *p2);
}
int lookup(int key){
    int *itemptr;
    itemptr = bsearch (&key, numarray, NELEMS(numarray),
//采用二分法进行搜索
        sizeof(int), (int(*) (const void *,const void *))numeric);
    return (itemptr != NULL);
}
void main( ){
    if (lookup(512)) //根据搜索结果输出提示信息
        printf("512 is in the table.\n");
    else printf("512 isn't in the table.\n");
}
```

运行上述程序，首先声明自定义函数，在自定义函数中采用二分法进行搜索；然后在主函数中进行调用，并根据搜索的结果输出提示信息。

21.26 整数相除函数：div()

函数 div() 用于将两个整数相除，返回商和余数，其原型如下。

```
div_t (int number, int denom);
```

【参数】 参数 number 为被除数；参数 denom 为除数。

【返回值】 两个数相除后，将返回商和余数。

【例 21-26】 下面的示例演示了 div() 函数的使用，采用该函数进行整数相除，代码如下。

```
#include <stdlib.h>           //加入工具库
#include <stdio.h>           //加入标准输入输出库
div_t x;
void main( ){
    x = div(10,3);           //相除
    printf("10 div 3 = %d remainder %d\n", x.quot, x.rem); //输出
}
```

运行上述程序，首先将两个数相除，然后再将相除所返回的值输出到控制台。

21.27 长整型数相除函数：ldiv()

函数 ldiv() 用于实现两个长整型数相除，返回商和余数，其原型如下。

```
ldiv_t ldiv(long lnumber, long ldenom);
```

【参数】 参数 lnumber 为被除数；参数 ldenom 为除数。

【返回值】两个数相除后，将返回商和余数。

【例 21-27】下面的示例演示了 ldiv() 函数的使用，采用该函数进行长整型数相除，代码如下。

```
#include <stdlib.h>           //加入工具库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    ldiv_t lx;
    lx = ldiv(100000L, 30000L);    //相除
    printf("100000 div 30000 = %ld remainder %ld\n",lx.quot, lx.rem);
    //输出
}
```

运行上述程序，采用 ldiv() 函数进行长整型数相除，然后将返回值输出至控制台。

21.28 发出 DOS 命令函数: system()

函数 system() 用于发出一个 DOS 命令，其原型如下。

```
int system(char *command);
```

【参数】参数 command 为需要执行的 DOS 命令。

【返回值】该函数执行成功后，将返回 0；否则，返回 -1。

【例 21-28】下面的示例演示了 system() 函数的使用，在程序中使用该函数执行 DOS 命令，代码如下。

```
#include <stdlib.h>           //加入工具库
#include <stdio.h>           //加入标准输入输出库
void main( ){
    printf("About to spawn command.com and run a DOS command\n");
    system("dir");            //执行 DOS 命令
}
```

运行上述程序，首先输出提示信息，然后使用 system() 函数调用 DOS 命令。

附录 索引

A

abort()函数: 异常终止进程.....	321
abs()函数: 整数绝对值.....	40
absread()函数: 磁盘读数据.....	215
abswrite()函数: 磁盘写数据.....	216
access()函数: 文件访问权限设置.....	260
acos()函数: 反余弦值.....	40
allocmem()函数: DOS分配存储段.....	217
arc()函数: 画弧线.....	110
asctime()函数: 获取系统时间.....	310
asin()函数: 反正弦值.....	41
atan()函数: 反正切.....	42
atan2()函数: Y/X反正切.....	42
atof()函数: 转换字符串.....	324
atoi()函数: 转换字符串.....	325
atol()函数: 转换字符串.....	325

B

bar()函数: 绘制二维条形图.....	146
bar3d()函数: 绘制三维条形图.....	147
bcmp()函数: 比较字符串是否相等.....	72
bcopy()函数: 复制字符串.....	73
bdos()函数: DOS系统调用.....	218
bell()函数: 响铃.....	96
bioscom()函数: 串行IO通信.....	279

biosdisk()函数: 软硬盘IO	280
biosequip()函数: 检查设备	281
bioskey()函数: 键盘接口	282
biosmemory()函数: 获取存储块大小	283
biostime()函数: 设置BIOS时间	283
block()函数: 在屏幕上画一矩形并填充	97
brk()函数: 更改数据段空间分配	195
bsearch()函数: 二分法搜索	335
bzero()函数: 将字符串指定字节置零	74

C

calloc()函数: 分配主存储器	315
ceil()函数: 不小于某数的最小整数	43
chdir()函数: 更改工作目录	204
chmod()函数: 改变文件访问方式	254
circle()函数: 画圆	111
cleardevice()函数: 清除图形屏幕	141
clearerr()函数: 复位错误标志	25
clearviewport()函数: 清除图形视区	137
close()函数: 关闭文件	246
closegraph()函数: 关闭图形系统	161
clreol()函数: 清除字符	265
clrscr()函数: 清除文本	265
