

知识点1【文件的概述】（了解）

1、文件分类（存储介质）

磁盘文件：文件的数据 存放在磁盘上（音视频、图片文件、文档文件）

设备文件：通过系统将外部设备抽象文件

2、文件分类（存储方式）

任何磁盘文件 在物理上都是二进制存储。

逻辑上：磁盘文件分为二进制文件、文本文件

文本文件：基于字符编码的文件

二进制文件：基于值编码的文件

3、文本文件

基于字符编码，常见编码有 ASCII、UNICODE 等

一般可以使用文本编辑器直接打开

例如：数 5678 的以 ASCII 存储形式为：ASCII 码：00110101 00110110 00110111
00111000 歌词文件(lrc):文本文件

4、二进制码文件：

基于值编码,把内存中的数据原样输出到磁盘上

一般需要自己判断或使用特定软件分析数据格式

例如：数 5678 的存储形式为：二进制码：00010110 00101110

5、二进制和文件文件的区别

文本文件：

优点：一个字节一个意思、便于查看

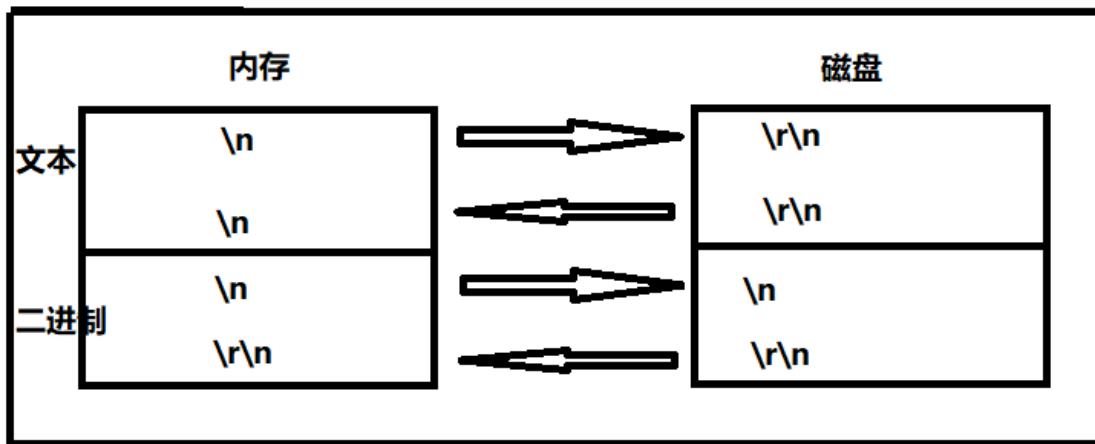
缺点：空间大、效率低

二进制文件：

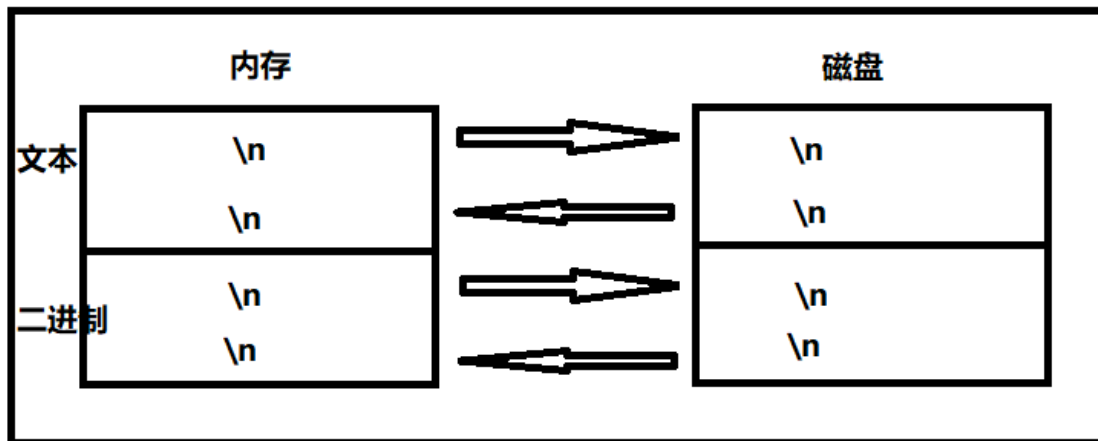
优点：效率高、空间小

缺点：不定长、不便于查看

windows

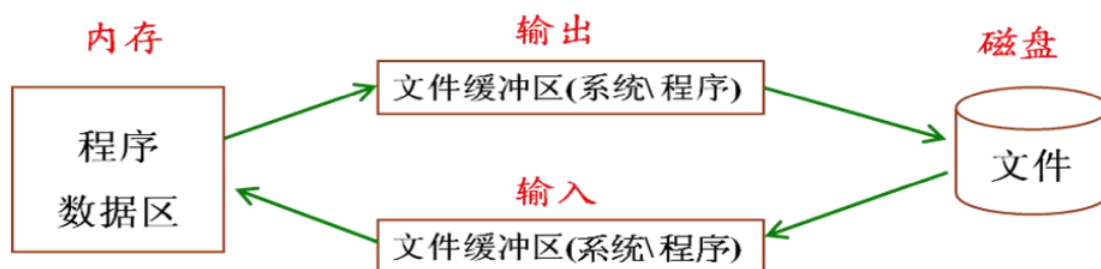


Linux



知识点2【文件缓冲区】（了解）

文件缓冲区的目的：提高访问效率 提高磁盘使用寿命



1、文件缓冲区的刷新方式

1、行刷新（遇到换行符 刷新）

```
void test01()
{
    printf("hello word\n");

    while (1);
}
```

2、满刷新（缓冲区数据放满 刷新）

```
int i = 0;
for (i = 0; i < 1000; i++)
{
