文件

在实际应用的程序中,某个程序的输入可能是另一个程序的输出,某个程序的数据也可能要永久保存.为了解决这些问题,C语言提供了对文件的操作,实现数据的保存,读取和自动输入

文件概述

数据流

数据流的输入输出必须通过计算机的外围设备,不同外围设备输入输出的格式与方法不同,这导致编写文件访问程序困难,而且容易产生不兼容问题.C语言将各种输入输出的终端设备,硬盘以及各种磁盘,串行口等都统一映射成**C语言逻辑层面的数据流**,只需按照标准I/O的提供的库函数对数据流进行操作就可以完成数据的输入与输出.
数据流将整个文件内的数据看作**一串连续的字符(字节)**,而且没有记录的限制.
数据流**借助文件指针的移动来访问数据**,文件指针目前所指的位置就是要处理的数据,访问后文件指针自动后移.
每个数据文件后面有一个**文件结束符号(EOF)**,用来告知该数据文件到此结束.

文件的概念

文件的定义

文件是指存储在外部介质上的**有界数据集合**,一批数据按照一定的组织架构以文件形式存放在硬盘,磁盘,光盘上,可以长期储存,多次使用.
操作系统对外部介质上的数据是以文件的形式管理的.当打开一个文件或者创建一个新文件时,一个数据流和一个外部文件相关联.

一个文件必须有文件名,它通常由一串ASCII码或者汉字组成,,用户利用文件名访问文件.
C语言支持流式文件,即数据流,它把文件看出看成一个字节序列,并以文件结束符结束.通过文件名确定这一组文件中第一个字节的物理地址,字节流中的每一个字节用位移量标识,第一个字节的位移量为0,后续字节位移量依次递增.可以根据该物理位置和位移量读写文件中任何一个字节或者任意位置开始的一组连续字符块.

文件的读写操作

文件的读写操作都以CPU和内存为参考点,读文件称为输入操作,他将文件中的数据读入内存供CPU处理.写文件操作称为输出操作,他将经过CPU处理且存储在内存中的结果数据写到文件中保存起来.

缓冲区

缓冲区(buffer)由称为缓存,是内存空间中的一部分,也就是说在内存空间中预留了一定的存储空间,这些存储空间用来缓冲输入输出的数据.他在输入输出设备和CPU之间缓冲数据,可以使低速输入输出的设备和高速运行的CPU之间协调工作,避免输入输出设备占用CPU.
缓冲区分为输入缓冲区和输出缓冲区.

当使用**标准I/O函数**时(包含在头文件stdio.h中),系统会自动设置缓冲区,并通过数据流来读写数据.文件读取时,不会直接对磁盘进行读取,而是先打开数据流,将磁盘上的文件信息复制到缓冲区,程序再从缓冲区读取所需数据.写入文件时,先写入缓冲区,缓冲区满或关闭文件使才会将数据写入磁盘.

文件类型

C语言根据流式文件中字节的编码方式,将文件分为ASCII文件和二进制文件两种.

ASCII文件

ASCII文件又称为**文本文件(text)**,在文本文件中,每个字节存放一个ASCII码值,代表一个字符,**单个字符的ASCII码是文本文件的基本组成单位**.
对文本文件的操作可以是单个字符的输入输出,字符串的输入输出,格式化的输入和输出.无论用什么方式对文本文件进行输入输出操作,数据都将按照写入文件的先后顺序以ASCII码存放.

```
/* 在Windows下的命令行窗口,输入以下命令可以生成一个名为text1.txt文件 */
E:\C2019\copy con text1.txt
Hello,world!
2004
11.05
^Z
```

通过磁盘编辑工具可以看到text1.txt文件在磁盘扇区中的存储格式,数据是按照正文输入的先后顺序以字符的ASCII码形式存放.

二进制文件

在二进制文件中,**每个字节都是对应数据在内存中存放时的表现形式**,也就是说,二进制文件是把内存中的数据按照其内存中的存储形式原样输出到磁盘上存放,不同数据有**不同的存储长度和不同的存储格式**.因此在读写二进制文件时,要注意读写数据的格式.
例如,定义结构体:

```
struct dat{
    char c[13];
    int x;
    float y;
} pt={"Hello world!",2004,11.05};
```

将这个结构体成员依次写入一个二进制文件text1.bin中.整型常量2004的二进制为0x00 00 07 D4,因此文件形式在磁盘扇区中的数据存储为0xD4 07 00 00,**低字节在前,高字节在后**.

