C语言

第12讲：文件操作

赵岩

哈尔滨工业大学软件学院

August 20, 2011

目录

目录

流的概念

- 一般称为数据流，也叫做字节流、比特流等
  - 类似于水流
- 可控的流
  - 介质保存的文件流
- 不可控流
  - 缓存保存的流或网络数据流

Input设备

- 输入设备
  - 标准输入设备–键盘(STDIN)、
  - 鼠标、软盘、硬盘、光驱（以文件的形式）
  - 串行口、并行口、USB接口、IEEE1394口、网络端口
  - 扫描仪、视频采集卡、电视卡、游戏杆、话筒
  - ……

Output设备

- 输出设备
  - 标准输出设备–显示器(STDOUT)
  - 打印机、软盘、硬盘、CD-RW/DVD-RW（以文件的形式）
  - 串行口、并行口、USB接口、IEEE1394口、网络端口
  - 音箱
  - ……
- 单纯的输入设备或者单纯的输出设备越来越少

标准输入和输出

- 操作系统一般都设定标准输入与输出设备
  - DOS、Linux、Unix、Mac_OS
- 标准输入(STDIN)就是键盘
- 标准输出(STDOUT)就是显示器
- 标准错误输出(STDERR)也是显示器
- 输入，输出重定向

问题

*STDOUT*和*STDERR*的区别何在*?*

标准输入和输出重定向

```
void main(void) {
    int c;
    while ((c=getchar()) != EOF)
    putchar(c);
}
```

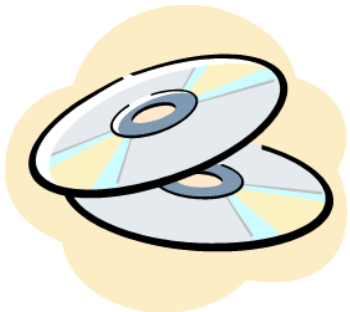
- 无重定向
  - prog.exe < STDIN > STDOUT
- 输入重定向
  - prog.exe < input.txt
- 输出重定向
  - prog.exe > output.txt
- 输入,输出全部重定向
  - prog.exe < input.txt > output.txt

复杂的命令）

- 管道
  - ls -l | grep ”^d” | awk ’print $8’ | sort -nr
- 占位符
  - split -l 1 test split
  - ls -l | split -l 1 split ?
  - ls -l | split -l 1 - split
- xarg
  - rm splitaa splitab splitac
  - ls | grep ”splita[a-c]” | rm ?
  - ls | grep ”splita[a-c]” | xarg rm

外部存储设备

- 数据断电后不丢失
  - 磁盘（Magnetic disks）
  - 光盘（CD、DVD）
  - U盘（Flash Memory）
  - ...



存储设备保存原理

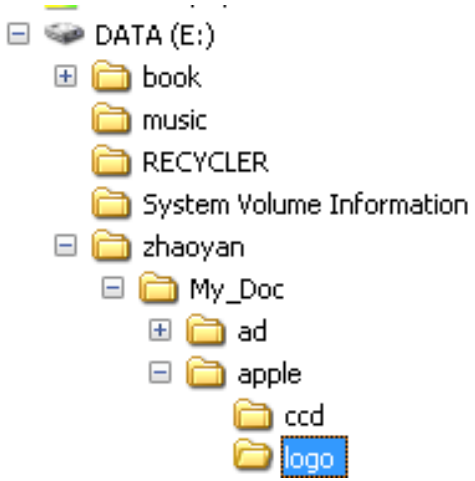
- 磁盘
  - 磁盘表面涂有磁性物质
  - 磁性单元的N-S极的两种指向表示0-1
- 光盘
  - 凹坑（原理演示）
- U盘
  - 电擦写门（原理演示）

文件（File）

- 为什么需要文件？
  - 要让数据在断电后依然存在，就必须用外部存储设备
  - 外存原理迥异，如果必须按其特性使用....杯具！
- 文件哪里来？
  - 操作系统有一个模块，叫“文件系统”
  - 文件系统封装了外存的特性，让我们可以用同一个接口使用不同外存
  - 这个接口就是“文件”
- 原理
  - 操作系统决定文件在介质上的所有保存细节。
- 使用文件需要知道
  - 目录结构，文件结构，命令接口，开发接口