    printf("123456789");
    usleep(20 * 1000);
}

while (1);
```

3、强制刷新（使用fflush函数 将缓冲刷新）

```
printf("123456789");
fflush(stdout);

while (1);
```

edu@edu: ~/work/c/day09

edu@edu: ~/work/c/day09\$ sudo gcc 00_code.c

edu@edu: ~/work/c/day09\$./a.out

123456789

4、关闭要新（关闭文件的时候 将缓冲区数据 全部刷新）

```
#include <stdio.h>
#include <unistd.h>
void test01()
{
    printf("123456789");
}
```

edu@edu: ~/work/c/day09

edu@edu: ~/work/c/day09\$ sudo gcc 00_code.c

edu@edu: ~/work/c/day09\$./a.out

123456789edu@edu: ~/work/c/day09\$

2、模拟时钟

```

void test01()
{
    int i = 0;
    while (1)
    {
        printf("\r%02d:%02d", i / 60, i % 60);
        fflush(stdout);
        sleep(1);
        i++;
    }
}

```

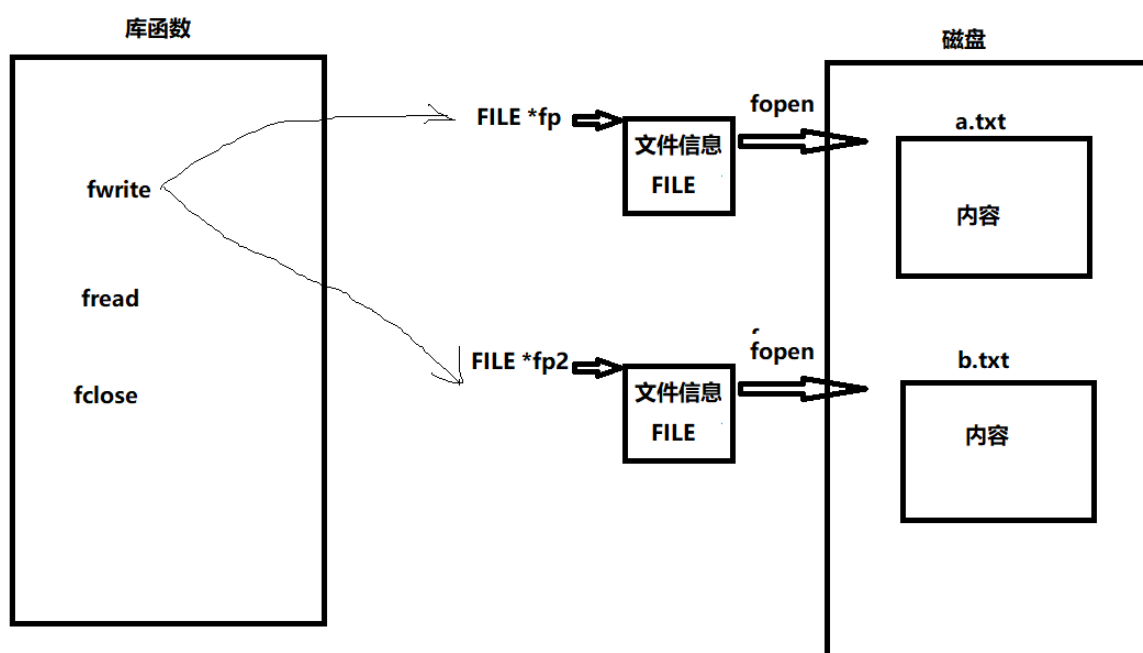
```

edu@edu: ~/work/c/day09
edu@edu: ~/work/c/day09$ sudo gcc 00_code.c
edu@edu: ~/work/c/day09$ ./a.out
00:11_

