小甲鱼C语言《带你学C带你飞》学习笔记(2)

日常 3-15 894阅读 27点赞 5评论

视频链接https://www.bilibili.com/video/av27744141

由于篇幅限制，本笔记分两部分：

小甲鱼C语言《带你学C带你飞》学习笔记(1)-传送门

小甲鱼C语言《带你学C带你飞》学习笔记(2)-本文

本笔记是本人根据小甲鱼视频讲课内容记录，一些程序做了小修改。根据自身补充了一部分拓展内容。如有错误，欢迎一起讨论 ——正能量的康sir

如果喜欢我的文章，不妨点个关注、点个赞、投个币，谢谢：）

P45单链表1

单链表

信息域 指针域

信息域存储数据，指针域指向下一个地址

最后一个指针域指向NULL

单链表插入元素(头插法)

```
#include<stdio.h>
#include<stdlib.h>
struct Book {
    char title[128];
    char author[40];
    struct Book *next;
};
void getInput(struct Book *book){
    printf("请输入书名：");
    scanf("%s",book->title);
    printf("请输入作者：");
    scanf("%s",book->author);
}
void addBook(struct Book **library){//**library指向book结构指针的指针
    struct Book *book,*temp;
    book=(struct Book *)malloc(sizeof(struct Book));
    if(book==NULL){
```

正能量的康sir
粉丝: 3737 阅读: 6606
关注

查看目录 来自文集：C语言/C++/数据

推荐文章

链表随想(链表是个神么玩意nie～)
11-1 阅读5

Struct sans设定
11-29 阅读31

1
10-25 阅读0

何炅悉心挑选“四小天鹅”，主角沦为“小透明”，毫不起眼的配角却越来越火
12-7 阅读2.3万

学习C语言对我以后的工作到底有没有帮助
9-24 阅读18

查看更多

更多

专栏投稿 前去写文章
专栏帮助 查看专栏使用

```
            printf("内存分配失败！\n");
            exit(1);//需要stdlib.h
        }
        getInput(book);
        if(*library!=NULL){//有书的情况，头指针 *library指向新插入书位置
            temp=*library;//头指针原来书的位置
            *library=book;//指向新插入书位置
            book->next=temp;//next指向下一个节点地址，即原来书的位置
        }
        else{//开始没有书的情况 *library=NULL的情况下添加书
            *library=book;//第一个节点指针不是 NULL了是book节点的
            book->next=NULL; //next指向下一个节点地址，即NULL
        }
}

void printLibrary(struct Book *library){
        struct Book *book;
        int count = 1;
        book=library;
        while(book!=NULL){
            printf("-----------Book%d-----------\n",count);
            printf("书名：%s\n",book->title);
            printf("作者：%s\n",book->author);
            book=book->next;
            count++;
        }
}
void releseLibrary(struct Book *library){//释放资源
        struct Book *temp;
        while(library!=NULL){
            temp=library->next;
            free(library);
            library=temp;
        }
}
int main(){
        struct Book *library=NULL;
        int ch;
        while(1){
```

```
        do{
            printf("是否录入书籍信息(Y/N):");
            ch=getchar();
        } while(ch!='Y'&&ch!='N');
        if(ch=='Y')
        {
            addBook(&library);
        }
        else
        {
            break ;
        }
    }
    do{
        printf("是否打印书籍信息(Y/N):");
        ch=getchar();
    } while(ch!='Y'&&ch!='N');
    if(ch=='Y')
    {
        printLibrary(library);
    }
    releseLibrary(library);
    return 0;
}
```

P46单链表2

单链表插入元素(尾插法)

只需修改addBook函数的有书情况下的插法

```
    struct Book *temp;
。。。
    if(*library!=NULL){//有书的情况
        temp=*library;
        while(temp->next!=NULL){//定位 单链表尾部位置
            temp=temp->next;
        }
        //插入数据
        temp->next=book;
        book->next=NULL;
    }
```

优化 定义一个指针始终指向尾部，提高效率

```
static struct Book *tail;
。。。
    if(*library!=NULL){//有书的情况
        tail->next=book;
        book->next=NULL;
    }
    else{//开始没有书的情况 *library=NULL的情况下添加书
        *library=book;//第一个节点指针不是 NULL了是book节点的
        book->next=NULL; //next指向下一个节点地址，即NULL
    }
    tail=book;
```