文件目录

- 树形的目录结构是文件系统的事实标准
- 绝对路径（Absolute Path）表示
  - 在Unix/Linux下：/home/zhaoyan/main.c
  - 在DOS/Windows下：D:\home\zhaoyan\main.c
- 相对路径（Relative Path）表示
  - ../main.c



文件的类型

- 文件在外存存储就和数据在内存保存一样
  - 都是数据二进制形式的保存
  - 介质不关心数据类型
  - 你把数据当作是什么类型，它就是什么类型
  - 如果以类型A的二进制形式存入，却当作类型B读出，就乱套了
- 读文件前，必须确切知道文件每一个字节的确切类型和含义
- 文件格式分类
  - 二进制方式和文本方式

文件的格式

- 文本文件（Text File）
  - 所有内容都是可打印字符的文件
  - .txt、.c、.h、.htm
- 二进制文件（Binary File）
  - 平实地反映数据和文件的真面貌，不做任何处理
  - .exe、.bmp、.mp3、.jpg、.doc、.swf……

文本文件换行符

- DOS/Windows下：
  - 用”\r\n”表示换行
  - 按文本文件写入”\n”，会自动写入”\r\n”
  - 假设文件中有”\r\n”这样的连续字符
  - 按文本文件读，只能得到”\n”
- Unix/Linux下：
  - 用”\n”表示换行
- Mac下：
  - 用”\r”表示换行
  - 写入“\n”，自动改为“\r”
  - 读入“\r”，自动转为“\n”

Shell接口

- 操作系统界面（Shell）提供的文件操作命令
- DOS下：
  - copy、del、move、type……
- Windows下：
  - 鼠标拖拽、**ctrl+c, v**、ctrl+x, v……
- Unix/Linux下：
  - cp、rm、mv、cat……
- 命令接口都是一些用开发接口编写的程序

文件开发接口

- 给程序员在程序中使用的接口
- DOS下：
  - INT 21H……
- Windows下：
  - CreateFile()、ReadFile()、WriteFile()、CloseHandle()……
- Unix/Linux下：
  - open()、read()、write()、close()……
- 很多语言提供了自己的访问文件接口，它们都是基于操作系统提供的接口实现的

文件访问

- open：打开文件，获得对此文件的指针、引用和句柄等，以后通过它们使用此文件
- read：读文件。参数一般指明要读多少字节，读到哪块内存
- write：写文件。参数一般指明把哪块内存的内容写入文件，要写多少字节
- close：关闭文件，表明操作结束，本程序不再使用此文件。open后要及时关闭。
- lseek：随时调整file position
  - 表示距离文件头部的偏移量
  - read/write结束后，自动加上读/写的字节数

打开文件

- pathname是文件路径。可以是绝对或相对路径
- flags是打开方式，常用为O_RDONLY、O_WRONLY、O_RD
  及O_TEXT、O_BINARY的或运算
- 返回值为文件描述符（File Descriptor）也叫文件句柄（File Handle），留待以后使用。如果打开失败，返回值为-1

```
int open(const char *pathname, int flags);
int fd = open("C:\\AUTOEXEC.BAT",
              O_RDWR | O_CREAT | O_TEXT );
```

读入文件

- fd是open返回的文件描述符
- 读入的数据将保存在buf指向的内存
- count是最大允许读入的字节数
- 返回值为实际读入的字节数，可能小于count，返回-1表示出错

```
int read(int fd, void *buf,
         unsigned int count);
int n_read = read(fd, buf, BUFSIZE);
```

写入文件

- fd是open返回的文件句柄
- buf指向的数据被写入文件
- count是写入多少字节
- 返回值为实际写入的字节数，可能小于count。返回-1表示出错

```
int write(int fd, const void *buf,
          unsigned int count);
int n_write = write(fd, buf, BUFSIZE);
```

关闭文件

- fd是open返回的文件句柄
- 关闭成功返回0，否则返回-1

```
int close(int fd);
int ret;
ret = close(fd);
```

文件位置

- fd是open返回的文件句柄
- offset是相对fromwhere的位置偏移多少，可以是负数
- fromwhere可以是SEEK_SET、SEEK_CUR或SEEK_END中的一个，分别表示文件头部、当前位置和文件末尾
- 成功返回移位后的当前位置，从文件头算起；否则返回-1L

```
long lseek(int fd, long offset,
           int fromwhere);
long pos;
pos = lseek(fd, 100L, SEEK_CUR);
```