```

知识点3【文件指针】（重要）

所有操作文件的库函数 都需要借助 文件指针 操作文件。



FILE 在 **stdio.h** 文件中的文件类型声明:

```
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 charhold;     //如无缓冲区不读取字符
    short bsize;           //缓冲区的大小
    unsigned char *buffer; //数据缓冲区的位置
    unsigned ar*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;
```

为每一个进程 默认打开的3个文件指针

stdin: 标准输入 默认为当前终端（键盘）

我们使用的 `scanf`、`getchar` 函数默认从此终端获得数据

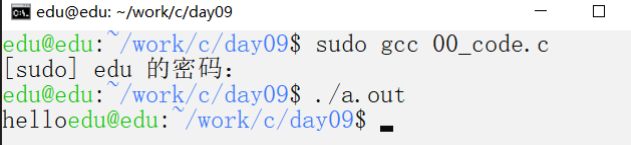
stdout: 标准输出 默认为当前终端（屏幕）

我们使用的 `printf`、`puts` 函数默认输出信息到此终端

stderr: 标准错误输出设备文件 默认为当前终端（屏幕）

当我们程序出错使用 `perror` 函数时信息打印在此终端

```
void test02()
{
    fwrite("hello", 6, 1, stdout);
}
```



知识点4 【文件的API】 (重要)

文件的操作步骤: 打开 读写 关闭

1、打开文件 `fopen`

函数的定义:

```
1 FILE *fopen(const char *path, const char *mode);
2 path: 文件的路径
3 mode: 打开文件方式
4 返回值:
5 成功: 就是打开的文件指针
```

文件的打开方式mode:

r w a + t b

r:只读的方式打开 w:只写的方式打开 a:追加的方式打开 +:可读可写方式打开

t:以文本文件方式打开（默认是省略）

b:以二进制方式打开（必须显示说明）

模 式	功 能
r 或 rb	以只读方式打开一个文本文件（不创建文件）
w 或 wb	以写方式打开文件（使文件长度截断为 0 字节，创建一个文件）
a 或 ab	以添加方式打开文件，即在末尾添加内容，当文件不存在时，创建文件用于写
r+或 rb+	以可读、可写的方式打开文件(不创建新文件)

w+或 wb+	以可读、可写的方式打开文件 （使文件长度为 0 字节，创建一个文件）
a+或 ab+	以添加方式打开文件，打开文件并在末尾更改文件（如果文件不存在，则创建文件）

2、关闭文件

```
1 int fclose(FILE *fp);
2 返回值:
3 成功返回 0
4 失败返回非 0
```

```

void test03()
{
    FILE *fp = NULL;
    fp = fopen("test.txt", "r");
    if (fp == NULL)
    {
        perror("fopen");
        return;
    }

    fclose(fp);
}