可见,文本文件是一个字节存储一个字符,这样便于对字符进行处理,方便在文件编辑器中阅读和修改,但是文本文件占用存储空间较大,计算机处理数据时需要将ASCII码转换成二进制形式,会花费较多时间.
二进制文件中的数据的存储空间小,执行效率高,但是其可读性差.
C语言处理这些数据时,不区分类型,都视为字符流,按字节进行处理.

文件指针

在C语言中,FILE 结构类型是用于表示文件的数据结构.它定义在 `<stdio.h>` 头文件中.FILE 结构类型通常用于打开,关闭,读取和写入文件等文件操作.

这个结构体中**包含了与文件相关的信息**,例如文件的当前位置,文件状态以及用于读写文件的缓冲区等.C标准库提供了一系列库函数来操作 FILE 结构类型,例如 `fopen()` 用于打开文件, `fclose()` 用于关闭文件, `fread()` 和 `fwrite()` 用于读写文件内容等.

不同的C编译器关于 FILE 结构类型的定义会有差异,但是这仅仅会影响库函数的实现,对库函数的调用没有影响.以下是一个简化定义:

```
typedef struct _IO_FILE {
    int fd;           // 文件描述符
    char *buffer;     // 输入/输出缓冲区
    char *ptr;        // 当前指针位置
    size_t bufsize;   // 缓冲区大小
    size_t bytes;     // 当前缓冲区中已使用的字节数
    int flags;        // 文件状态标志
    // 其他成员可能包括文件位置指示器等
} FILE;
```

FILE类型的结构变量在打开文件时由系统创建,其成员的值的由系统更新,程序使用的只是指向FILE类型的结构变量指针,称为文件指针.每个打开的文件必须有一个指向FILE类型的结构变量的指针,通过相关的文件操作函数和文件指针对文件进行操作.
定义文件指针的一般形式为:

```
FILE *fp;
```

文件操作的基本步骤

1. **打开文件**.建立文件指针或文件描述符与新建文件或已有文件之间的联系.
2. **对文件进行读写操作**.
3. **关闭文件**.取消文件指针或文件描述符与已打开文件之间的连续,关闭文件保证将文件缓冲区数据写入文件,并释放系统分配的文件缓冲区.

文件的打开与关闭

打开文件函数fopen

打开一个文件必须**指定访问模式(access mode)**,表明计划对文件进行读还是写还是读写结合
头文件stdio.h中关于fopen的原型声明为:

```
FILE *__cdecl fopen(const char * filename, const char * mode);
```

__cdecl表示使用C语言约定,即调用时所有参数从右到左依次入栈,这些参数由调用者清除.fopen的第一个形参是文件名filename,第二个形参是文件的访问模式mode.如果文件成功打开,**fopen返回指向所打开文件的文件指针值**.

文件名

第一个形参filename指向的**字符串**就是待打开的文件名,它可以包括文件名,拓展名(扩展名用于表示文件的类型或格式),也可以包含驱动器名(标识不同存储设备或存储空间的符号或名称)和目录路径.

访问路径

第二个形参mode指定文件的访问模式,该**字符串**的第一个字符只能为三种形式:r,w,a(append).后面还可以包括+和b(没有次序限制).
访问模式中的加号表示读写操作都可以进行,但是程序不可以在读写操作中立即切换.写操作之后必须调用函数fflush或者定位函数(fseek,fsetpos或rewind),然后才可以执行读操作.在读操作之后,必须调用定位函数,然后才能执行写操作.模式字符串中的b表示文件以二进制文件打开,没有b,新建的流就是字符串流.

- 当访问模式字符串以r开始时,该文件必须**存在于文件系统中**.
- 当访问模式字符串以w开始时,如果文件不存在,会**新建一个文件**;如果文件存在该文件**当前内容会被清除**,因为在写模式中,函数fopen将文件长度设置为0.
- 当访问模式字符串以a开始时,如果文件不存在,则**建立一个新文件,从文件开始处写入内容**;如果文件存在,则**文件当前内容被保留,写入的内容会从文件尾部进行添加**.

- +:表示打开或创建的文件运行读写操作,也称为**更新操作**,即文件位置指针不在文件结尾时进行写操作将以**覆盖方式写**.
- r+和a+:二者的区别在于文件打开时,**文件指针的位置不同**.r+的文件指针的位置总位于文件首;a+写数据时文件的指针位置在文件尾,读数据时,文件指针的位置指针在文件首.