clrscr()函数: 清屏	94
coreleft()函数: 获取未使用内存大小	196
cos()函数: 余弦值	44
cosh()函数: 双曲余弦值	44
country()函数: 返回国家相关信息	218
cputs()函数: 写字符	266
creat()函数: 创建文件	247
ctime()函数: 转换日期时间字符串	312
ctrlbrk()函数: 设置Ctrl_Break处理程序	219

cursor()函数: 设定光标形态 98

D

delay()函数: 短暂延时 99

delline()函数: 删除行 267

detectgraph()函数: 通过检测硬件确定图形驱动程序和模式 163

difftime()函数: 计算时间差 311

DispBCD()函数: 显示七段数码管数字 107

div()函数: 整数相除 336

dosexterr()函数: 获取扩展DOS错误信息 220

dostounix()函数: UNIX时间格式转换 243

drawpoly()函数: 绘制多边形 164

dup()函数: 复制文件句柄 256

E

ecvt()函数: 转换浮点数 326

ellipse()函数: 绘制椭圆 116

eof()函数: 文件检测结束 247

exit()函数: 终止程序 322

exp()函数: e的次幂 45

F

fabs()函数: 浮点数绝对值 39

farcalloc()函数: 申请堆栈空间 197

farcoreleft()函数: 获取空余存储区空间大小 198

farfree()函数: 释放堆中空间 199

farmalloc()函数: 存储块分配空间 199

farrealloc()函数: 存储块空间调整 200

fclose()函数: 关闭流 22

fcvt()函数: 转换浮点数 327

fdopen()函数: 连接文件句柄	259
feof()函数: 检测流上的文件结束符	29
ferror()函数: 检测流上的错误	23
fflush()函数: 清除流	22
fgetc()函数: 从流中读取字符	26
fgetchar()函数: 从流中读取字符	27
fgetpos()函数: 取得当前文件的句柄	19
fgets()函数: 从流中读取字符	28
filelength()函数: 文件字节数统计	251
fillellipse()函数: 绘制并填充椭圆	117
fillpoly()函数: 绘制多边形并填充	165
findfirst()函数: 搜索磁盘目录	205
findnext()函数: 搜索磁盘目录	206
floodfill()函数: 填充有界区域	167
floor()函数: 不大于某数的最大整数	46
fmod()函数: 余数	47
fnmerge()函数: 构造文件名	214
fopen()函数: 打开流	21
_fpreset()函数: 重置浮点运算系统	261
fprintf()函数: 将格式化内容输出到流	32
fputchar()函数: 输出字符到标准输出流 (stdout)	27
fputs()函数: 送字符串到流中	30
fread()函数: 从流中读数据	30
free()函数: 释放已分配块	316
freemem()函数: 释放已分配DOS内存块	201
freopen()函数: 替换流	24
frexp()函数: 计算浮点数尾数和指数	47
fscanf()函数: 从流中执行格式化输入内容	33
fseek()函数: 重定位流上的文件指针	37
fsetpos()函数: 定位流上的文件指针	20
ftell()函数: 返回当前文件指针	38
fwrite()函数: 写内容到流	31

G

gcvt()函数: 转换浮点数	328
get_chi_font()函数: 取汉字的点阵	99
get_eng_font()函数: 取英文字符的点阵	101
getarccoords()函数: 取得最后一次调用画弧线坐标	109
getaspectratio()函数: 显示当前图形模式的纵横比	113
getbkcolor()函数: 获取当前背景颜色	153
getc()函数: 从流中取字符	1
getcbrk()函数: 获取Control_break设置	221
getch()函数: 获取字符	264
getchar()函数: 从stdin流中读字符	2
getche()函数: 从控制台取字符(带回显)	2
getcolor()函数: 获取当前画线颜色	151
getcurdir()函数: 指定当前目录	207
getcwd()函数: 获取当前工作目录	208
getdate()函数: 获取DOS日期	222
getdisk()函数: 获取磁盘驱动器号	209
getenv()函数: 获取环境中字符串	322
getfillpattern()函数: 将用户定义的填充模式复制到内存	120
getfillsettings()函数: 获取填充模式和填充颜色	118
getftime()函数: 获取文件时间	253
getgraphmode()函数: 获取当前图形模式	168
getimage()函数: 将指定区域的位图调入内存	170
getkey()函数: 读键	102
getlinesettings()函数: 获取当前线型、模式和宽度	152
getmaxcolor()函数: 获取可以传给函数setcolor的最大颜色	155
getmaxx()函数: 返回屏幕的最大x坐标	142
getmaxy()函数: 返回屏幕的最大y坐标	143
getmoderange()函数: 获取给定图形驱动程序的模式范围	174
getpalette()函数: 获取有关当前调色板的信息	156
getpixel()函数: 获取指定像素的颜色	175
gets()函数: 从流中取字符	7

gettext()函数: 文本复制	268
gettextsettings()函数: 获取当前图形文本字体的信息	124
gettime()函数: 获取系统时间	244
getviewsettings()函数: 获取当前视口的信息	136
getw()函数: 从流中取整数	3
getx()函数: 获取当前图形位置的x坐标	138
gety()函数: 获取当前图形位置的y坐标	139
gmtime()函数: 获取格林尼治结构的时间	310
gotoxy()函数: 设置光标	269
graphdefaults()函数: 将所有图形设置复位为它们的默认值	178
grapherrormsg()函数: 返回一个错误信息串的指针	179
_graphfreemem()函数: 用户可修改的图形存储区释放	180
_graphgetmem()函数: 用户可修改的图形存储区分配	182
graphresult()函数: 返回最后一次不成功的图形操作的错误代码	135

H

highvideo()函数: 高亮显示文本	269