```

```

edu@edu: ~/work/c/day09
edu@edu: ~/work/c/day09$ sudo gcc 00_code.c
edu@edu: ~/work/c/day09$ ./a.out
fopen: No such file or directory
edu@edu: ~/work/c/day09$

```

3、一次读写一个字符

一次写一个字节: fputc

函数的定义:

```
int fputc(int c, FILE *stream)
```

函数的说明:

fputc 将 c 的值写到 stream 所代表的文件中。

返回值:

如果输出成功, 则返回输出的字节值;

如果输出失败, 则返回一个 EOF。

EOF 是在 stdio.h 文件中定义的符号常量, 值为-1

EOF只是在文本文件中有效。

```

1 void test04()
2 {
3     FILE *fp = NULL;
4     fp = fopen("test.txt", "w");
5     if (fp == NULL)
6     {
7         perror("fopen");
8         return;
9     }
10
11     char *file_data = "hello world";
12     while (*file_data != '\0')

```

```

13     {
14         fputc(*file_data, fp);
15         file_data++;
16     }
17
18     fclose(fp);
19 }

```

一次写一个字节: fputc

函数定义:

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

函数说明:

fputc 从 stream 所标示的文件中读取一个字节, 将字节值返回
返回值:

以 t 的方式: 读到文件结尾返回 EOF

以 b 的方式: 读到文件结尾, 使用 feof(后面会讲)判断结尾

```

void test05()
{
    FILE *fp = NULL;
    fp = fopen("test.txt", "r");
    if (fp == NULL)
    {
        perror("fopen");
        return;
    }

    while (1)
    {
        char ch = fgetc(fp);
        if (ch == EOF)
            break;
        printf("%c", ch);
    }

    fclose(fp);
}

```

```

edu@edu: ~/work/c/day10
edu@edu: ~/work/c/day10$ ./a.out
hello world
edu@edu: ~/work/c/day10$

```

3、一次读写一个字符串

写一个字符串

```
1 int fputs(const char *s, FILE *stream)
```


读取一个字符串(遇到换行符 结束) , 读取一行文件数据

```
1 char *fgets(char *s, int size, FILE *stream)
2 成功返回目的数组的首地址, 即 s
3 失败返回 NULL
```

```
1 void test06()
2 {
3     FILE *fp_r = fopen("test.txt", "r");
4     if (fp_r == NULL)
5     {
6         perror("fopen");
7         return;
8     }
9     FILE *fp_w = fopen("b.txt", "w");
10    if (fp_w == NULL)
11    {
12        perror("fopen");
13        return;
14    }
15
16    while (1)
17    {
18        char buf[128] = "";
19        //读一行
20        char *ret = fgets(buf, sizeof(buf), fp_r);
21        if (ret == NULL)
22            break;
23
24        //写一行
25        fputs(buf, fp_w);
26    }
27
28    fclose(fp_r);
29    fclose(fp_w);
30 }
```


```
edu@edu: ~/work/c/day10$ gcc 00_code.c
edu@edu: ~/work/c/day10$ ./a.out
edu@edu: ~/work/c/day10$ ls
00_code.c  a.out  b.txt  test.txt
edu@edu: ~/work/c/day10$ █
```

4、一次读写n块文件数据

块写：将内存的数据 原样的写入到 磁盘文件中

```
1 size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);
2 返回值： 实际写入的块数
```

```
1  #include <stdio.h>
2  typedef struct
3  {
4      char name[16];
5      int atk;
6      int def;
7  } HERO;
8  void test01()
9  {
10     HERO hero[] = {"德玛西亚", 50, 80}, {"小法", 70, 30}, {"盲僧", 80, 80}};
11     int n = sizeof(hero) / sizeof(hero[0]);
12
13     FILE *fp = fopen("hero.txt", "w");
14     if (fp == NULL)
15     {
16         perror("fopen");
17         return;
18     }
19
20     fwrite(hero, sizeof(HERO), n, fp);
21
22     fclose(fp);
23 }
24 int main(int argc, char const *argv[])
25 {
26     test01();
27     return 0;
```



