

# 知识点 1【文件的概述】

## 1、文件的分类

**磁盘文件：**（我们通常认识的文件）

指一组相关数据的有序集合,通常存储在外部介质(如磁盘)上，使用时才调入内存。

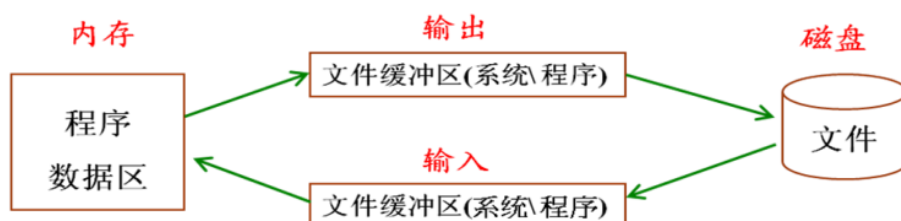
**设备文件：**

在操作系统中把每一个与主机相连的输入、输出设备看作是一个文件，把它们的输入、输出等同于对磁盘文件的读和写。

键盘：标准输入文件 屏幕：标准输出文件

## 2、文件的缓冲区。

缓冲区的目的：提高访问效率



## 3、缓冲区的刷新方式（库函数）。

行刷新：缓冲区遇到\n 将内容刷新 设备

满刷新：缓冲区放满数据，将内容刷新 设备

强制刷新： `fflush(stdout)` 强制将内容刷新 设备

关闭刷新：关闭文件或进程 将内容刷新 设备

## 4、磁盘文件的分类（背）

不管任何文件 在物理上都是二进制存储。

从逻辑上将磁盘文件分为二进制文件、文本文件。

## 1、文本文件

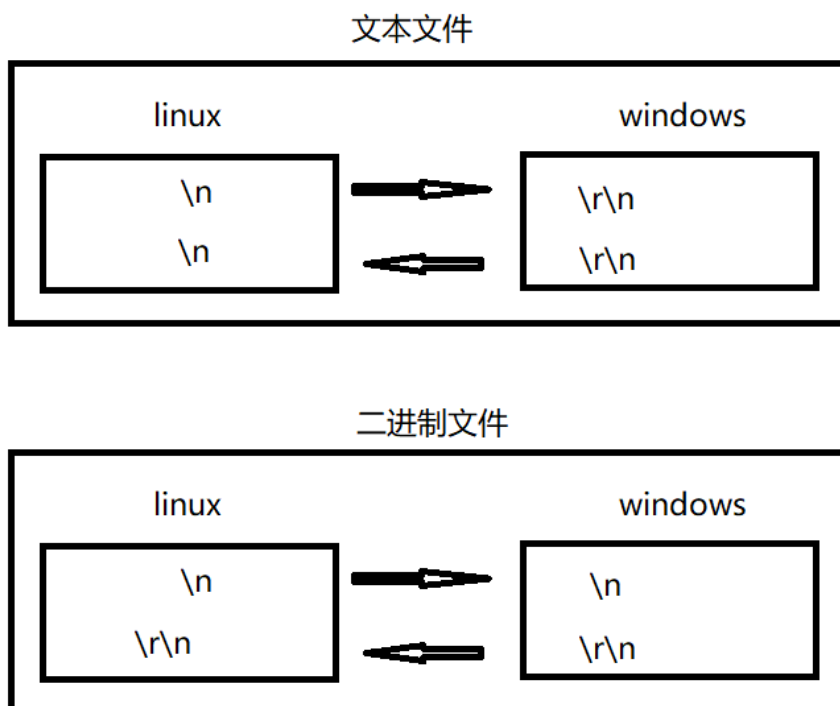
基于字符编码 存储的是每个字符的 ASCII 值 一个字节一个意思 便于查看，所占空间大，效率低。