hypot()函数: 计算直角三角形斜边长度	50

I

imagesize()函数: 返回保存位图像所需的字节数	183
initgraph()函数: 初始化图形系统	135
inport()函数: 从硬件端口中输入	224
inportb()函数: 从硬件端口中输入	225
insline()函数: 插入空行	270
int86()函数: 通用 8086 软中断接口	229
int86x()函数: 通用 8086 软中断接口	230
intdos()函数: 通用DOS接口	227
intdosx()函数: 通用DOS中断接口	228
intr()函数: 改变软中断接口	231
isalnum()函数: 判断字符是否为字母或数字	58

isalpha()函数: 判断字符是否为英文字母	59
isascii()函数: 判断字符是否为ASCII码	60
isatty()函数: 设备类型检查	257
isblank()函数: 判断字符是否为TAB或空格	61
iscntrl()函数: 判断字符是否为控制字符	57
isgraph()函数: 判断字符是否为除空格外的可打印字符	61
islower()函数: 判断字符是否为小写英文字母	62
isprint()函数: 判断字符是否为可打印字符(含空格)	63
ispunct()函数: 判断字符是否为标点符号	64
isspace()函数: 判断字符是否为空白符	65
isupper()函数: 判断字符是否为大写英文字母	66
isxdigit()函数: 判断字符是否为十六进制数字	67
itoa()函数: 转换整数	329

K

keep()函数: 退出并继续驻留	232
-------------------	-----

L

ldexp()函数: 计算幂	48
ldiv()函数: 长整型数相除	336
lfind()函数: 线性搜索	332
line()函数: 在指定两点间画直线	185
linerel()函数: 从当前位置点到当前点绘制直线	187
lineto()函数: 在指定两点间画直线	186
localtime()函数: 获取tm结构的系统时间	309
lock()函数: 文件共享锁设置	258
log()函数: 自然对数	51
log10()函数: 对数	51
longjmp()函数: 非局部转移执行	294
lowvideo()函数: 选择低亮度字符	188
lsearch()函数: 线性搜索	333

lseek()函数: 移动文件指针	252
-------------------------	-----

M

malloc()函数: 内存分配	316
memccpy()函数: 字节复制	285
memchr()函数: 字符搜索	287
memcmp()函数: 串比较	288
memcpy()函数: 字节复制	286
memicmp()函数: 串比较	289
memmove()函数: 字节移动	289
memset()函数: 串设置	290
mkdir()函数: 建立目录	209
mktemp()函数: 建立文件名	210
modf()函数: 取浮点数小数部分	49
move()函数: 将光标移动到指定位置	103
moverel()函数: 将当前点移动一相对距离	145
moveto()函数: 将当前点移到(x, y)	144
movmem()函数: 从源字符中移动字节到目标字符	76

N

noidle()函数: 调用系统例程	103
--------------------------	-----

O

open()函数: 打开文件	245
outport()函数: 从硬件端口中输出	226
outportb()函数: 从硬件端口中输出	226
outtext()函数: 在视区显示字符串	126
outtextxy()函数: 在指定位置显示字符串	127

P

parsfnm()函数: 分析文件名	233
peek()函数: 检查存储单元	234
peekb()函数: 检查存储单元	235
perror()函数: 系统错误信息	11
pieslice()函数: 绘制扇形并填充	112
poke()函数: 存值到给定存储单元	236
pokeb()函数: 存值到给定存储单元	237
pow()函数: 计算x的y次幂	52
pow10()函数: 计算10的x次幂	53
printf()函数: 格式化输出	36
putc()函数: 输出一个字符到指定流中	4
putchar()函数: 在stdout上输出字符	5
putenv()函数: 加载环境中字符串	323
putimage()函数: 在屏幕上输出位图	172
putpixel()函数: 在指定位置绘制像素	176
puts()函数: 将字符串送到流中	6
puttext()函数: 文本复制	271
putw()函数: 将字符或字送到流中	7
pyfc()函数: 查询汉字拼音	104

R

raise()函数: 发送信号	298
rand()函数: 随机数发生器	318
randbrd()函数: 随机块读	237
randbwr()函数: 随机块写	239
random()函数: 随机数发生器	319
randomize()函数: 初始化随机数发生器	319
read()函数: 读文件	248
realloc()函数: 重新分配主存	317
rectangle()函数: 绘制矩形	149

registerbgidriver()函数: 登录已连接进来的图形驱动程序代码	188
registerbgifont()函数: 注册链入的字体代码	128
remove()函数: 删除文件	9
rename()函数: 重命名文件	9
restorecrtmode()函数: 将屏幕模式恢复为先前的initgraph设置	189
rewind()函数: 将文件指针重新指向流的开头	10
rmdir()函数: 删除文件目录	211

S

sbrk()函数: 改变数据段空间位置	202
scanf()函数: 格式化输入	34
searchpath()函数: 查找文件	213
sector()函数: 绘制并填充椭圆扇区	115
segread()函数: 读段寄存器值	240
setactivepage()函数: 设置图形输出活动页	191
setallpalette()函数: 按指定方式改变所有的调色板颜色	158
setaspectratio()函数: 设置图形纵横比	114
setbkcolor()函数: 设置当前背景颜色	154
setblock()函数: 修改DOS分配	242
setbuf()函数: 把缓冲区与流相联	12
setcolor()函数: 设置当前画线颜色	150
setdate()函数: 设置DOS日期	222
setdisk()函数: 设置当前工作驱动器	212
setfillpattern()函数: 选择用户定义的填充模式	121