搜索单链表

```
struct Book *searchBook(struct Book *library,char *target){
    struct Book *book;
    book=library;
    while(book!=NULL){
        if(!strcmp(book->title,target)||!strcmp(book->author,target)){//strcmp相等返回0 。需要string.h
            break;
        }
        book=book->next;
    }
    return book;
}
void printBook(struct Book *book){
    printf("书名：%s",book->title);
    printf("作者：%s",book->author);
}
int main(){
。。。
    char input[128];
    struct Book *book;
。。。
    printf("请输入查找的书名或作者") ;
    scanf("%s",input);
    book=searchBook(library,input);
    if(book==NULL)
```

```
    {
        printf("很抱歉，未找到");
    }
    else{
        do{
            printf("已找到符合条件的图书...");
            printBook(book);
        }while((book =searchBook(book->next,input))!=NULL);//多本图书都匹配的话可以重复找
    }
。。。
}
```

P47单链表3

单链表插入节点(中间插入)

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
    int value;
    struct Node *next;
};
void insertNode(struct Node **head,int value){
    struct Node *previous;//上一个
    struct Node *current;//当前
    struct Node *it;//new是关键字 就用it吧
    current= *head;
    previous=NULL;
    while(current!=NULL&&current->value<value){
        previous=current;
        current=current->next;
    }
    it=(struct Node *)malloc(sizeof(struct Node));
    if(it==NULL){
        printf("内存分配失败！\n");
        exit(1);
    }
    it->value=value;
    it->next=current;
```

```
    if(previous==NULL){//空链表，current为NULL即*head为NULL的情况下，不执行循环导致 previous为NULL
        *head=it;
    }
    else//不是空链表
    {
        previous->next= it;
    }
}
void printNode(struct Node *head){
    struct Node *current;
    current=head;
    while(current!=NULL){
        printf("%d ",current->value);
        current=current->next;
    }
    printf("\n");
}
int main(){
    struct Node *head =NULL;
    int input;
    while(1){
        printf("请输入一个整数(输入-1表示结束)：");
        scanf("%d",&input);
        if(input==-1){
            break;
        }
        insertNode(&head,input);
        printNode(head);
    }
}
```

单链表删除节点

```
void deleteNode(struct Node **head,int value){
    struct Node *previous;
    struct Node *current;
    current = *head;
    previous=NULL;
    while(current!=NULL&&current->value!=value){
```

```
            previous=current;
            current=current->next;
        }
        if(current==NULL){
            printf("找不到匹配的节点");
            return;
        }
        else{
            if(previous==NULL){
                *head=current->next;
            }
            else{
                previous->next=current->next;
            }
            free(current);
        }
}
```

main函数修改

```
        printf("开始测试删除整数。。。\n");
        while(1){
            printf("请输入一个整数(输入-1表示结束)：");
            scanf("%d",&input);
            if(input==-1){
                break;
            }
            deleteNode(&head,input);
            printNode(head);
        }
```

P48内存池

内存碎片

时间上消耗 应用层——内核层——应用层

解决方法 内存池 让程序额外维护的一个缓存区域

当申请内存时检查内存池有没有适合的垃圾内存块，重新使用

想申请内存时

如果内存池非空，则直接从里面获取空间

如果内存池为空，则重新申请内存空间

想释放内存时

如果内存池有空位（小于内存池容量），那么将即将废弃的内存块插入内存池

如果内存池没有空位，那么就用free等释放掉

P49基础typedef

typedef基本功能 给数据类型(包括结构体)起别名

typedef A B;//给A取别名B

typedef A B,C;//给A取别名B/C。可以取多个别名

define也可以 #define B A

区别

define是直接替换

typedef是对类型的封装。真正的起别名。可以对指针类型取别名。

课外知识 fortran世界上第一门高级编程语言

给结构体取别名

struct Date{

int year;

int month;

int day

};

typedef struct Date DATE；//如果需要，可以同时定义指针typedef struct Date DATE，*PDATE;

或者

typedef struct Date{

int year;

int month;

int day

} DATE；//如果需要，可以同时定义指针

P50进阶typedef

使用typedef目的一般有两个

给变量起容易记住且意义明确的别名；

简化一些比较复杂的类型声明。

一些比较复杂的声明语句

int (*ptr)[3]数组指针 ptr是一个指针，指向三个整型元素的数组(数组起始地址)

哪个是数组指针，哪个是指针数组呢：

A) int *p1[10];

B) int (*p2)[10];

这里需要明白一个符号之间的优先级问题。

“[]”的优先级比“*”要高。p1 先与“[]”结合，构成一个数组的定义，数组名为p1，int * 修饰的是数组的内容，即数组的每个元素。那现在我们清楚，这是一个数组，其包含10 个指向int 类型数据的指针，即指针数组。

至于p2 就更好理解了，在这里“（）”的优先级比“[]”高，“*”号和p2 构成一个指针的定义，指针变量名为p2，int 修饰的是数组的内容，即数组的每个元素。数组在这里并没有名字，是个匿名数组。那现在我们清楚p2 是一个指针，它指向一个包含10 个int 类型数据的数组，即数组指针