例如：数 5678 的以 ASCII 存储形式为 00110101 00110110 00110111 00111000

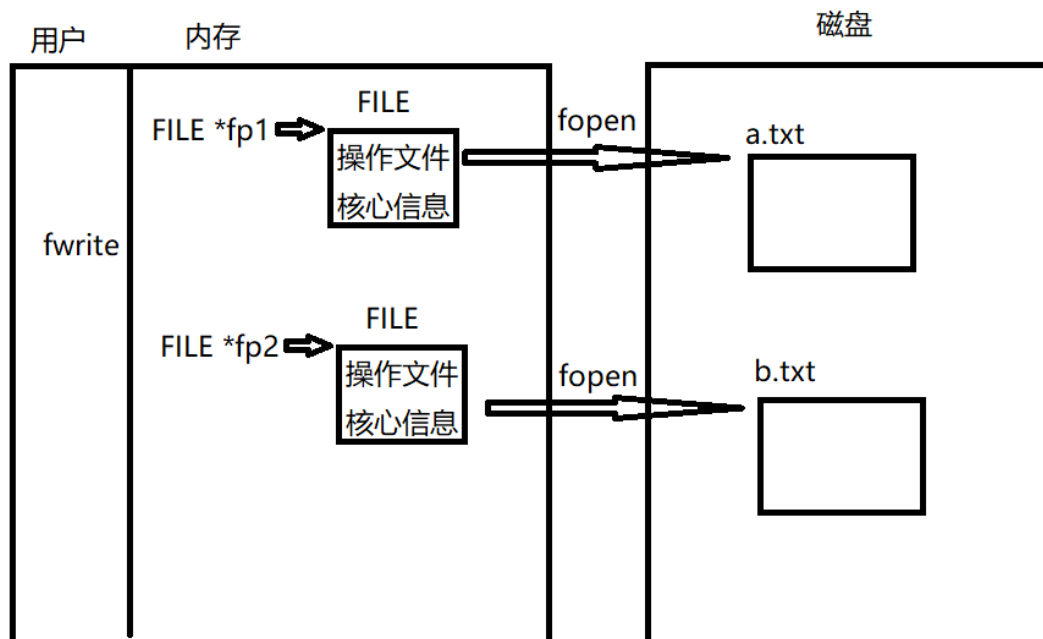
## 2、二进制文件

基于值编码 将数据的内存的二进制 直接放入磁盘 字节数不确定 二进制文件一般需要特定的软件才能打开。不便于查看 空间小 效率高。

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



## 知识点 2【文件指针】



`FILE *` 指针变量标识符。

`FILE` 是系统使用 `typedef` 定义出来的有关文件信息的一种结构体类型，结构中含有文件名、文件状态和文件当前位置等信息。

```
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
    shorttoken;            //用于有效性检查
} FILE;
```

# 知识点 3【文件的 API】

## 1、fopen 打开文件

```
#include <stdio.h>
```

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

功能：打开 path 指定的文件 通过返回值得到文件指针

参数 path:文件名的路径

参数 mode:打开文件的操作权限

返回值：

成功：得到文件指针

失败：NULL

打开方式 mode 的详解：r w a + t b

r 只读 w 只写 a 追加 +可读可写 t 打开文件文件（默认省略） b 打开二进制文件必

须加 b

模 式	功 能
<b>r 或 rb</b>	以只读方式打开一个文本文件（不创建文件）
<b>w 或 wb</b>	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）
<b>a 或 ab</b>	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
<b>r+或 rb+</b>	以可读、可写的方式打开文件(不创建新文件)

<b>w+或 wb+</b>	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）
<b>a+或 ab+</b>	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）

```
FILE *fp = fopen("a.txt", "r");

if (fp == NULL)

{

    perror("fopen");

    return;

}
```

```
printf("打开成功\n");
```

## 2、fclose 关闭文件

```
#include <stdio.h>
```

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

## 3、逐个字符读写 (fputc 写 fgetc 读)

### 3.1 fputc 一次写一个字符

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

功能：将字符 c 写入 stream 代表的文件

参数 c:需要写入的字符

参数 stream: 被写的文件指针

返回值:

成功：返回写入的字符

失败：-1

```
void test01()
```

```
{
```

```
    FILE *fp = fopen("a.txt", "w");
```

```
    if (fp == NULL)
```

```
    {
```

```
        perror("fopen");
```

```
        return;

    }

    char *text = "hello BK2302";

    while (*text)

    {

        fputc(*text, fp);

        text++;

    }

    fclose(fp);
}void test01()
{

    FILE *fp = fopen("a.txt", "w");

    if (fp == NULL)

    {

        perror("fopen");

        return;

    }

    char *text = "hello BK2302";
```

```
while (*text)

{

    fputc(*text, fp);

    text++;

}


fclose(fp);

}
```