错误处理流程

- 根据errno，做适当的处理
  - extern int errno;
  - stdio模块中定义的全局变量
  - 它的值说明错误类型
- 用perror告诉用户错误的具体信息，让他协助排除错误
  - void perror(const char *string);
  - 向stderr输出string，并在其后附加一个':'，再输出错误对应的文字信息

基本文件实例

```
int WriteData(int fh, void *buf, int len){
 int written = 0; int val;
 while(written<len){
  val = write(fh,((char*)buf)+
                  written,len-written);
  if(val == -1){
   return val;
  }
  written += val;
 }
 return written;
}
```

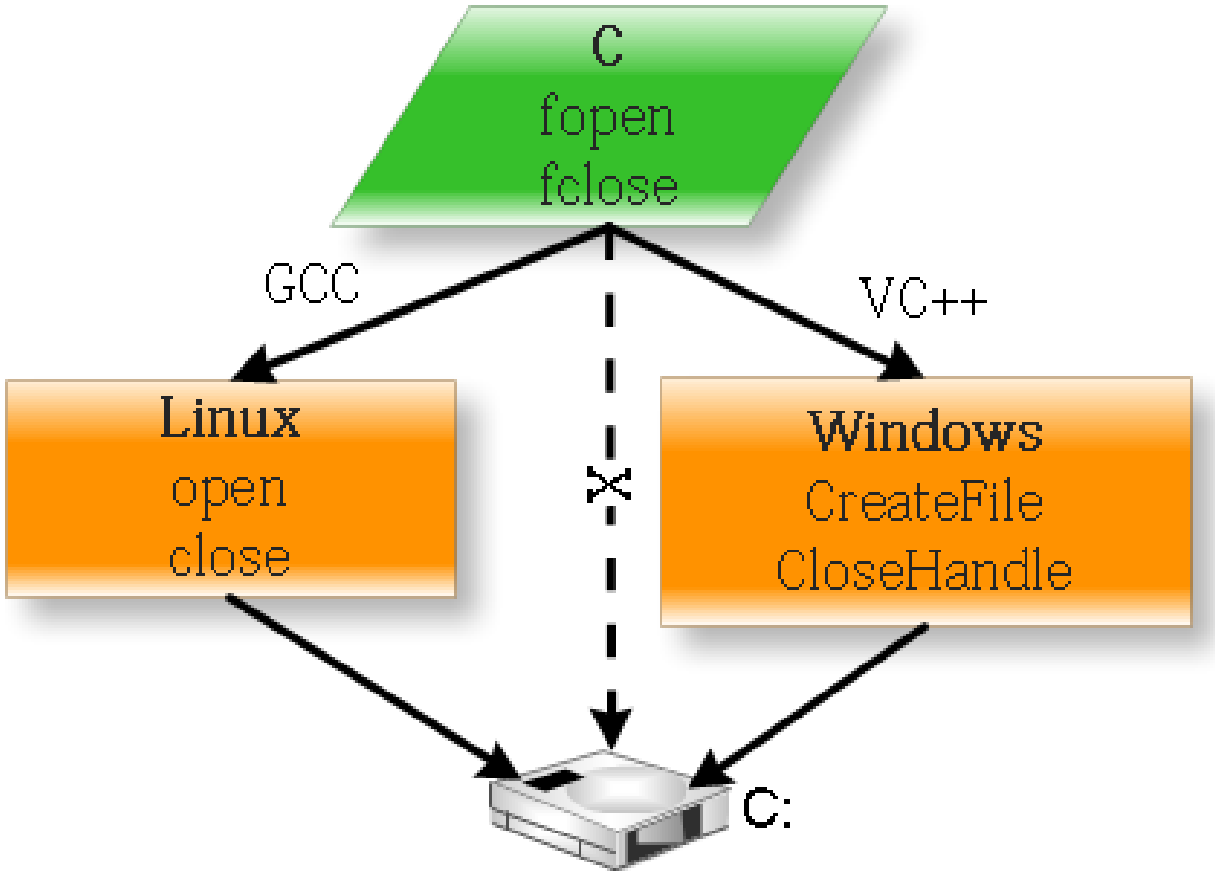
基本文件操作

- open()、read()、write()、close()和lseek()并不是ANSI C支持的
- 它们定义在POSIX标准中
  - POSIX是Unix/Linux等众多操作系统遵循的标准，Windows也有支持POSIX的环境（Cygwin）
  - 在POSIX的世界里具有可移植性
- VC支持它们，但在标识符前加了一个下划线
  - _open(), _O_RDWR
- ANSI C有自己的标准文件操作函数
  - 它们包装了操作系统提供的文件操作接口
  - 在C语言的世界里具有可移植性