```
#include<stdio.h>

typedef int (*PTR_TO_ARRAY)[3];

int main(){
    int array[3]={1,2,3};
    PTR_TO_ARRAY ptr_to_array=&array;
    int i;
    for(i=0;i<3;i++){
        printf("%d\n",(*ptr_to_array)[i]);
    }
    return 0;
}
```

int(*fun)(void);函数指针 指向一个参数为void，返回值为int的函数

```
#include<stdio.h>

typedef int (*PTR_TO_FUN)(void);

int fun(){
    return 520;
}

int main(){
    PTR_TO_FUN ptr_to_fun=&fun;//直接写fun也可，函数名即是地址
    printf("%d\n",(*ptr_to_fun)());
    return 0;
}
```

int *(*array[3])(int);//*array[3]是指针数组，int *(*array[3])(int);是指针函数，返回值int参数int

```
#include<stdio.h>

typedef int *(*PTR_TO_FUN)(int);

int *funA(int num){
```

```
                printf("%d\t",num);
                return &num;//返回无意义，只是测试
}
int *funB(int num){
                printf("%d\t",num);
                return &num;//返回无意义，只是测试
}
int *funC(int num){
                printf("%d\t",num);
                return &num;//返回无意义，只是测试
}
int main(){
                PTR_TO_FUN array[3]={&funA,&funB,&funC};
                int i;
                for(i=0;i<3;i++){
                    printf("addr of num:%p\n",(*array[i])(i));
                }
                return 0;
}
```

void (*funA(int,void (*funB)(int)))(int);函数指针

P51共用体

union共用体名称

{

共用体成员1;

共用体成员2;

共用体成员3;

。。。

};

共用体所有成员共享同一个内存地址。地址长度足以容纳最大成员的长度。也会受内存对齐影响

```
#include<stdio.h>
#include<string.h>
typedef int *(*PTR_TO_FUN)(int);
union Test{
    int i;
    double pi;
    char str[10];
```

```
};
int main(){
    union Test test;
    test.i=520;
    test.pi=3.14;
    strcpy(test.str,"FishC.com");
    printf("addr of test.i:%p\n",&test.i);
    printf("addr of test.pi:%p\n",&test.pi);
    printf("addr of test.str:%p\n",&test.str); //输出结果几个地址%p相同
    printf("test.i:%d\n",test.i);
    printf("test.pi:%.2f\n",test.pi);
    printf("test.str:%s\n",test.str); //输出结果只有str正确， 因为前两个被覆盖了
    printf("size of int:%d\n",sizeof(int));
    printf("size of double:%d\n",sizeof(double));
    printf("size of test.str:%d\n",sizeof(test.str));
    printf("size of test:%d\n",sizeof(test));
    return 0;
}
```

定义共用体类型变量，和结构体两种方式基本相似。

另外，共用体的名字不是必须的

```
union {
    int a;
    char b;
} a,b,c;
```

初始化共用体

union data a={520}；//初始化第一个成员

union data b=a;//使用一个共用体初始化另一个

union data c={.ch='c'};//C99新特性，指定初始化成员

拓展

共用体常用来节省内存，特别是一些嵌入式编程，内存是非常宝贵的!

共用体也常用于操作系统数据结构或硬件数据结构!

union 在操作系统底层的代码中用的比较多，因为它在内存共享布局上方便且直观。所以网络编程，协议分析，内核代码上有一些用到 union 都比较好懂，简化了设计。

共用体（union）是一种数据格式，它能够存储不同类型的数据，但是只能同时存储其中的一种类型。

————————————————————

P52枚举类型

——列举

如果一个变量只有几种可能的值，那么就可以将其定义为枚举(enumeration)类型

声明枚举类型

enum枚举类型名称 {枚举值名称，枚举值名称。。。};

定义枚举变量

enum枚举类型名称 枚举变量1，枚举变量2;