### 3.2 fgetc 从文件读取一个字节

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

功能：从 stream 指向的文件 读取一个字节 通过返回值返回

参数 stream:代表读取的文件

返回值：

成功：返回读取的字符

失败：读取到文件末尾返回 EOF

```
void test02()

{

    FILE *fp = fopen("a.txt", "r");

    if (fp == NULL)

    {
```



```
perror("fopen");

return;

}

char buf[128] = "";

int i = 0;

while (1)

{

    char ch = fgetc(fp);

    if (ch == EOF) // EOF 代表文件末尾 只有文本文件才有 EOF

        break;

    buf[i] = ch;

    i++;

}

printf("%s\n", buf);

fclose(fp);

}
```

## 4、字符串读写 fgets fputs

### 4.1 fputs 写入一个字符串

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

功能：将 s 指向的字符串 写入 stream 指向的文件中（遇到\0 结束输入）

参数 s:指向被写入的字符串的首元素的地址

参数 stream: 指向的是文件

返回值:

成功：非负数

失败：-1

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

    char *text = "hello\0BK2302 good";

    int len = fputs(text, fp);

    printf("len=%d\n", len);

    fclose(fp);
}
```

## 4.2 fgets 读取一个字符串

```
char *fgets(char *s, int size, FILE *stream);
```

功能:从 stream 文件中读取一个字符串 存入 s 指向内存中, 读取的字节数最多为 size-1

功能: 读取文件一行 (遇到换行符结束 包含换行符)

参数 s:指向内存的起始地址

参数 size: 表示一次最多读取 size-1 个字符

参数 stream: 指向读取的文件

返回值:

成功: 返回 s 指向的地址

失败: NULL

```
void test04()
```

```
{
```

```
    FILE *fp = fopen("a.txt", "r");
```

```
    if (fp == NULL)
```

```
    {
```

```
        perror("fopen");
```

```
        return;
```

```
    }
```

```
    while (1)
```

```
    {
```

```
        char buf[128] = "";
```

```
char *ret = fgets(buf, sizeof(buf), fp);

if (ret == NULL)

    break;

printf("%s\n", buf);

}

fclose(fp);

}
```

**案例 1：假如 a.txt 有内容 将 a.txt 的内容读取 并写入到 b.txt 中**

## 5、块读写 fread 块读 fwrite 块写

### 1、fwrite 可以写入多块数据

```
size_t fwrite(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

功能：将 ptr 指定的内存 按照每块 size 大小总共写 nmemb 块的数据 写入 stream 指向的文件

注意：将内存数据原样 搬运到 磁盘

参数 ptr：内存的起始地址

参数 size：每一块的大小

参数 nmemb：代表的是块数

参数 stream：指向写入的文件

返回值：

成功：返回实际写入的块数

失败：返回 0

```
void test05()
{
    struct hero arr[5] = {"德玛", 50, 80}, {"小法", 70, 50}, {"小炮", 90, 60}, {"盲僧", 80,
80}, {"提莫", 90, 80}};

    FILE *fp = fopen("hero.txt", "w");

    if (fp == NULL)
    {
        perror("fopen");

        return;
    }

    // 将结构体数组的内容写入文件

    fwrite(arr, sizeof(struct hero), 5, fp);

    fclose(fp);
}
```

## 2、fread 块读

```
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

功能：将 ptr 指定的内存 按照每块 size\*块数 nmemb 的字节数从 stream 执行文件中读

取数据

注意：将磁盘数据原样 搬运到内存

参数 ptr：内存的起始地址

参数 size:每一块的大小

参数 nmemb：代表的是块数

参数 stream：指向写入的文件

返回值：

成功：返回实际读到的块数（显示整块数）

失败：返回 0

```
void test06()
{
    struct hero arr[5];

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

    // 将结构体数组的内容写入文件
```

```

size_t ret = fread(arr, sizeof(struct hero), 10, fp);

printf("ret=%d\n", ret);


int i = 0;

for (i = 0; i < 5; i++)