setfillstyle()函数: 设置填充模式和颜色	122
setgraphbufsize()函数: 改变内部图形缓冲区的大小	162
setgraphmode()函数: 设置当前图形模式	169
setjmp()函数: 非局部转移	295
setlinestyle()函数: 设置当前画线宽度和类型	193
setmem()函数: 把内存区域的字节设置成字符	75
setmode()函数: 设置文件打开方式	255
setpalette()函数: 设置有关当前调色板的信息	159

SetScrollBar()函数: 显示滚动条.....	108
settextjustify()函数: 设置文本的对齐方式.....	130
settextstyle()函数: 输出当前的文本属性.....	123
settime()函数: 设置DOS时间.....	223
setusercharsize()函数: 为矢量字体改变字符宽度和高度.....	131
setvbuf()函数: 把缓冲区与流相关.....	13
setviewport()函数: 为图形输出设置当前视口.....	140
setvisualpage()函数: 设置可见图形页号.....	192
signal()函数: 设置信号对应动作.....	297
sin()函数: 正弦值.....	53
sinh()函数: 双曲正弦值.....	54
sleep()函数: 执行挂起.....	241
spawnl()函数: 创建并运行子程序.....	292
spawnle()函数: 创建并运行子程序.....	293
sprintf()函数: 将格式化输出到字符串中.....	13
sqrt()函数: 平方根.....	55
srand()函数: 初始化随机数发生器.....	320
sscanf()函数: 执行字符串中的格式化输入.....	14
_status87()函数: 获取浮点处理器状态值.....	262
strcpy()函数: 把字符串复制到数组.....	76
strcat()函数: 字符串追加.....	78
strchr()函数: 查找字符串首次出现位置.....	79
strcmp()函数: 字符串比较.....	79
strcmpi()函数: 字符串比较(不区分大小写).....	81
strcpy()函数: 复制字符串到数组.....	77
strcspn()函数: 字符串查找.....	82
strdup()函数: 字符串复制.....	83
stricmp()函数: 字符串比较(不区分大小写).....	80
strlen()函数: 字符串长度.....	84
strlwr()函数: 将字符串转换为小写形式.....	84
strncat()函数: 字符串尾部追加.....	86
strncmp()函数: 字符串比较.....	86

strncmpi()函数: 字符串比较(不区分大小写)	88
strncpy()函数: 将字符串复制到数组	89
strnicmp()函数: 字符串比较(不区分大小写)	87
strpbrk()函数: 字符串查找	90
strrev()函数: 字符串倒序	91
strset()函数: 将字符串设置成指定字符	91
strstr()函数: 在字符串中查找指定字符首次出现位置	92
strtod()函数: 转换字符串	330
strtok()函数: 用分隔符分解字符串	93
strtol()函数: 转换字符串	331
strupr()函数: 将字符串转换为大写形式	85
swab()函数: 交换字节	331
system()函数: 发出DOS命令	337

T

tan()函数: 正切值	55
tanh()函数: 双曲正切值	56
tell()函数: 获取文件指针位置	251
textattr()函数: 设置文本属性	272
textbackground()函数: 文本背景色选择	273
textcolor()函数: 文本字符颜色选择	274
textheight()函数: 获取以像素为单位的字符串高度	132
textmode()函数: 文本模式设置	274
TextOut()函数: 在屏幕上指定位置输出字符串	96
textwidth()函数: 获取以像素为单位的字符串宽度	133
time()函数: 获取或设置时间	313
tmpfile()函数: 以二进制方式打开暂存文件	16
tmpnam()函数: 创建一个唯一的文件名	17
toascii()函数: 将字符转换为ASCII码	68
tolower()函数: 将字符转换为小写英文字母	69
toupper()函数: 将字符转换为大写英文字母	70
tzset()函数: UNIX时间兼容	314

U

ungetc()函数: 把字符退回到输入流.....	17
ungetch()函数: 把字符退回到键盘缓冲区.....	18
UpdateLCD()函数: 以指定模式刷新屏幕.....	95

V

va_arg()函数: 调用可变参数列表.....	308
va_end()函数: 停止使用可变参数.....	307
va_start()函数: 开始使用可变参数.....	306
vfprintf()函数: 格式化输出流.....	299
vfprintf()函数: 执行流中格式化输入.....	304
vprintf()函数: 格式化输出.....	300
vscanf()函数: 执行格式化输入.....	302
vsprintf()函数: 格式化输出串.....	301
vsscanf()函数: 执行流中格式化输入.....	303

W

wherex()函数: 返回水平光标位置.....	275
wherey()函数: 返回垂直光标位置.....	275
window()函数: 定义活动文本窗口.....	276
write()函数: 写文件.....	249
write_chi_font()函数: 写汉字点阵.....	105
write_eng_font()函数: 写英文字符的点阵.....	106

[G e n e r a l I n f o r m a t i o n]

书名 = C语言常用函数速查手册

作者 = 陈超等编著

页数 = 3 5 0

出版社 = 北京市：化学工业出版社

出版日期 = 2 0 1 0 . 0 6

SS号 = 1 2 6 0 5 8 7 4

DX号 = 0 0 0 0 0 6 9 0 1 7 0 7

URL = <http://book.szdnnet.org.cn/bookDetail.jsp?dxNumber=000006901707&d=E0AA5146DED77E4F72B373E8107C9CDD>

第 1 章	输入输出函数库：s t d i o . h
1 . 1	从流中取字符函数：g e t c ()
1 . 2	从 s t d i n 流中读字符函数：g e t c h a r ()