```
#include<stdio.h>
#include<time.h>
int main(){
    enum Week {sun,mon,tue,wed,thu,fri,sat};//默认第一个是0，此时sun=0,mon=1。。。。也可以指定值，后面的会依次自动加1。enum Week {sun=1,mon,tue,wed,thu,fri,sat};
    enum Week today;
    struct tm *p;//tm结构体 包含了当地时间和日期，其中成员变量int tm_wday 表示星期几范围0-6
    time_t t;
    time(&t);//time函数返回表示当前时间的time_t
    p=localtime(&t);//localtime函数将time_t类型的值转化为具体的本地时间和日期
    today = (enum Week)p->tm_wday;
    switch(today){
        case mon:
        case tue:
        case wed:
        case thu:
        case fri:
            printf("工作日\n");break;
        case sat:
        case sun:
            printf("休息日\n");break;
        default:printf("Error\n");
    }
    return 0;
}
```

拓展

https://blog.csdn.net/L202132061/article/details/79383855

https://blog.csdn.net/L202132061/article/details/79383855

P53位域

单片机 集成电路芯片，把CPU、RAM、ROM、I/O等集成到一块硅片上构成小而完善的微型计算机系统。

节约空间

位域，或称位段、位字段。

例如，对一个字节划分为几个部分并命名，这几部分就是位域

使用位域的做法是 在结构体定义时，在结构体或成员后面使用冒号和数字来表示该成员所占的位byte数。

```
#include<stdio.h>
int main(){
    struct Test{
        unsigned int a:1;
        unsigned int b:1;
        unsigned int c:2;
    };
    struct Test test;
    test.a=0;
    test.b=1;
    test.c=2;
    printf("a=%d,b=%d,c=%d\n",test.a,test.b,test.c);
    printf("size of test=%d",sizeof(test));
    return 0;
}
//a=0,b=1,c=2
//size of test=4
//如果改为 unsigned int c:1;则c=0因为2二进制为10，不够两位，只能存后面的0
```

无名位域

位域成员可以没有名称，只要给出数据类型和宽度即可。

```
struct Test{
    unsigned int x:100;
    unsigned int :100//位域成员可以没有名称，只要给出数据类型和宽度即可。
}
```

P54位操作

c语言并没有规定一个字节有几位(一般是8位)，只是规定“可寻址的数据存储单位，其尺寸必须可以容纳运行环境的基本字符集的任何成员”。一般是由编译器规定在limits.h中

逻辑位运算符

只作用于整型数据

~按位取反

1变0,0变1

&按位与

不是逻辑与(&&)。同时为1才是1

^按位异或

两个不同为1，相同为0

|按位或

不是逻辑或(||)。有1则1,全0才0

和赋值号=结合

除了按位取反都可以和赋值号结合

```
#include<stdio.h>
int main(){
    int mask = 0xFF;//0x表示是16进制  FF是15 15即0000 0000 0000 0000 1111 1111
    int v1= 0xABCDEF; //10 11 12 13 14 15 即   1010 1011 1100 1101 1110 1111
    int v2= 0xABCDEF;
    int v3= 0xABCDEF;
    v1 &=mask;//即v1=v1&mask;下面两句也是
    v2 |=mask;
    v3 ^=mask;
    printf("v1=0x%X\nv2=0x%X\nv3=0x%X\n",v1,v2,v3);
    return 0;
}
//v1=0xEF              0000 0000 0000 0000 1110 1111
//v2=0xABCDFF        1010 1011 1100 1101 1110 1111
//v3=0xABCD10        1010 1011 1100 1101 0001 0000
```

P55移位和位操作的应用

移位运算符:

左移位运算符<<

右移位运算符>>

左移位运算符<<

左边操作数是即将被移位的数据，右边是要移动的位数。移出的数据扔掉，移动后空位用0填充

右移位运算符<<

左边操作数是即将被移位的数据，右边是要移动的位数。移出的数据扔掉，移动后空位用0填充

和赋值号结合