{

    printf("%s %d %d\n", arr[i].name, arr[i].atk, arr[i].def);

}


fclose(fp);
}

```

## 6、格式化读写 fprintf 写 fscanf 读

### 1、格式化写 fprintf

fprintf ( 文件指针, 格式字符串, 输出表列) ;

```

void test08()

{

    struct hero arr[5] = {"德玛", 50, 80}, {"小法", 70, 50}, {"小炮", 90, 60}, {"盲僧", 80, 80}, {"提莫", 90, 80};


    FILE *fp = fopen("hero.txt", "w");

    if (fp == NULL)

```

```

{
    perror("fopen");

    return;
}

int i = 0;

for (i = 0; i < 5; i++)
{
    fprintf(fp, "英雄:%s 攻击:%d 防御:%d\n", arr[i].name, arr[i].atk, arr[i].def);
}

fclose(fp);
}

```

```

1  英雄:德玛 攻击:50 防御:80
2  英雄:小法 攻击:70 防御:50
3  英雄:小炮 攻击:90 防御:60
4  英雄:盲僧 攻击:80 防御:80
5  英雄:提莫 攻击:90 防御:80
6

```

## 2、格式化读 fscanf

```
void test09()
```



```
{

    struct hero arr[5];

    FILE *fp = fopen("hero.txt", "r");

    if (fp == NULL)

    {

        perror("fopen");

        return;

    }

    // 将结构体数组的内容写入文件

    int i = 0;

    for (i = 0; i < 5; i++)

    {

        fscanf(fp, "英雄:%s 攻击:%d 防御:%d\n", arr[i].name, &arr[i].atk, &arr[i].def);

    }

    for (i = 0; i < 5; i++)

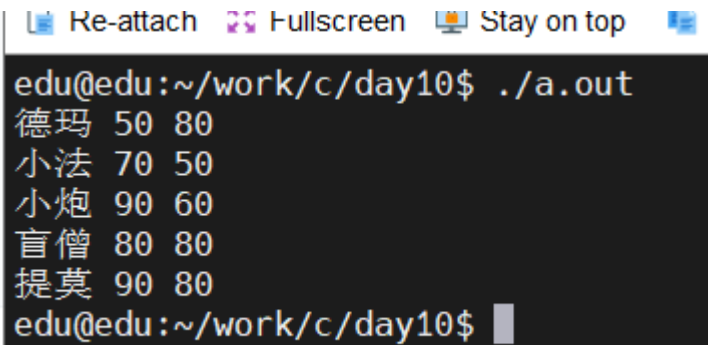
    {

        printf("%s %d %d\n", arr[i].name, arr[i].atk, arr[i].def);

    }

}
```

```
fclose(fp);  
}
```



A terminal window with a dark background and light-colored text. The window has a title bar with icons for 'Re-attach', 'Fullscreen', 'Stay on top', and a close button. The terminal shows the command `./a.out` being executed, followed by five lines of output: '德玛 50 80', '小法 70 50', '小炮 90 60', '盲僧 80 80', and '提莫 90 80'. The prompt `edu@edu:~/work/c/day10$` is visible at the bottom.

```
edu@edu:~/work/c/day10$ ./a.out  
德玛 50 80  
小法 70 50  
小炮 90 60  
盲僧 80 80  
提莫 90 80  
edu@edu:~/work/c/day10$
```

## 7、文件读写总结

`fopen` 打开文件得到文件指针 操作完文件记得 `fclose` 关闭文件

一个字节读写 `fgetc` `fputc`

一个字符串读写 `fgets` `fputs`

块的读写 `fread` `fwrite` 内存和磁盘原样数据输入或输出 效率高 不便于直接打开查看

格式化读写 `fscanf` `fprintf` 效率低 但是方便直接查看

## 知识点 4【随机读写】

### 1、`rewind` 复位读写位置

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

复位文件流指针 将文件流指针 指向文件的开始位置

```

186 void test10()
187 {
188     FILE *fp = fopen("a.txt", "w+");
189
190     char buf1[128] = "hello world1";
191     fputs(buf1, fp);
192
193     // 复位文件流指针
194     rewind(fp);
195
196     char buf[128] = "";
197     fgets(buf, sizeof(buf), fp);
198     printf("buf=%s\n", buf);
199
200     fclose(fp);
201 }

```