基本文件操作的理论意义

- open、read、write、close这种模式成为使用流的经典模式
  - 比如网络流操作就是使用此模式，甚至在一些平台上可以和文件采用完全相同的函数操作
- lseek并不支持网络流
- 对于不能进行“流控”的流
  - 如果当前数据不立即处理
  - 后来的数据就可能会冲走当前数据
  - 于是当前数据丢失
- 流控成为流处理中的一个专门技术
  - 缓冲、握手协议

函数分层表示


C
fopen
fclose
GCC
VC++
Linux
open
close
Windows
CreateFile
CloseHandle
C:

fopen函数

```
FILE* fopen(const char *path,
            const char *mode);
```

- path是文件路径。可以是绝对或相对路径
- mode是打开方式，常用为
  - ”r” 表示只读，文件不存在，则失败
  - ”w” 表示只写（如文件已存在，则覆盖，否则建立新文件）
  - ”a” 表示末尾添加，如文件不存在，建立新文件
  - ”+”用在”r”、”w”、”a”之后，表示按读写方式打开
  - ”b”和上面的各种组合合用（不可做第一个字符），表示按二进制方式打开，否则就是文本方式（Unix/Linux下忽略）

fopen函数的三个问题

问题

$r+$与$w+$之间的区别是什么？

问题

为什么标志b在$linux/unix$被忽略？

A:参考http://user.qzone.qq.com/1619604531/infocenter

问题

$FILE$是什么东东？

FILE的类型

- FILE是C语言定义的一个结构类型
  - 不同的库在成员的定义上会有所不同
  - stdin、stdout和stderr的类型都是FILE *
- fopen成功后
  - 建立一个FILE结构的全局变量，该变量里记录了访问文件的重要信息：
    - 文件位置标记，缓冲区指针，等
  - 该变量的地址作为返回值返回

文件指针细节

```
typedef struct
  {
   short level;              /*缓冲区'满'或'空'的程度*/
   unsigned flags;           /*文件状态标志*/
   char fd;                  /*文件描述符*/
   unsigned char hold;       /*如无缓冲区不读字符*/
   short bsize;              /*缓冲区的大小*/
   unsigned char *buffer;    /*数据缓冲区的位置*/
   unsigned char *curp;      /*指针当前的指向*/
   unsigned istemp;          /*临时文件指示器*/
   short token;              /*用于有效性检查*/
  }FILE;
```

文件（句柄）指针

- FILE *fp ;
  - 是FILE型指针变量
  - 标识一个特定的磁盘文件
  - 与文件相关联的每个流都有一个FILE类型的控制结构，定义有关文件操作的信息
  - 用户绝对不应修改

关闭文件

- fclose(文件指针);
- 关闭fp所指向的文件。把遗留在缓冲区中的数据写入文件
- 打开文件前先关闭无用文件，因为系统限制同时打开的文件总数
- fclose函数的返回值：
  - 当顺利地执行了关闭操作，则返回值为0；
  - 如果返回值为非零值，则表示关闭时有错误。
  - 一般只有驱动器中无盘或盘空间不够时，才失败。

fopen使用惯例

- 只在windows下可用
  - FILE *fp = fopen(”C:\\AUTOEXEC.BAT”, ”r+”);
- linux和windows下都可以用
  - FILE *fp = fopen(”../AUTOEXEC.BAT”, ”r+”);
- 一定要判断fopen是否成功!