1 . 3	从控制台取字符（带回显）函数：g e t c h e ()
1 . 4	从流中取整数函数：g e t w ()
1 . 5	输出一个字符到指定流中函数：p u t c ()
1 . 6	在 s t d o u t 上输出字符函数：p u t c h a r ()
1 . 7	将字符串送到流中函数：p u t s ()
1 . 8	从流中取字符函数：g e t s ()
1 . 9	将字符或字送到流中函数：p u t w ()
1 . 1 0	重命名文件函数：r e n a m e ()
1 . 1 1	删除文件函数：r e m o v e ()
1 . 1 2	将文件指针重新指向流的开头函数：r e w i n d ()
1 . 1 3	系统错误信息：p e r r o r ()
1 . 1 4	把缓冲区与流相联函数：s e t b u f ()
1 . 1 5	把缓冲区与流相关函数：s e t v b u f ()
1 . 1 6	将格式化输出到字符串中函数：s p r i n t f ()
1 . 1 7	执行字符串中的格式化输入函数：s s c a n f ()
1 . 1 8	以二进制方式打开暂存文件函数：t m p f i l e ()
1 . 1 9	创建一个唯一的文件名函数：t m p n a m ()
1 . 2 0	把字符退回到输入流函数：u n g e t c ()
1 . 2 1	把字符退回到键盘缓冲区函数：u n g e t c h ()
1 . 2 2	取得当前文件的句柄函数：f g e t p o s ()
1 . 2 3	定位流上的文件指针函数：f s e t p o s ()
1 . 2 4	打开流函数：f o p e n ()
1 . 2 5	关闭流函数：f c l o s e ()
1 . 2 6	清除流函数：f f l u s h ()
1 . 2 7	检测流上的错误函数：f e r r o r ()
1 . 2 8	替换流函数：f r e o p e n ()
1 . 2 9	复位错误标志函数：c l e a r e r r ()
1 . 3 0	从流中读取字符函数：f g e t c ()
1 . 3 1	从流中读取字符函数：f g e t c h a r ()
1 . 3 2	输出字符到标准输出流（s t d o u t）函数：f p u t c h a r ()
1 . 3 3	从流中读取字符函数：f g e t s ()
1 . 3 4	检测流上的文件结束符函数：f e o f ()
1 . 3 5	送字符串到流中函数：f p u t s ()
1 . 3 6	从流中读数据函数：f r e a d ()
1 . 3 7	写内容到流函数：f w r i t e ()
1 . 3 8	将格式化内容输出到流函数：f p r i n t f ()
1 . 3 9	从流中执行格式化输入内容函数：f s c a n f ()
1 . 4 0	格式化输入函数：s c a n f ()
1 . 4 1	格式化输出函数：p r i n t f ()
1 . 4 2	重定位流上的文件指针函数：f s e e k ()
1 . 4 3	返回当前文件指针函数：f t e l l ()
第 2 章	数学函数库：m a t h . h
2 . 1	浮点数绝对值函数：f a b s ()
2 . 2	整数绝对值函数：a b s ()
2 . 3	反余弦值函数：a c o s ()
2 . 4	反正弦值函数：a s i n ()
2 . 5	反正切函数：a t a n ()
2 . 6	Y / X 反正切函数：a t a n 2 ()
2 . 7	不小于某数的最小整数函数：c e i l ()
2 . 8	余弦值函数：c o s ()
2 . 9	双曲余弦值函数：c o s h ()
2 . 1 0	e 的次幂函数：e x p ()

- 2.1.1 不大于某数的最大整数函数：f l o o r ()
- 2.1.2 余数函数：f m o d ()
- 2.1.3 计算浮点数尾数和指数函数：f r e x p ()
- 2.1.4 计算幂函数：l d e x p ()
- 2.1.5 取浮点数小数部分函数：m o d f ()
- 2.1.6 计算直角三角形斜边长度函数：h y p o t ()
- 2.1.7 自然对数函数：l o g ()
- 2.1.8 对数函数：l o g 1 0 ()
- 2.1.9 计算 x 的 y 次幂函数：p o w ()
- 2.2.0 计算 1 0 的 x 次幂函数：p o w 1 0 ()
- 2.2.1 正弦值函数：s i n ()
- 2.2.2 双曲正弦值函数：s i n h ()
- 2.2.3 平方根函数：s q r t ()
- 2.2.4 正切值函数：t a n ()
- 2.2.5 双曲正切值函数：t a n h ()

第 3 章 字符函数库：c t y p e . h

- 3.1 判断字符是否为控制字符函数：i s c n t r l ()
- 3.2 判断字符是否为字母或数字函数：i s a l n u m ()
- 3.3 判断字符是否为英文字母函数：i s a l p h a ()
- 3.4 判断字符是否为 A S C I I 码函数：i s a s c i i ()
- 3.5 判断字符是否为 T A B 或空格函数：i s b l a n k ()
- 3.6 判断字符是否为除空格外的可打印字符函数：i s g r a p h ()
- 3.7 判断字符是否为小写英文字母函数：i s l o w e r ()
- 3.8 判断字符是否为可打印字符（含空格）函数：i s p r i n t ()
- 3.9 判断字符是否为标点符号函数：i s p u n c t ()
- 3.10 判断字符是否为空白符函数：i s s p a c e ()
- 3.11 判断字符是否为大写英文字母函数：i s u p p e r ()
- 3.12 判断字符是否为十六进制数字函数：i s x d i g i t ()
- 3.13 将字符转换为 A S C I I 码函数：t o a s c i i ()
- 3.14 将字符转换为小写英文字母函数：t o l o w e r ()
- 3.15 将字符转换为大写英文字母函数：t o u p p e r ()

第 4 章 字符串函数库：s t r i n g . h

- 4.1 比较字符串是否相等函数：b c m p ()
- 4.2 复制字符串函数：b c o p y ()
- 4.3 将字符串指定字节置零函数：b z e r o ()
- 4.4 把内存区域的字节设置成字符函数：s e t m e m ()
- 4.5 从源字符中移动字节到目标字符函数：m o v m e m ()
- 4.6 把字符串复制到数组函数：s t p c p y ()