```

2.10.9.42.114
Re-attach Fullscreen Stay on top Duplicate
edu@edu:~/work/c/day10$ ./a.out
buf=hello world1
edu@edu:~/work/c/day10$

```

## 2、ftell 测文件读写位置距文件开始有多少个字节

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

```

186 void test10()
187 {
188     FILE *fp = fopen("a.txt", "w+");
189
190     char buf1[128] = "hello world1";
191     fputs(buf1, fp);
192
193     int len = ftell(fp);
194     printf("len=%d\n", len);
195
196     fclose(fp);
197 }

```

```

2.10.9.42.114
Re-attach Fullscreen Stay on top Duplicate
edu@edu:~/work/c/day10$ ./a.out
len=12
edu@edu:~/work/c/day10$

```

## 3、fseek 定位位置指针（读写位置）

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

whence 起始位置

文件开头 SEEK_SET	0
文件当前位置 SEEK_CUR	1
文件末尾 SEEK_END	2

案例 1：将 fp 定位到文件尾部:

```
fseek(fp,0,2);
```

## 案例 2：一次性读取文件的数据

```
void test11()
{
    FILE *fp = fopen("a.txt", "r");

    if (NULL == fp)
    {
        perror("fopen");

        return;
    }

    // 得到文件总大小

    // 将文件指针定位文件的尾部

    fseek(fp, 0, 2);

    // 获取文件长度

    int len = ftell(fp);

    // 复位文件流指针

    rewind(fp);

    // 根据文件总大小申请堆区空间

    char *msg = (char *)malloc(len + 1);

    // 读取数据
```

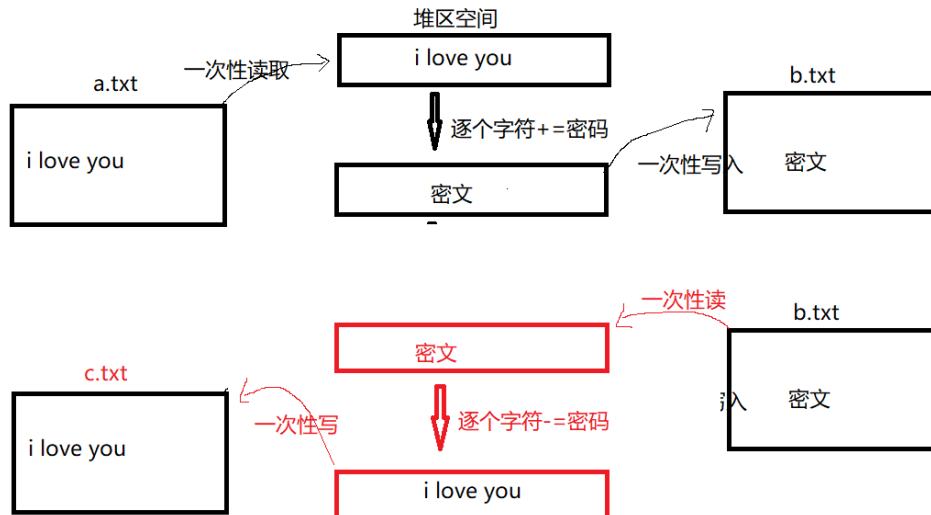
}





[illegible]

## 知识点 5【文件加密器】



```
#include <stdio.h>

#include <stdlib.h>

void get_file_name(char *dest_file_name, char *src_file_name);

char *read_src_file(unsigned long int *file_length, char *src_file_name);

char *file_text_encrypt(char *src_file_text, unsigned long int length, unsigned int
password);

void save_file(char *text, unsigned long int length, char *file_name);

char *file_text_decrypt(char *src_file_text, unsigned long int length, unsigned int
password);

void help()
{

printf("*****1:加密文件*****\n");
```

```
printf("*****2:解密文件*****\n");

printf("*****3:退出程序*****\n");

