



§ 15. 输入输出流

相关内容:

第03模块:

3. 4. C++的输入与输出

第08模块: 输入输出流

8. 1. C++的输入与输出

8. 2. 标准输入流

8. 3. 标准输出流

8. 4. 文件操作与文件流

补充:

★ cin提取数据后, 会根据数据类型是否符合要求而返回逻辑值

- 当cin返回为1/true时, 读入的值才可信

=>正确的处理逻辑: cin读入后, 先判断cin, 为1再取值

- 不同编译器, cin为0时, a的值可能不同(不可信)

- 还可以用cin.good()/cin.fail()来判断

注意: cin.good()与cin.fail()不是任何时候都互斥的!

右例: 观察什么时候in.good()和in.fail()同时为0!!!

```
#include <iostream>
#include <fstream>
using namespace std;
```

```
int main()
{
```

```
    ifstream in("a.txt", ios::in);
    char buf[128];
```

```
    if (in.is_open() == 0) {
        cout << "无法打开文件" << endl;
        return -1;
    }
```

```
    while (1) {
```

```
        in.getline(buf, sizeof(buf));
```

```
        cout << bool(in) << ' ' << in.good() << ' ' << in.fail() << endl;
```

```
        if (in.eof())
```

```
            break;
```

```
        cout << '*' << buf << '*' << endl;
```

```
    }
```

```
    in.close();
```

```
    return 0;
```

```
}
```

另: 建立a.txt文件, 两行

1234567 张三

7654321 李四

注: 如果是记事本编辑, 记得编码为ANSI

分两种情况运行:

最后一行后有换行/无换行



§. C语言的文件操作

1. 文件指针

FILE *文件指针变量

- ★ FILE是系统定义的结构体
- ★ C语言中文件操作的基本依据，所有针对文件的操作均要依据该指针
- ★ #include <stdio.h> (VS可以不需要)
- ★ VS以为不安全，需要加 #define _CRT_SECURE_NO_WARNINGS
- ★ 文件读写后，文件指针会自动后移



§. C语言的文件操作

2. 文件的打开与关闭

假设: `FILE *fp`定义一个文件指针

2.1. 文件的打开

`FILE *fopen(文件名, 打开方式)`

`fp = fopen("test.dat", "r");`

`fp = fopen("c:\\demo\\test.dat", "w");` //也可表示为: `"c:/demo/test.dat"`

★ 打开的基本方式如下:

r: 只读方式

w: 只写方式

a: 追加方式

+: 可读可写

b: 二进制

t: 文本方式(缺省)

★ 打开的基本方式及组合见右表

★ 若带路径, 则\必须用\\表示(也可以/)

★ 若打开不成功, 则返回NUL

打开方式	意义
r/rt	只读方式打开文本文件(不存在则失败)
w/wt	只写方式打开或建立文本文件(存在则清零)
a/at	追加写方式打开或建立文本文件(头读尾写)
rb	只读方式打开二进制文件(不存在则失败)
wb	只写打开或建立二进制文件(存在则清零)
ab	追加写方式打开或建立二进制文件(头读尾写)
r+/rt+	读写方式打开文本文件(不存在则失败)
w+/wt+	读写方式创建文本文件(存在则清零)
a+/at+	读+追加写方式打开或建立文本文件(头读尾写)
rb+	读写方式打开二进制文件(不存在则失败)
wb+	读写方式创建二进制文件(存在则清零)
ab+	读+追加写方式打开二进制文件(头读尾写)

2.2. 文件的关闭

`fclose(文件指针)`

`fclose(fp);`



§. C语言的文件操作

3. 文本文件的读写

3.1. 按字符读写文件

读: `int fgetc(文件指针)`

- 返回读到字符的ASCII码 (返回值同`getchar`)

写: `int fputc(字符常量/变量, 文件指针)`

- 返回写入字符的ASCII码 (返回值同`putchar`)

★ 必须保证文件的打开方式符合要求

```
char ch1;
ch1=fgetc(fp);

char ch2 = 'A';
fputc(ch2, fp);
```

3.2. 判断文件是否到达尾部

`int feof(文件指针)`

★ 若到达尾部, 返回1, 否则为0

3.3. 按格式读写文件

读: `int fscanf(文件指针, 格式串, 输入表列)`

- 返回读取正确的数量 (返回值同`scanf`)

写: `int fprintf(文件指针, 格式串, 输出表列)`

- 返回输出字符的个数 (返回值同`printf`)

★ 格式串、输入/输出表列的使用同`scanf/printf`

```
int i;
char ch;
fscanf(fp, "%d%c", &i, &ch);

int i=10;
char ch='A';
fprintf(fp, "%d%c", i, ch);
```



§. C语言的文件操作

3. 文本文件的读写

3.4. 用文件方式进行标准输入输出

stdin : 标准输入设备
stdout : 标准输出设备
stderr : 错误输出设备

这三个是系统预置的FILE *, 直接用, 不需要打开关闭

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

int main()
{
    int i;
    char ch;
    ch = fgetc(stdin);           ⇔ getchar();
    putchar(ch);
    fputc('A', stdout);         ⇔ putchar('A');
    fscanf(stdin, "%d", &i);     ⇔ scanf("%d", &i);
    fprintf(stdout, "i=%d\n", i); ⇔ printf("i=%d\n", i);
    fprintf(stderr, "i=%d\n", i); ⇔ cerr<<"i="<<i<< endl;

    return 0;
}
```

//C方式无专用错误输出
//perror() 功能不同



§. C语言的文件操作

3. 文本文件的读写

3.5. 用freopen重定向标准输入输出

★ FILE *freopen(文件名, 打开方式, 原FILE *);

功能: 将已存在的FILE *映射为另一个新的FILE *

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

int main()
{
    FILE *fp;
    if ((fp = freopen("out.txt", "w", stdout)) == NULL) {
        printf("freopen failed!\n");
        return -1;
    }
    printf("Hello, world!\n");
    fclose(fp);

    return 0;
}
```