打开文件函数的说明

1. 如果文件以只读方式打开,则不能向文件写入数据,文件路径必须是正确的且指定文件存在;
2. 如果文件以只写方式打开,则不能从文件读入数据,如果文件不存在,会新建文件;如果文件存在,则清空文件;
3. 写入字节过程中写入的内容写入的内容覆盖位置指针指向的字节;读入文件过程中读入位置指针指向的字节内容.完成读写操作后,自动调整位置指针值.
4. **当位置指针指向文件尾时,读入文件内容为EOF,表示读文件操作出错**.EOF是stdio.h头文件中定义的符号常量.
5. 打开文件出错处理.==如果打开文件出错,可以终止程序,返回操作系统,也可以给予提示返回程序进一步处理.
6. 输入文本文件时,将回车换行符转化成一个换行符;输出时,将换行符转换成回车符和换行符两个字符.二进制文件不进行这种转换.
7. 在程序开始运行时,系统自动打开三个标准文件:**标准输入,标准输出,标准出错输出**.这三个文件都与终端相联系.系统自动定义了三个文件指针stdin,stout,stderr,分别指向终端输入,终端输出和标准出错输出.

关闭文件函数fclose

头文件中stdio.h中关于fclose的原型为:

```
int__cdecl fclose(FILE *);
```

参数为文件指针,fclose关闭文件指针所指的文件,它使缓冲区内尚未存盘的数据全部强制性存盘,释放打开文件时系统分配的输入输出缓冲区,取消FILE指针与文件的对应关系.正常关闭文件时,函数返回.

```
int main()
{
    FILE * fp;/* 定义文件指针 */
    char c[30];

    if((fp=fopen("D:\\C_code\\Notes\\_file_1.txt","w"))==NULL)/* 打开文件 */
    {
        printf("failed\n");
        return 0;
    }
    while(fgets(c,80,stdin)!=NULL)/* 输入字符串直到Ctrl+z */
        fprintf(fp,"%s",c);/* 将c的内容以%s格式写到文本文件中 */
    fclose(fp);/* 关闭文件 */
    return 0;
}
```

文件的顺序读写

读写文件比较灵活,既可以每次读写一个字符,也可以每次读写一个字符串甚至是任意字节数据.标准C语言提供了相应的文件读写函数:**字符读写函数fgetc和fputc,字符串读写函数fgets和fputs,数据块读写函数fread和fprintf**.这些函数的原型声明都在stdio.h中.

字符读写操作

字符读取函数

字符读取函数从指定文件中读取一个字符.读取的前提是该文件必须以读或者读写的方式打开.函数原型声明是:

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

fgetc的形参是当前打开的文件的指针,函数的功能是**从文件指针fp当前所指的位置读取一个字符**,然后文件指针自动指向下一个字节.

函数读取成功时,将读取的字符作为unsigned char类型转换为整型值返回.否则函数返回EOF.

字符写入函数

字符写入函数将字符写入指定文件,文件必须以写或读写形式打开.函数原型声明为:

```
int fputc(int ch,FILE *fp);
```

函数fputs将参数ch转换成unsigned char类型然后写到文件指针fp指向的文件中.写入成功时,返回值是被写字符,否则返回EOF.

```c
#include<stdio.h>
#include<stdlib.h>
void read_display(const char * filename);/* 将文件内容在屏幕上显示 */
void input_save(const char * filename);/* 将键盘输入保存到文件中 */
int main()
{
    input_save("D:\\C_code\\Notes\\_file_2.txt");/* 将从键盘上输入的一行字符保存到文件中 */
    read_display("D:\\C_code\\Notes\\_file_2.txt");/* 将文件内容在屏幕上显示输出 */
    return 0;
}
void read_display(const char * filename)
{
    FILE * fp;
    char ch;
    if ((fp=fopen(filename,"r"))==NULL)
    {
        printf("Can't open the file");
        exit(-1);
    }
    while((ch=fgetc(fp))!=EOF)/* 从文件中读取一个字符ch */
        putchar(ch);/* 在显示器上显示字符ch */
    fclose(fp);
}
void input_save(const char * filename)
{
    char ch;
    FILE * fp;
    if((fp=fopen(filename,"w"))==NULL)
    {
        printf("Can't open the file");
        exit(-1);
    }
    while((ch=getchar())!='\n')/* 从键盘上读取字符 */
        fputc(ch,fp);/* 将字符依次写入文件 */
    fclose(fp);
}
```

需要熟悉 `fp=fopen(filename,"r")`，`ch=fgetc(fp)`，`ch=getchar()`，`fputc(ch,fp)` 的用法.exit函数包含在stdlib.h中,通常用于在子程序中终结程序,使用后程序自动结束,返回操作系统.