- 4.7 复制字符串到数组函数：s t r c p y ()
- 4.8 字符串追加函数：s t r c a t ()
- 4.9 查找字符串首次出现位置函数：s t r c h r ()
- 4.10 字符串比较函数：s t r c m p ()
- 4.11 字符串比较（不区分大小写）函数：s t r i c m p ()
- 4.12 字符串比较（不区分大小写）函数：s t r c m p i ()
- 4.13 字符串查找函数：s t r c s p n ()
- 4.14 字符串复制函数：s t r d u p ()
- 4.15 字符串长度函数：s t r l e n ()
- 4.16 将字符串转换为小写形式函数：s t r l w r ()
- 4.17 将字符串转换为大写形式函数：s t r u p r ()
- 4.18 字符串尾部追加函数：s t r n c a t ()
- 4.19 字符串比较函数：s t r n c m p ()
- 4.20 字符串比较（不区分大小写）函数：s t r n i c m p ()
- 4.21 字符串比较（不区分大小写）函数：s t r n c m p i ()
- 4.22 将字符串复制到数组函数：s t r n c p y ()
- 4.23 字符串查找函数：s t r p b r k ()
- 4.24 字符串倒序函数：s t r r e v ()
- 4.25 将字符串设置成指定字符函数：s t r s e t ()
- 4.26 在字符串中查找指定字符首次出现位置函数：s t r s t r ()
- 4.27 用分隔符分解字符串函数：s t r t o k ()

第5章 标准库函数库：system.h

- 5.1 清屏函数：clrscr()
- 5.2 以指定模式刷新屏幕函数：UpdateLCD()
- 5.3 在屏幕上指定位置输出字符串函数：TextOut()
- 5.4 响铃函数：bell()
- 5.5 在屏幕上画一矩形并填充函数：block()
- 5.6 设定光标形态函数：cursor()
- 5.7 短暂延时函数：delay()
- 5.8 取汉字的点阵函数：get_chi_font()
- 5.9 取英文字符的点阵函数：get_eng_font()
- 5.10 读键函数：getkey()
- 5.11 将光标移动到指定位置函数：move()
- 5.12 调用系统例程函数：noidle()
- 5.13 查询汉字拼音：pyfc()
- 5.14 写汉字点阵函数：write_chi_font()
- 5.15 写英文字符的点阵函数：write_eng_font()
- 5.16 显示七段数码管数字函数：DispBCD()
- 5.17 显示滚动条函数：SetScrollBar()

第6章 图形处理函数库：graphics.h

- 6.1 取得最后一次调用画弧线坐标函数：getarccoords()
- 6.2 画弧线函数：arc()
- 6.3 画圆函数：circle()
- 6.4 绘制扇形并填充函数：pieslice()
- 6.5 显示当前图形模式的纵横比函数：getaspectratio()
- 6.6 设置图形纵横比函数：setaspectratio()
- 6.7 绘制并填充椭圆扇区函数：sector()
- 6.8 绘制椭圆函数：ellipse()
- 6.9 绘制并填充椭圆函数：fillellipse()
- 6.10 获取填充模式和填充颜色函数：getfillsettings()
- 6.11 将用户定义的填充模式复制到内存函数：getfillpattern()
- 6.12 选择用户定义的填充模式函数：setfillpattern()
- 6.13 设置填充模式和颜色函数：setfillstyle()
- 6.14 输出当前的文本属性函数：settextstyle()
- 6.15 获取当前图形文本字体的信息函数：gettextsettings()
- 6.16 在视区显示字符串函数：outtext()
- 6.17 在指定位置显示字符串函数：outtextxy()
- 6.18 注册链入的字体代码函数：registerbgiFont()
- 6.19 设置文本的对齐方式函数：settextjustify()
- 6.20 为矢量字体改变字符宽度和高度函数：setusercharsize()
- 6.21 获取以像素为单位的字符串高度函数：textheight()
- 6.22 获取以像素为单位的字符串宽度函数：textwidth()
- 6.23 返回最后一次不成功的图形操作的错误代码函数：graphresult()

)

- 6.24 初始化图形系统函数：initgraph()
- 6.25 获取当前视口的信息函数：getviewsettings()
- 6.26 清除图形视区函数：clearviewport()
- 6.27 获取当前图形位置的x坐标函数：getx()
- 6.28 获取当前图形位置的y坐标函数：gety()
- 6.29 为图形输出设置当前视口函数：setviewport()
- 6.30 清除图形屏幕函数：cleardevice()
- 6.31 返回屏幕的最大x坐标函数：getmaxx()
- 6.32 返回屏幕的最大y坐标函数：getmaxy()
- 6.33 将当前点移到(x, y)函数：moveto()
- 6.34 将当前点移动一相对距离函数：moverel()
- 6.35 绘制二维条形图函数：bar()
- 6.36 绘制三维条形图函数：bar3d()
- 6.37 绘制矩形函数：rectangle()
- 6.38 设置当前画线颜色函数：setcolor()

- 6.39 获取当前画线颜色函数：getcolor()
- 6.40 获取当前线型、模式和宽度函数：getlinestettings()
- 6.41 获取当前背景颜色函数：getbkcolor()
- 6.42 设置当前背景颜色函数：setbkcolor()
- 6.43 获取可以传给函数setcolor的最大颜色函数：getmaxcolor()
- 6.44 获取有关当前调色板的信息函数：getpalette()
- 6.45 按指定方式改变所有的调色板颜色函数：setallpalette()
- 6.46 设置有关当前调色板的信息函数：setpalette()
- 6.47 关闭图形系统函数：closegraph()
- 6.48 改变内部图形缓冲区的大小函数：setgraphbufsize()
- 6.49 通过检测硬件确定图形驱动程序和模式函数：detectgraph()
- 6.50 绘制多边形函数：drawpoly()
- 6.51 绘制多边形并填充函数：fillpoly()
- 6.52 填充有界区域函数：floodfill()