- 1、正常运行, 观察运行结果
- 2、不删除已存在的out.txt, 换成"r", 观察运行结果
- 3、先删除已存在的out.txt, 换成"r", 观察运行结果
- 4、如果在fclose的后面再加printf, 能否正常输出?
如果可以, 输出到哪里了? 如果没有, 为什么?

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

int main()
{
    FILE *fp;
    int a, b;
    if ((fp = freopen("in.txt", "r", stdin)) == NULL) {
        printf("freopen failed!\n");
        return -1;
    }
    scanf("%d %d", &a, &b);
    printf("a=%d b=%d\n", a, b);
    fclose(fp);

    return 0;
}
```

- 1、在当前目录下建立in.txt文件, 写入两个整数, 观察运行结果
- 2、在当前目录下没有/删除in.txt的情况下运行, 观察运行结果
- 3、如果在fclose的后面再加scanf, 能否正常输入? 如果可以, 从哪里读? 如果不行, 为什么?



§. C语言的文件操作

3. 文本文件的读写

3.5. 用freopen重定向标准输入输出

- ★ FILE *freopen(文件名, 打开方式, 原FILE *);
功能: 将已存在的FILE *映射为另一个新的FILE *
- ★ 用freopen可以重定向普通文件 (一般不用)

观察运行结果

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

int main()
{
    FILE *fp, *fdup;
    if ((fp = fopen("out.txt", "w")) == NULL) {
        printf("fopen out.txt failed!\n");
        return -1;
    }
    fprintf(fp, "Hello, world!\n");

    if ((fdup = freopen("out_dup.txt", "w", fp)) == NULL) {
        printf("freopen failed!\n");
        fclose(fp);
        return -1;
    }
    fprintf(fp, "I am a student.\n"); //注意: 不是fdup !!!

    fclose(fp);
    fclose(fdup);
    return 0;
}
```



§. C语言的文件操作

3. 文本文件的读写

3.6. 用popen/pclose与系统命令进行交互

★ VS下是_popen与_pclose

★ Linux下的popen与pclose

★ Dev C++下popen/pclose/_popen/_pclose均可

★ Windows示例(分两步操作)

```
//假设编译为 D:\VS-Demo\Debug\demo-cpp.exe
#include <iostream>
using namespace std;
int main()
{
    cout << "Welcome to Tongji University!" << endl;
    return 0;
}

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main()
{
    FILE* fp = _popen("D:\\VS-Demo\\Debug\\demo-cpp.exe", "r");
    if (fp == NULL) {
        printf("popen failed!\n");
        return -1;
    }
    char ch;
    while (1) {
        ch = fgetc(fp);
        if (feof(fp);
            break;
        putchar(ch);
    }
    _pclose(fp);
    return 0;
}
```

Step1

Step2

再换为 "dir C:\Windows"

★ Linux示例(分两步操作)

```
//假设编译为 /home/u1234567/test
#include <iostream>
using namespace std;
int main()
{
    cout << "Welcome to Tongji University!" << endl;
    return 0;
}

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main()
{
    FILE* fp;
    if ((fp = popen("/home/u1234567/test", "r")) == NULL) {
        printf("popen failed!\n");
        return -1;
    }
    char ch;
    while (1) {
        ch = fgetc(fp);
        if (feof(fp))
            break;
        putchar(ch);
    }
    pclose(fp);
    return 0;
}
```

Step1

Step2

再换为 "ls -l /etc"



§. C语言的文件操作

4. 二进制文件的读写

4.1. 按字符读写文件

读: `int fgetc(文件指针)`

- 返回读到字符的ASCII码 (返回值同`getchar`)

写: `int fputc(字符常量/变量, 文件指针)`

- 返回写入字符的ASCII码 (返回值同`putchar`)

★ 必须保证文件的打开方式符合要求

★ 同C++方式, 仅能按字符读写, 且文件中不能有0x1A

4.2. 按块读写文件

读: `int fread(缓冲区首址, 块大小, 块数, 文件指针)`

- ★ 返回读满的块数

写: `int fwrite(缓冲区首址, 块大小, 块数, 文件指针)`

- ★ 返回写入成功的块数



§. C语言的文件操作

4. 文件指针的移动

4.1. 指针复位(回到开头)

`rewind(文件指针)`

例: `rewind(fp);`

4.2. 任意移动

`fseek(文件指针, 位移量, 位移方式)`

例: `fseek(fp, 123, SEEK_SET):` 从开始移动

`fseek(fp, 78, SEEK_CUR):`
`fseek(fp, -25, SEEK_CUR):` } 从当前位置移动

`fseek(fp, -57, SEEK_END):` 从最后移动

★ `SEEK_SET`的位移必须为正

`SEEK_CUR`的位移可正可负

`SEEK_END`的位移必须为负

4.3. 求文件指针的当前位置

`long ftell(文件指针)`

例: `ftell(fp);`

★ 从开始位置计算