Demo 1: open and close file

```
FILE *fp = fopen("test.txt", "w+");
if(fp == NULL){
    printf("%s file failure","test.txt");
    exit(1);
}
fclose(fp);
```

文件字符读写

```
int fgetc(FILE *fp);
int fputc(int c, FILE *fp);
```

- ch=fgetc (fp);
  - 从指定文件读入一个字符。
  - 该文件应以只读或读写方式打开。
  - 返回值：若读入成功，则返回字符，否则返回一个EOF
- fputc (ch,fp);
  - 把字符ch写到fp所指向的文件中去
  - 该文件应以写或读写方式打开。
  - 返回值：若输出成功，则返回输出的字符，否则返回EOF

文件字符读写实例

```
/*输出字符*/
FILE *fp = fopen("test.txt","w+");
fputc('a',fp);
fclose(fp);

/*读入字符*/
int c;
FILE *fp = fopen("test.txt","r+");
c = fgetc(fp);
printf("%c",c);
fclose(fp);
```

文件字符串读写

```
char *fgets(char *s,int n,FILE *fp);
int fputs(const char *s, FILE *fp);
```

- fgets
  - 当读到换行、文件末尾或读满n-1个字符时返回
  - 且在字符串末尾添加'0'
  - 返回值：若读入成功，则返回字符串地址，否则返回一个NULL
- fputs
  - 字符串输出到fp
  - 返回值：若输出成功，则返一个非负值，否则返回EOF

字符串读写实例

```
/*输出字符串*/
char *p = "hello world";
FILE *fp = fopen("test.txt","w+");
fputs(p, fp);
fclose(fp);

/*读入字符串*/
char str[20];
FILE *fp = fopen("test.txt","r+");
fgets(str,20,fp);
printf("%s",str);
fclose(fp);
```

读入一个长行

- getline (C++)
- 动态增加内存

```
while((c=getche())!='\n'){
    if(len==number){
        times++;
        number=times*Step;
        temp=str;
        str=(char *)realloc(str,number);
    }
    *(str+len)=c;
    len++;
}
```

文件格式化读写

```
fprintf(fp,format,arg1,arg2,...,argn);
fscanf(fp,format,&arg1,&arg2,...&argn);
```

- fprintf函数
  - 将输出项arg1,arg2...argn按指定format格式写入fp所指的文件中,即输出到磁盘文件;
  - 格式控制符与printf定义一致
- fscanf函数
  - 按指定format格式从fp所指的文件中读取数据依次送入arg1,arg2...argn的内存单元;
  - 格式控制符与scanf定义一致

文件格式化读写实例

```
/*格式化输出*/
FILE *fp = fopen("test.txt","w+");
fprintf(fp,"%d\n",100000000);
fprintf(fp, "%5.2f",100.1234);
fclose(fp);

/*格式化输入*/
float f; int i;
FILE *fp = fopen("test.txt","r+");
fscanf(fp, "%d",&i);
fscanf(fp, "%f",&f); /*scanf不支持printf(”fclose(fp);
```

C语言的二进制读写

```
size_t fread(void *ptr,    size_t size,
                    size_t nmemb, FILE *fp);
size_t fwrite(const void *ptr, size_t size,
                    size_t nmemb, FILE *fp);
```

- fread
  - 读二进制文件
  - 在fp所指向的文件中读取一组数据并将其放到内存中某个地址
- fwrite
  - 写二进制文件
  - 将内存中的某地址内容写入到fp所指向的文件中去

二进制读写实例

```
/*二进制写文件*/
int i = 100000000;
FILE *fp = fopen("test.bin","wb+");
fwrite(&i, sizeof(int), 1, fp);
fclose(fp);

/*二进制读文件*/
int i=0;
FILE *fp = fopen("test.bin","rb+");
fread(&i,sizeof(int),1,fp);
printf("%d",i);
fclose(fp);
```