将内存数据 原样输出 到磁盘
缺点：可读性差

快读：将磁盘数据 原样 输入到 内存

```
1 size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
2 返回值：
3 实际读到的整块数：不足一块 不计数 但是数据是读到的
```

```
1 #include <string.h>
2 void test02()
3 {
4     HERO hero[3];
5     memset(hero, 0, sizeof(hero));
6
7     FILE *fp = fopen("hero.txt", "r");
8     if (fp == NULL)
9     {
10         perror("fopen");
11         return;
12     }
13
14     fread(hero, sizeof(HERO), 3, fp);
15
16     int i = 0;
17     for (i = 0; i < 3; i++)
18     {
19         printf("%s %d %d\n", hero[i].name, hero[i].atk, hero[i].def);
20     }
21
22     fclose(fp);
23 }
```

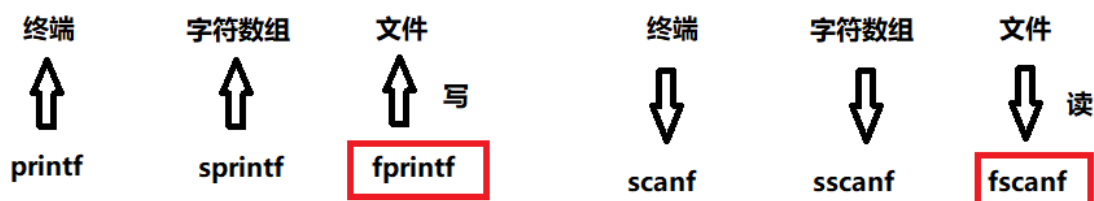
```

edu@edu: ~/work/c/day10
edu@edu: ~/work/c/day10$ gcc 01_code.c
edu@edu: ~/work/c/day10$ ./a.out
德玛西亚 50 80
小法 70 30
盲僧 80 80
edu@edu: ~/work/c/day10$

```

fread读到的数据 不便于查看

5、格式化读写



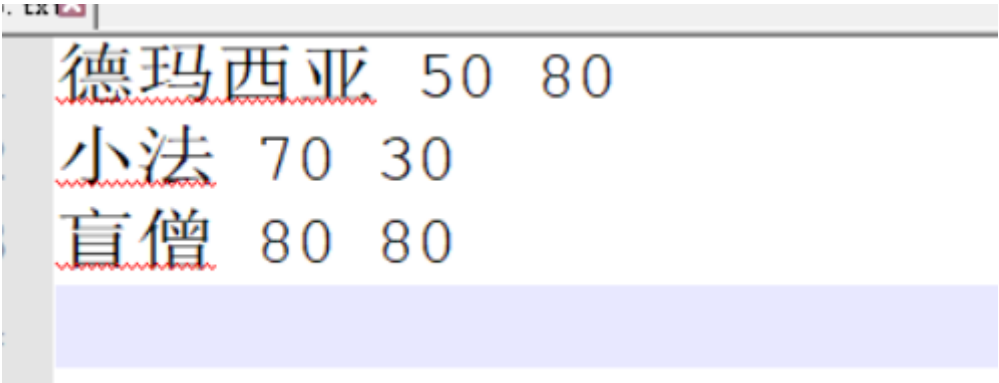
1、格式化写 fprintf

1 **fprintf** (文件指针, 格式字符串, 输出表列)

```

1 void test03()
2 {
3     HERO hero[] = {"德玛西亚", 50, 80}, {"小法", 70, 30}, {"盲僧", 80, 80};
4     int n = sizeof(hero) / sizeof(hero[0]);
5
6     FILE *fp = fopen("hero.txt", "w");
7     if (fp == NULL)
8     {
9         perror("fopen");
10        return;
11    }
12
13    //格式化写fprintf
14    int i = 0;
15    for (i = 0; i < n; i++)
16    {
17        fprintf(fp, "%s %d %d\n", hero[i].name, hero[i].atk, hero[i].def);
18    }
19
20    fclose(fp);