- 6.53 获取当前图形模式函数：getgraphmode()
- 6.54 设置当前图形模式函数：setgraphmode()
- 6.55 将指定区域的位图调入内存函数：getimage()
- 6.56 在屏幕上输出位图函数：putimage()
- 6.57 获取给定图形驱动程序的模式范围函数：getmoderange()
- 6.58 获取指定像素的颜色函数：getpixel()
- 6.59 在指定位置绘制像素函数：putpixel()
- 6.60 将所有图形设置复位为它们的默认值函数：graphdefaults()
- 6.61 返回一个错误信息串的指针函数：grapherrormsg()
- 6.62 用户可修改的图形存储区释放函数：_graphfreemem()
- 6.63 用户可修改的图形存储区分配函数：_graphgetmem()
- 6.64 返回保存位图像所需的字节数函数：imagesize()
- 6.65 在指定两点间画直线函数：line()
- 6.66 在指定两点间画直线函数：lineto()
- 6.67 从当前位置点到当前点绘制直线函数：linereel()
- 6.68 选择低亮度字符函数：lowvideo()
- 6.69 登录已连接进来的图形驱动程序代码函数：registerbgidriver()
- 6.70 将屏幕模式恢复为先前的initgraph设置函数：restorecrtmode()
- 6.71 设置图形输出活动页函数：setactivepage()
- 6.72 设置可见图形页号函数：setvisualpage()
- 6.73 设置当前画线宽度和类型函数：setlinestyle()
- 第7章 动态内存管理函数库：alloc.h
 - 7.1 更改数据段空间分配函数：brk()
 - 7.2 获取未使用内存大小函数：coreleft()
 - 7.3 申请堆栈空间函数：farcalloc()
 - 7.4 获取空余存储区空间大小函数：farcoreleft()
 - 7.5 释放堆中空间函数：farfree()
 - 7.6 存储块分配空间函数：farmalloc()
 - 7.7 存储块空间调整函数：farrealloc()
 - 7.8 释放已分配DOS内存块函数：freemem()
 - 7.9 改变数据段空间位置函数：sbrk()
- 第8章 目录操作函数库：dir.h
 - 8.1 更改工作目录函数：chdir()
 - 8.2 搜索磁盘目录函数：findfirst()
 - 8.3 搜索磁盘目录函数：findnext()
 - 8.4 指定当前目录函数：getcurdir()
 - 8.5 获取当前工作目录函数：getcwd()
 - 8.6 获取磁盘驱动器号函数：getdisk()
 - 8.7 建立目录函数：mkdir()
 - 8.8 建立文件名函数：mktemp()
 - 8.9 删除文件目录函数：rmdir()
 - 8.10 设置当前工作驱动器函数：setdisk()
 - 8.11 查找文件函数：searchpath()

8.12 构造文件名函数：fnmerge()

第9章 系统接口函数库：dos.h

9.1 磁盘读数据函数：absread()

9.2 磁盘写数据函数：abswrite()

9.3 DOS分配存储段函数：allocmem()

9.4 DOS系统调用函数：bdos()

9.5 返回国家相关信息函数：country()

9.6 设置Ctrl_Break处理程序函数：ctrlbrk()

9.7 获取扩展DOS错误信息函数：dosexterr()

9.8 获取Control_break设置函数：getcbrk()

9.9 获取DOS日期函数：getdate()

9.10 设置DOS日期函数：setdate()

9.11 设置DOS时间函数：settime()

9.12 从硬件端口中输入函数：inport()

9.13 从硬件端口中输入函数：inportb()

9.14 从硬件端口中输出函数：outport()

9.15 从硬件端口中输出函数：outportb()

9.16 通用DOS接口函数：intdos()

9.17 通用DOS中断接口函数：intdosx()

9.18 通用8086软中断接口函数：int86()

9.19 通用8086软中断接口函数：int86x()

9.20 改变软中断接口函数：intr()

9.21 退出并继续驻留函数：keep()

9.22 分析文件名函数：parsfnm()

9.23 检查存储单元函数：peek()

9.24 检查存储单元函数：peekb()

9.25 存值到给定存储单元函数：poke()

9.26 存值到给定存储单元函数：pokeb()

9.27 随机块读函数：randbrd()

9.28 随机块写函数：randbwr()

9.29 读段寄存器值函数：segread()

9.30 执行挂起函数：sleep()

9.31 修改DOS分配函数：setblock()

9.32 UNIX时间格式转换函数：dostounix()

9.33 获取系统时间函数：getttime()

第10章 输入输出函数库：io.h

10.1 打开文件函数：open()

10.2 关闭文件函数：close()

10.3 创建文件函数：creat()

10.4 文件检测结束函数：eof()

10.5 读文件函数：read()

10.6 写文件函数：write()

10.7 文件字节数统计函数：filelength()

10.8 获取文件指针位置函数：telli()

10.9 移动文件指针函数：lseek()

10.10 获取文件时间函数：getftime()

10.11 改变文件访问方式函数：chmod()

10.12 设置文件打开方式函数：setmode()

10.13 复制文件句柄函数：dup()

10.14 设备类型检查函数：isatty()

10.15 文件共享锁设置函数：lock()

10.16 连接文件句柄函数：fdopen()

10.17 文件访问权限设置函数：access()

第11章 浮点数据处理库：float.h

11.1 重置浮点运算系统函数：_fpreset()

11.2 获取浮点处理器状态值函数：_status87()

第12章 控制台输入输出函数库：conio.h

12.1 获取字符函数：getch()

12.2 清除字符函数：clr_eol()

- 1 2 . 3 清除文本函数：c l r s c r ()
- 1 2 . 4 写字符函数：c p u t s ()
- 1 2 . 5 删除行函数：d e l l i n e ()
- 1 2 . 6 文本复制函数：g e t t e x t ()
- 1 2 . 7 设置光标函数：g o t o x y ()