文件位置定位

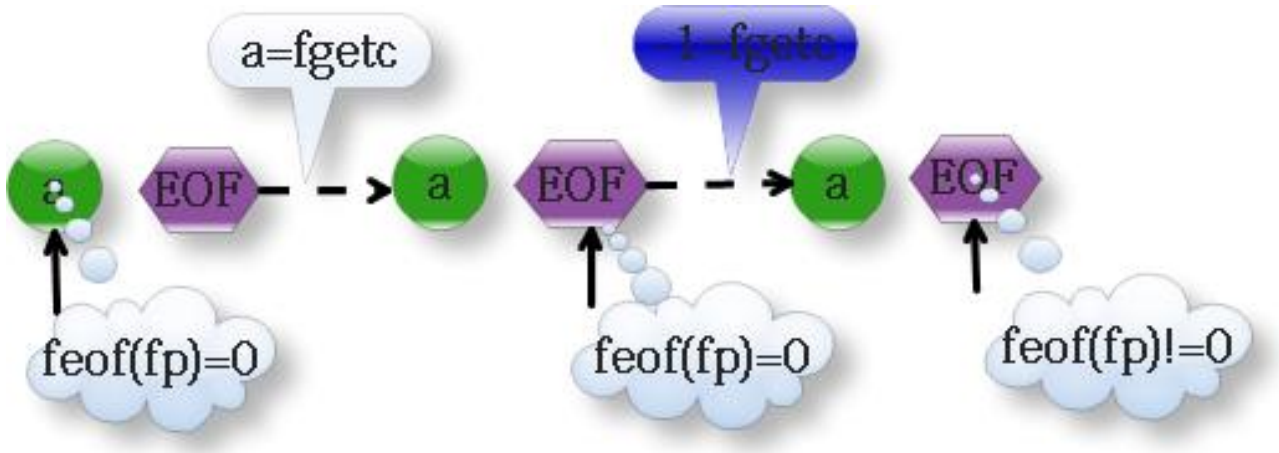
- void rewind(FILE *fp)
  - 使文件指针重新指向文件的开头位置。
  - 此函数没有返回值
- int fseek(FILE *fp, long offset, int whence);
  - 利用该函数可以改变文件指针的位置，从而实现随机读写
  - offset：文件指针的位移量，字节数，long型，加l或L；
  - from：起始位置
    - 0—文件开始，1—当前位置，2—文件末尾
- long ftell(FILE *fp)
  - 取得文件指针的当前位置，用字节数，返回long类型。

文件末尾判断

- int feof(FILE *fp);
  - ~~文件位置指针指向末尾的时候返回非0值，否则返回0。~~
  - Checks indicator, This indicator is generally set by a previous operation on the stream that reached the End-of-File.

文件末尾判断的实例

```
FILE *fp = fopen("test.txt","r+"); int c;
/* only 'a' in the file*/
while(!feof(fp)){
    c=fgetc(fp);
    printf("%d",c);
}
```



文件末尾判断的改进

```
FILE *fp = fopen("test.txt","r+"); int c;
while((c=fgetc(fp))!=EOF){
    printf("%d",c);
}
/*读出全部字符串*/
while(fgets("%s",20,fp)!=NULL){
....
}
/*读出全部二进制数据*/
while(fread(buffer, 16, 1, file) == 1){
....
}
```

缓存写入文件

- int fflush(FILE *fp);
  - 把缓冲区内容写入到物理设备

```
int c;
FILE *fp = fopen("test.txt","w+");
fputc('a', fp);
fputc('b',fp);
rewind(fp);
fflush(fp); /*fflush在混合读写的时候非常重要*/
c = fgetc(fp)
printf("%c",c);
fclose(fp);
```

处理文件注意事项

- 文件操作是事故多发区
  - 文件不存在
  - 文件已存在
  - 磁盘满
  - 写一个只读文件，读一个只写文件
  - 设备损坏
  - .....
- 在事故发生时，程序必须能自如应对，这就需要错误处理技术

错误处理

- ferror()函数
  - ferror(fp);
  - 检查输入输出函数的调用是否正确，返回非0值，表示出错。
  - 调用一个输入输出函数后，立即检查ferror函数的值

错误处理判断和策略

函数	当返回值	常用策略
fopen()	==NULL	转入错误处理流程
fprintf()	<0	
fputc()、putc()	==EOF	
fwrite()	<nmemb	
fputs()	==EOF	
fgets()	==NULL	停止再次调用该函数，继续下一步
fscanf()	==0 或 ==EOF	
fgetc()、getc()	==EOF	
fread()	<nmemb	
fclose()、fflush()	==EOF	忽略

两种方式的区别

- open族的功能一般由OS直接提供，其使用方式也比较具有通用性，在各种语言里基本一样
- fopen族的函数系包装了open族的函数，提供更强大的功能，但是效率略逊
- open族通常情况能直接反映文件的真实情况，因为它的操作都不假定文件的任何结构
- fopen族比较适合处理文本文件，或者结构单一的文件。会为了处理的方便而改变一些内容