```
#include<stdio.h>
int main(){
    int value=1;
    printf("-----左移---------\n");
    while(value<1024){
        value<<=1;//value=value<<1;
        printf("value=%d\n",value);
    }
    printf("-----右移---------\n");
    value=1024;
    while(value>1){
        value>>=1;
        printf("value=%d\n",value);
    }
    return 0;
}
```

一些未定义行为

移位运算符右操作数如果是负数，或右操作数大于左操作数支持的最大宽度，那么表达式结果均属于“未定义行为”。不同编译器结果不同。

有符号和无符号也对移位运算符有不同的影响。有符号数移动后是否覆盖符号位决定权还是在编译器。

位运算符的应用

掩码

配电箱

只开主卧 掩码10001000 相"&"结果10001000

打开位 开主卧 掩码10001000 相"|"结果11001000

关闭位 关闭客厅 掩码01000000 "~"掩码10111111 相"&"结果10001000

转置位 掩码01001000 "^"结果10001000

P56打开和关闭文件

万物皆文件Everything is a file

KISS原则Keep it simple,stupid.

文本文件和二进制文件

文本文件　由一些字符的序列组成的文件

二进制文件 通常指除了文本文件以外的

严格来说，文本文件也属于二进制文件。打开方式要区分开主要是因为换行符。C语言换行符\n，unix系统\n，windows用\r\n，mac用\r。如果在windows系统以文本文件打开，读会将\r\n自动转换为\n，写会\n转换为\r\n,但是以二进制模式打开则不会做转换。如果在unix系统二者是一样的。

打开和关闭文件

读 从文件中获取数据

写 将数据写入到文件里

在完成读写操作后，必须将其关闭

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    FILE *fp;
    int ch;

    if ((fp = fopen("hello.txt", "r")) == NULL)
    {
        printf("打开文件失败！\n");
        exit(EXIT_FAILURE);
    }
    while ((ch = getc(fp)) != EOF)//EOF通常是-1
    {
        putchar(ch);
    }
    fclose(fp);//关闭，释放缓冲区
    return 0;
}
```

fopen用于打开文件并返回函数指针

打开成功，返回指向FILE结构的文件指针；打开失败，返回NULL并设置errno为指定的错误

fopen(路径，模式)

路径参数可以是相对路径(./a.txt)，也可以是绝对路径(/home/user1/a.txt)。如果只给出文件名(a.txt)则表示该文件在当前文件夹。

模式参数

模式	描述
"r"	1. 以只读的模式打开一个文本文件，从文件头开始读取 2. 该文本文件必须存在
"w"	1. 以只写的模式打开一个文本文件，从文件头开始写入 2. 如果文件不存在则创建一个新的文件 3. 如果文件已存在则将文件的长度截断为 0（重新写入的内容将覆盖原有的所有内容）
"a"	1. 以追加的模式打开一个文本文件，从文件末尾追加内容 2. 如果文件不存在则创建一个新的文件
"r+"	1. 以读和写的模式打开一个文本文件，从文件头开始读取和写入 2. 该文件必须存在 3. 该模式不会将文件的长度截断为 0（只覆盖重新写入的内容，原有的内容保留）
"w+"	1. 以读和写的模式打开一个文本文件，从文件头开始读取和写入 2. 如果文件不存在则创建一个新的文件 3. 如果文件已存在则将文件的长度截断为 0（重新写入的内容将覆盖原有的所有内容）
"a+"	1. 以读和追加的模式打开一个文本文件 2. 如果文件不存在则创建一个新的文件 3. 读取是从文件头开始，而写入则是在文件末尾追加
"b"	1. 与上面 6 中模式均可结合（"rb"， "wb", "ab", "r+b", "w+b"， "a+b"） 2. 其描述的含义一样，只不过操作的对象是二进制文件

P57读写文件1

顺序读写和随机读写

读单个字符

fgetc、getc

```
#include <stdio.h>

 int fgetc( FILE *stream );
```

fgetc()函数返回来自stream(流)中的下一个字符

stream参数是FILE对象的指针，指定一个待读取的文件流

返回值：

将读取到的unsigned char类型转换为int并返回

如果到达文件尾或者发生错误时返回EOF.

fgetc 函数和 getc 函数两个的功能和描述基本上是一模一样的，它们的区别主要在于实现上：fgetc 是一个函数；而 getc 则是一个宏的实现。

一般来说宏产生较大的代码，但是避免了函数调用的堆栈操作，所以速度会比较快。

由于 getc 是由宏实现的，对其参数可能有不止一次的调用，所以不能使用带有副作用（side effects）的参数。所谓带有副作用的参数就是指 getc(fp++) 这类型的参数，因为参数在宏的实现中可能会被调用多次，所以你的想法是 fp++，而副作用下产生的结果可能是 fp++++++。