- 1 2 . 8 高亮显示文本函数：h i g h v i d e o ()
- 1 2 . 9 插入空行函数：i n s l i n e ()
- 1 2 . 1 0 文本复制函数：p u t t e x t ()
- 1 2 . 1 1 设置文本属性函数：t e x t a t t r ()
- 1 2 . 1 2 文本背景色选择函数：t e x t b a c k g r o u n d ()
- 1 2 . 1 3 文本字符颜色选择函数：t e x t c o l o r ()
- 1 2 . 1 4 文本模式设置函数：t e x t m o d e ()
- 1 2 . 1 5 返回水平光标位置函数：w h e r e x ()
- 1 2 . 1 6 返回垂直光标位置函数：w h e r e y ()
- 1 2 . 1 7 定义活动文本窗口函数：w i n d o w ()
- 第 1 3 章 D E B U G 相关函数库：a s s e r t . h
- 第 1 4 章 B I O S 相关函数库：b i o s . h
 - 1 4 . 1 串行 I O 通信函数：b i o s c o m ()
 - 1 4 . 2 软硬盘 I O 函数：b i o s d i s k ()
 - 1 4 . 3 检查设备函数：b i o s e q u i p ()
 - 1 4 . 4 键盘接口函数：b i o s k e y ()
 - 1 4 . 5 获取存储块大小函数：b i o s m e m o r y ()
 - 1 4 . 6 设置 B I O S 时间函数：b i o s t i m e ()
- 第 1 5 章 内存相关函数库：m e m . h
 - 1 5 . 1 字节复制函数：m e m c c p y ()
 - 1 5 . 2 字节复制函数：m e m c p y ()
 - 1 5 . 3 字符搜索函数：m e m c h r ()
 - 1 5 . 4 串比较函数：m e m c m p ()
 - 1 5 . 5 串比较函数：m e m i c m p ()
 - 1 5 . 6 字节移动函数：m e m m o v e ()
 - 1 5 . 7 串设置函数：m e m s e t ()
- 第 1 6 章 进程管理函数库：p r o c e s s . h
 - 1 6 . 1 创建并运行子程序函数：s p a w n l ()
 - 1 6 . 2 创建并运行子程序函数：s p a w n l e ()
- 第 1 7 章 函数跳转函数库：s e t j m p . h
 - 1 7 . 1 非局部转移执行函数：l o n g j m p ()
 - 1 7 . 2 非局部转移函数：s e t j m p ()
- 第 1 8 章 信号定义函数库：s i g n a l . h
 - 1 8 . 1 设置信号对应动作函数：s i g n a l ()
 - 1 8 . 2 发送信号函数：r a i s e ()
- 第 1 9 章 函数参数处理函数库：s t d a r g . h
 - 1 9 . 1 格式化输出流函数：v f p r i n t f ()
 - 1 9 . 2 格式化输出函数：v p r i n t f ()
 - 1 9 . 3 格式化输出串函数：v s p r i n t f ()
 - 1 9 . 4 执行格式化输入函数：v s c a n f ()
 - 1 9 . 5 执行流中格式化输入函数：v s s c a n f ()
 - 1 9 . 6 执行流中格式化输入函数：v f s c a n f ()
 - 1 9 . 7 开始使用可变参数函数：v a _ s t a r t ()
 - 1 9 . 8 停止使用可变参数函数：v a _ e n d ()
 - 1 9 . 9 调用可变参数列表函数：v a _ a r g ()
- 第 2 0 章 时间函数库：t i m e . h
 - 2 0 . 1 获取 t m 结构的系统时间函数：l o c a l t i m e ()
 - 2 0 . 2 获取格林尼治结构的时间函数：g m t i m e ()
 - 2 0 . 3 获取系统时间函数：a s c t i m e ()
 - 2 0 . 4 计算时间差函数：d i f f t i m e ()
 - 2 0 . 5 转换日期时间字符串函数：c t i m e ()
 - 2 0 . 6 获取或设置时间函数：t i m e ()
 - 2 0 . 7 U N I X 时间兼容函数：t z s e t ()
- 第 2 1 章 标准工具库函数库：s t d l i b . h

- 2 1 . 1 分配主存储器函数：c a l l o c ()
- 2 1 . 2 释放已分配块函数：f r e e ()
- 2 1 . 3 内存分配函数：m a l l o c ()
- 2 1 . 4 重新分配主存函数：r e a l l o c ()
- 2 1 . 5 随机数发生器函数：r a n d ()
- 2 1 . 6 随机数发生器函数：r a n d o m ()
- 2 1 . 7 初始化随机数发生器函数：r a n d o m i z e ()
- 2 1 . 8 初始化随机数发生器函数：s r a n d ()
- 2 1 . 9 异常终止进程函数：a b o r t ()
- 2 1 . 1 0 终止程序函数：e x i t ()
- 2 1 . 1 1 获取环境中字符串函数：g e t e n v ()
- 2 1 . 1 2 加载环境中字符串函数：p u t e n v ()
- 2 1 . 1 3 转换字符串函数：a t o f ()
- 2 1 . 1 4 转换字符串函数：a t o i ()
- 2 1 . 1 5 转换字符串函数：a t o l ()
- 2 1 . 1 6 转换浮点数函数：e c v t ()
- 2 1 . 1 7 转换浮点数函数：f c v t ()
- 2 1 . 1 8 转换浮点数函数：g c v t ()
- 2 1 . 1 9 转换整数函数：i t o a ()
- 2 1 . 2 0 转换字符串函数：s t r t o d ()
- 2 1 . 2 1 转换字符串函数：s t r t o l ()
- 2 1 . 2 2 交换字节函数：s w a b ()
- 2 1 . 2 3 线性搜索函数：l f i n d ()
- 2 1 . 2 4 线性搜索函数：l s e a r c h ()
- 2 1 . 2 5 二分法搜索函数：b s e a r c h ()
- 2 1 . 2 6 整数相除函数：d i v ()
- 2 1 . 2 7 长整型数相除函数：l d i v ()
- 2 1 . 2 8 发出DOS命令函数：s y s t e m ()