字符串读写操作

字符串读取操作函数

字符串读取操作函数从指定文件中读取一个字符串,该文件必须以读或读写方式打开,函数声明为:

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

函数fgets从文件指针fp指向的文件文件中读取n-1个字符,并**将这些函数存放到以s为首地址的存储单元中**,在读取最后一个字符后自动添加字符串结束符'\0'.如果在读取n-1个字符结束前**遇到换行符或EOF**,则读取结束.

fgets读取成功后返回指针s,否则返回NULL.

fgets(s,n,fp)函数直接将读取的字符串存放在s为首的地址中,fgetc(fp)则往往需要将读取的一个字符存放到一个变量中.

```c
#include <stdio.h>

int main() {
    FILE *input_fp, *output_fp;
    char line[100];
    // 打开输入文件
    input_fp = fopen("input.txt", "r");
    if (input_fp == NULL) {
        printf("Error opening input file.\n");
        return 1;
    }
    // 打开输出文件
    output_fp = fopen("output.txt", "w");
    if (output_fp == NULL) {
        printf("Error opening output file.\n");
        fclose(input_fp);
        return 1;
    }
    // 逐行读取输入文件中的内容并写入到输出文件
    while (fgets(line, sizeof(line), input_fp) != NULL) {
        fprintf(output_fp, "%s", line); // 写入读取到的内容
    }
    // 关闭文件
    fclose(input_fp);
    fclose(output_fp);
    return 0;
}
```

字符串写入操作函数

字符串写入操作函数将一个字符串写入指定文件,该文件必须以写或者读写的方式打开.函数原型为:

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

函数fputs和函数fgets配对使用.fputs**将指针s指向的字符串写入文件指针fp指向的文件中**,其中字符串可以是字符串常量,字符数组名,字符指针变量.

字符串结束符'\0'不写入文件中.字符串写入成功返回0,否则返回EOF.

```
int main()
{
    FILE * fp;
    char *str="Hello world,I'm confused.\n",ch[80];
    if((fp=fopen("D:\\C_code\\Notes\\_file_3.txt","w"))==NULL)
    {
        printf("Can't open the file");
        exit(-1);
    }
    fputs(str,fp);
    fclose(fp);
    if((fp=fopen("D:\\C_code\\Notes\\_file_3.txt","r"))==NULL)
    {
        printf("Can't open the file");
        exit(-1);
    }
    while(!feof(fp))/* 遇到文件尾结束 */
    {
        if (fgets(ch,50,fp)!=NULL)
            printf("%s",ch);
    }
    fclose(fp);
    return 0;
}
```

格式化读写

格式化读取函数

格式化读取函数从指定文件中按指定格式格式化读取数据,并赋值给相应变量.读取的前提是该文件必须以读或读写方式打开.该函数原型声明为:

```
int fscanf(FILE *fp,const char *format,...);
```

函数fscanf从文件指针fp所指文件中,按照格式控制符format指定的格式读取数据并赋给相应的参数变量.
函数fscanf读取成功时,返回所读取数据项的个数,否则返回EOF.

```c
#include <stdio.h>
int main() {
    FILE *fp;
    int num1, num2;
    fp = fopen("input.txt", "r");
    if (fp == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    // 从文件中读取两个整数
    fscanf(fp, "%d %d", &num1, &num2);
    // 输出读取到的两个整数
    printf("Read integers: %d, %d\n", num1, num2);
    fclose(fp);
    return 0;
}
```

格式化写入函数

格式化写入函数从指定文件中按指定格式格式化写入数据,并赋值给相应变量.读取的前提是该文件必须以写或读写方式打开.该函数原型声明为:

```c
int fprintf(FILE *fp,const char *format,...);
```

fprintf和fscanf配对使用.函数fprintf将输出参数列表中的数据按指定格式写入文件指针fp所指向的文件中.
函数fprintf写入成功时,返回所写入数据项的个数,否则返回EOF.
函数fprintf和fscanf的读写对象是磁盘文件,而函数printf和scanf的读写对象是默认终端(键盘和显示器).

```c
#include <stdio.h>
int main() {
    FILE *fp;
    int num1 = 10, num2 = 20;
    fp = fopen("output.txt", "w");
    if (fp == NULL) {
        printf("Error opening file.\n");
        return 1;
    }
    // 将两个整数写入文件
    fprintf(fp, "Numbers: %d, %d\n", num1, num2);
    fclose(fp);
    return 0;
}
```

`fprintf(fp,"%s %5.2f %d\n",&goods[i][0],price[i],number[i]);` .在 `fprintf()` 函数中, `%s` 格式指示符用于打印一个字符串,而字符串通常被表示为一个字符数组的地址.在这种情况下, `goods[i][0]` 是一个字符串,因此要传递它的地址.