写单个字符fputc，putc

#include <stdio.h>

int fputc(int c, FILE *stream);

c　　指定待写入的字符

stream　　该参数是一个 FILE 对象的指针，指定一个待写入的文件流

返回值：

如果函数没有错误，返回值是写入的字符；

如果函数发生错误，返回值是EOF

fputc 是一个函数；而 putc 则是一个宏的实现

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    FILE *fp1,*fp2;
    int ch;
    if ((fp1 = fopen("hello.txt", "r")) == NULL)
    {
        printf("打开文件失败！\n");
        exit(EXIT_FAILURE);//exit() 需要stdlib.h
    }
    if ((fp2 = fopen("fish.txt", "w")) == NULL)
    {
        printf("打开文件失败！\n");
        exit(EXIT_FAILURE);//exit() 需要stdlib.h
    }
    while ((ch = getc(fp1)) != EOF)//EOF通常是-1
    {
        fputc(ch,fp2);
```

```
    }
    fclose(fp1);//关闭，释放缓冲区
    fclose(fp2);//关闭，释放缓冲区
    return 0;
}
```

读写整个字符串fgets、fputs

fgets 函数用于从指定文件中读取字符串。

fgets 函数最多可以读取 size - 1 个字符，因为结尾处会自动添加一个字符串结束符 '\0'。当读取到换行符（'\n'）或文件结束符（EOF）时，表示结束读取（'\n' 会被作为一个合法的字符读取，EOF不会）。

函数原型：

#include <stdio.h>

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

s　　字符型指针，指向用于存放读取字符串的位置

size　指定读取的字符数（包括最后自动添加的 '\0'）

stream　　该参数是一个 FILE 对象的指针，指定一个待操作的数据流

返回值：

1. 如果函数调用成功，返回 s 参数指向的地址。

2. 如果在读取字符的过程中遇到 EOF，则 eof 指示器被设置；如果还没读入任何字符就遇到这种 EOF，则 s 参数指向的位置保持原来的内容(s不变)，函数返回 NULL。

3. 如果在读取的过程中发生错误，则 error 指示器被设置，函数返回 NULL，但 s 参数指向的内容可能被改变。

fputs 函数用于将一个字符串写入到指定的文件中，表示字符串结尾的 '\0' 不会被一并写入。

#include <stdio.h>

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

s　　字符型指针，指向用于存放待写入字符串的位置

stream　　该参数是一个 FILE 对象的指针，指定一个待操作的数据流

返回值：

如果函数调用成功，返回一个非 0 值；(此处错误。API文档里是成功时返回非负值, 失败时返回EOF)

如果函数调用失败，返回EOF

feof 函数用于检测文件的末尾指示器（end-of-file indicator）是否被设置。

#include <stdio.h>

int feof(FILE *stream);

stream　　该参数是一个 FILE 对象的指针，指定一个待检测的文件流

返回值：

如果检测到末尾指示器（end-of-file indicator）被设置，返回一个非 0 值；

如果检测不到末尾指示器（end-of-file indicator）被设置，返回值为 0。

feof 函数仅检测末尾指示器的值，它们并不会修改文件的位置指示器。

文件末尾指示器只能使用 clearerr 函数清除。

```
#include <stdio.h>

#include <stdlib.h>

#define MAX 1024

int main()

{

    FILE *fp;

    char buffer[MAX];

    if ((fp = fopen("hello.txt", "w")) == NULL)

    {

        printf("打开文件失败！\n");

        exit(EXIT_FAILURE);//exit() 需要stdlib.h

    }

    fputs("hello 1111\n",fp);

    fputs("hello 2222\n",fp);

    fputs("hello 3333\n",fp);

    fclose(fp);//关闭,写入文件，释放缓冲区 。

        //需要fclose，不关的话，文件指示器指向文件末尾，影响后面操作 。而且还在缓冲区，还没写入文件。

    if ((fp = fopen("hello.txt", "r")) == NULL)

    {

        printf("打开文件失败！\n");

        exit(EXIT_FAILURE);//exit() 需要stdlib.h

    }

    while(!feof(fp)) //feof检测不到末尾，返回0 。所以这里是未到末尾 。

    {

      fgets(buffer,MAX,fp);//每次最多读取MAX-1个字符，因为结尾自动添加\0。 读取到\n或EOF会结束这一行，\n也会被作为合法字符读取，EOF不会 。

            printf("%s",buffer);