C语言文件操作实例

- 模拟DOS命令下的COPY命令
- 在DOS状态下键入命令行，以实现将一个已存在文本文件中的内容全部拷贝到新文本文件中去
- 利用文本编辑软件，通过查看文件内容验证程序执行结果。

C语言的文件操作实例-源码1

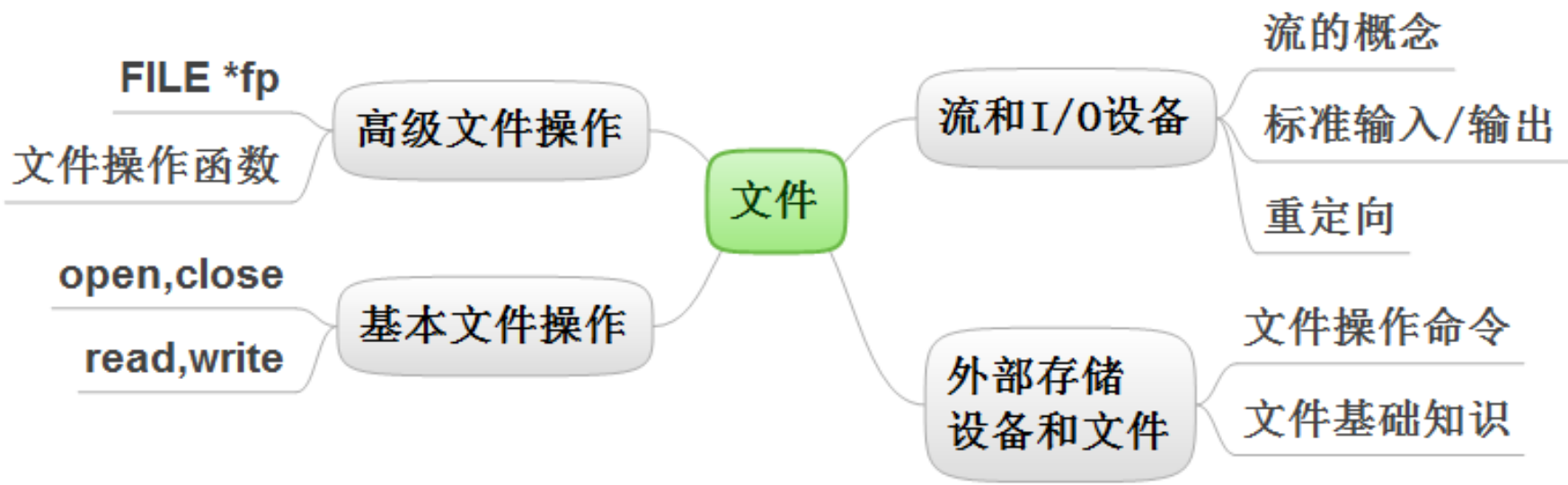
```
int CopyFile(const char* srcName,
 const char* dstName){
    FILE* fpSrc = NULL;
    FILE* fpDst = NULL;
    int ch, rval=1;
 fpSrc = fopen(srcName, "rb");
    if(fpSrc == NULL)
        goto ERROR;
 fpDst = fopen(dstName, "wb");
    if (fpDst == NULL)
        goto ERROR;
 while((ch=fgetc(fpSrc))!=EOF){
    if(fputc(ch, fpDst)==EOF)
        goto ERROR;
}
```

```
        fflush(fpDst);
        goto EXIT;
ERROR:
        rval=0;
EXIT:
        if(fpSrc != NULL)
            fclose(fpSrc);
        if(fpDst != NULL)
            fclose(fpDst);
        return rval;
```

C语言的文件操作-源码2

```
main(int argc, char *argv[])
{
    if (argc != 3){
        printf("too few parameters!\n");
        exit(0);
    }
    if (CopyFile(argv[1], argv[2])){
        printf("Copy succeed.\n");
    }
    else{
        perror("Copy failed");
    }
}
```

总结


文件
高级文件操作
FILE *fp
文件操作函数
基本文件操作
open,close
read,write
流和I/O设备
流的概念
标准输入/输出
重定向
外部存储
设备和文件
文件操作命令
文件基础知识

谢谢大家，欢迎提问！