}

int main(int argc, char const *argv[])
{
    while (1)
    {
        char src_file[31] = "";

        char dst_file[31] = "";

        unsigned long int file_len = 0;

        char *data = NULL;

        unsigned int passwd = 0;


        help();

        int cmd = 0;

        scanf("%d", &cmd);


        switch (cmd)
        {

        case 1:

            get_file_name(dst_file, src_file);

            data = read_src_file(&file_len, src_file);
```

```
    printf("please input your unsigned int passworld:");

    scanf("%d", &passwd);

    data = file_text_encrypt(data, file_len, passwd);

    save_file(data, file_len, dst_file);

    break;

case 2:

    get_file_name(dst_file, src_file);

    data = read_src_file(&file_len, src_file);

    printf("please input your unsigned int passworld:");

    scanf("%d", &passwd);

    data = file_text_decrypt(data, file_len, passwd);

    save_file(data, file_len, dst_file);

    break;

case 3:

    return 0;

default:

    break;

}

}

return 0;
```



```

}

/*****/

// 函数功能:获取 目的文件和源文件的名字

// 参数:

// src_file_name:源文件名字字符数组首地址。

// dest_file_name:目的文件的名字字符数组首地址

/*****/

void get_file_name(char *dest_file_name, char *src_file_name)

{

    printf("请输入你的原文件名称(30 个字符):");

    scanf("%s", src_file_name);

    printf("请输入你的目的文件名称(30 个字符):");

    scanf("%s", dest_file_name);

    return;

}

/*****/

// 函数功能:读出文件内容

// 参数: file_length:整型指针, 此地址中保存文件字节数。

// src_file_name:文件名字, 从此文件中读取内容。

// 返回值:读出字符串的首地址

// 在此函数中测文件的大小, 并 malloc 空间, 再把文件内容读出返回, 读出字符数组的
首地址

```

```
/******
```

```
char *read_src_file(unsigned long int *file_length, char *src_file_name)
```

```
{
```

```
    FILE *fp = fopen(src_file_name, "r");
```

```
    if (NULL == fp)
```

```
    {
```

```
        perror("fopen");
```

```
        return NULL;
```

```
    }
```

```
    fseek(fp, 0, 2);
```

```
    *file_length = ftell(fp);
```

```
    rewind(fp);
```

```
    char *p = (char *)malloc(*file_length + 1);
```

```
    if (NULL == p)
```

```
    {
```

```
        perror("malloc");
```

```
        return NULL;
```

```
    }
```

```
    fread(p, *file_length, 1, fp);
```

```

fclose(fp);

return p;
}

/*****

// 函数功能:加密字符串

// 参数:

// src_file_text:要加密的字符串。 length:字符串的长度

// password: 加密密码

// 返回值: 加密后的字符串的首地址

// 加密原理字符数组中每个元素加上 password

*****/

char *file_text_encrypt(char *src_file_text, unsigned long int length, unsigned int
password)
{
    int i = 0;

    for (i = 0; i < length; i++)
    {
        src_file_text[i] += password;
    }
}

```

```

    return src_file_text;
}

// 函数功能:将字符串保存到目的文件中

// 参数:

// text:要保存的字符串首地址

// file_name :目的文件的名字

// length:字符串的长度

// 思想: 传入字符数组的首地址和数组的大小及保存后的文件的名字, 即可保存数组到文件中

/*****/

void save_file(char *text, unsigned long int length, char *file_name)
{
    FILE *fp = fopen(file_name, "w");

    if (NULL == fp)
    {
        perror("fopen");

        return;
    }

    fwrite(text, length, 1, fp);

```

```

fclose(fp);

if (text != NULL)

{

    free(text);

    text = NULL;

}

printf("save sucess\n");

return;

}

/*****/

// 函数功能:解密字符串

// 参数:

// src_file_text:要解密的字符串。 length:字符串的长度

// password: 解密密码

// 返回值: 解密后的字符串的首地址

// 思想:把数组中的每个元素减去 password 给自己赋值。

/*****/

char *file_text_decrypt(char *src_file_text, unsigned long int length, unsigned int
password)

{

    int i = 0;

```

```
for (i = 0; i < length; i++)  
  
    {  
  
        src_file_text[i] -= password;  
  
    }  
  
    return src_file_text;  
}
```