```



```
德玛西亚 50 80
小法 70 30
盲僧 80 80
```

2、格式化读 fscanf

```
1 void test04()
2 {
3     HERO hero[3];
4     memset(hero, 0, sizeof(hero));
5
6     FILE *fp = fopen("hero.txt", "r");
7     if (fp == NULL)
8     {
9         perror("fopen");
10        return;
11    }
12
13    int i = 0;
14    for (i = 0; i < 3; i++)
15    {
16        fscanf(fp, "%s %d %d", hero[i].name, &hero[i].atk, &hero[i].def);
17    }
18
19    for (i = 0; i < 3; i++)
20    {
21        printf("%s %d %d\n", hero[i].name, hero[i].atk, hero[i].def);
22    }
23
24    fclose(fp);
25 }
```

```
edu@edu: ~/work/c/day10$ gcc 01_code.c
edu@edu: ~/work/c/day10$ ./a.out
德玛西亚 50 80
小法 70 30
盲僧 80 80
edu@edu: ~/work/c/day10$
```

fprintf fscanf成对使用 效率低 阅读性高

fwrite fread成对使用 效率高 阅读性低

知识点5 【文件的随机读写】（了解）

文件默认是顺序读写：读写才能移动流指针 用户不能修改

随机读写：用户可以更改文件流指针的位置

1、引入案例

```
1 void test05()
2 {
3     FILE *fp = fopen("c.txt", "w+");
4     if (fp == NULL)
5     {
6         perror("fopen");
7         return;
8     }
9
10    //写
11    fputs("hello file", fp);
12    fclose(fp);
13
14    fp = fopen("c.txt", "r");
15    //读
16    char buf[128] = "";
17    fgets(buf, sizeof(buf), fp);
18    printf("读到的数据%s\n", buf);
19
20    fclose(fp);
21 }
```

```
edu@edu: ~/work/c/day10$ gcc 01_code.c
edu@edu: ~/work/c/day10$ ./a.out
读到的数据hello file
edu@edu: ~/work/c/day10$ _
```

2、随机读写的API

fseek rewind ftell

1、rewind复位文件流指针

```
1 void rewind(FILE *stream);
```

```
void test05()
{
    FILE *fp = fopen("c.txt", "w+");
    if (fp == NULL)
    {
        perror("fopen");
        return;
    }

    //写
    fputs("hello file", fp);

    //复位文件流指针
    rewind(fp);

    //读
    char buf[128] = "";
    fgets(buf, sizeof(buf), fp);
    printf("读到的数据%s\n", buf);

    fclose(fp);
}
```

```
edu@edu: ~/work/c/day10
edu@edu: ~/work/c/day10$ gcc 01_code.c
edu@edu: ~/work/c/day10$ ./a.out
读到的数据hello file
edu@edu: ~/work/c/day10$
```

2、ftell返回文件流指针 距离文件首部的 字节数

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

```

void test05()
{
    FILE *fp = fopen("c.txt", "w+");
    if (fp == NULL)
    {
        perror("fopen");
        return;
    }

    // 写
    fputs("hello file", fp);

    long len = ftell(fp);
    printf("文件的大小:%ld\n", len);

    fclose(fp);
}

```

```

edu@edu: ~/work/c/day10
edu@edu: ~/work/c/day10$ gcc 01_code.c
edu@edu: ~/work/c/day10$ ./a.out
文件的大小:10
edu@edu: ~/work/c/day10$

```

3、fseek文件流指针定位

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

- 1 whence 起始位置
- 2 文件开头 SEEK_SET 0
- 3 文件当前位置 SEEK_CUR 1
- 4 文件末尾 SEEK_END 2
- 5 offset: 流指针的偏移量
- 6 如果为正则数 向右偏移（单位为字节数） 为负数就是向左偏移

案例1：一次性读取文件数据

- 1 //将文件流指针定位到尾部
- 2 //ftell返回文件流指针的偏移量 （文件总大小len）
- 3 //根据文件总大小 从堆区申请空间 len+1
- 4 //使用fread一次性将文件数据读取

```

1 #include <stdlib.h>
2 #include <string.h>
3 void test06()
4 {
5     FILE *fp = fopen("c.txt", "r");
6     if (fp == NULL)
7     {
8         perror("fopen");
9         return;

```