        }

      return 0;

}

//hello 1111

//hello 2222

//hello 3333

//hello 3333

//出现问题 第四行打印第三行内容，但文件hello中并没有第四行 ， 只有回车。
```

//原因 fgets如果还没读入任何字符就遇到 EOF，则 s 参数指向的位置保持原来的内容(s不变)，函数返回 NULL。

//解决 if(!fgets(buffer,MAX,fp)==NULL)printf("%s",buffer);

P58读写文件2

格式化读写文件

fscanf、fprintf

和scanf、printf相似，只不过是从文件读取、输出到文件

拓展 为什么scanf中用&取地址符，而printf不用。因为scanf本来就是一个函数，用取地址后就能将接受的数据存在这个地址里，在scanf函数外也能用。指针在函数内就是通过访问所指向地址的值来进行改写，并且能延续到函数外。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main()
{
    FILE *fp;
    struct tm *p;
    time_t t;
    time(&t);
    p=localtime(&t);
    if ((fp = fopen("date.txt", "w")) == NULL)
    {
        printf("打开文件失败！\n");
        exit(EXIT_FAILURE);//exit() 需要stdlib.h
    }
    fprintf(fp,"%d-%d-%d",1900+p->tm_year,1+p->tm_mon,p->tm_mday);
    fclose(fp);
    int year,month,day;
    if ((fp = fopen("date.txt", "r")) == NULL)
    {
        printf("打开文件失败！\n");
        exit(EXIT_FAILURE);//exit() 需要stdlib.h
    }
    fscanf(fp,"%d-%d-%d",&year,&month,&day);
    printf("%d-%d-%d",year,month,day);
    fclose(fp);
    return 0;
```

```
}
```

以二进制方式读写文本文件

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fp;
    if ((fp = fopen("text.txt", "wb")) == NULL)
    {
        printf("打开文件失败！\n");
        exit(EXIT_FAILURE);//exit() 需要stdlib.h
    }
        fputc('5',fp);
        fputc('2',fp);
        fputc('0',fp);
        fputc('\n',fp);
    fclose(fp);
      return 0;
}
```

二进制读写文件fread、fwrite

fread 函数用于从指定的文件中读取指定尺寸的数据。

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
```

ptr　指向存放数据的内存块指针，该内存块的尺寸最小应该是 size * nmemb 个字节

size　指定要读取的每个元素的尺寸，最终尺寸等于 size * nmemb

nmemb　指定要读取的元素个数，最终尺寸等于 size * nmemb

stream　该参数是一个 FILE 对象的指针，指定一个待读取的文件流

返回值：

1. 返回值是实际读取到的元素个数（nmemb）；

2. 如果返回值比 nmemb 参数的值小，表示可能读取到文件末尾或者有错误发生（可以使用 feof 函数或 ferror 函数进一步判断）。

fwrite 函数用于将指定尺寸的数据写入到指定的文件中。

函数原型：

```
#include <stdio.h>
size_t fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
```

ptr　指向存放数据的内存块指针，该内存块的尺寸最小应该是 size * nmemb 个字节

size　指定要写入的每个元素的尺寸，最终尺寸等于 size * nmemb

nmemb　指定要写入的元素个数，最终尺寸等于 size * nmemb

stream　该参数是一个 FILE 对象的指针，指定一个待写入的文件流

返回值：

1. 返回值是实际写入到文件中的元素个数（nmemb）；

2. 如果返回值与 nmemb 参数的值不同，则有错误发生。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Date{
        int year;
        int month;
        int day;
};
struct Book{
        char name[40];
        char author[40];
        char publisher[40];
        struct Date date;
};
int main()
{
    FILE *fp;
    struct Book *book_for_write,*book_for_read;
    book_for_write=(struct Book *)malloc(sizeof(struct Book));
    book_for_read=(struct Book *)malloc(sizeof(struct Book));
    if(book_for_write==NULL||book_for_read==NULL)
    {
       printf("内存分配失败");
              exit(EXIT_FAILURE);
        }
        strcpy(book_for_write->name,"带你学c带你飞");
        strcpy(book_for_write->author,"小甲鱼");
        strcpy(book_for_write->publisher,"清华大学出版社");
        book_for_write->date.year=2017;
        book_for_write->date.month=11;
        book_for_write->date.day=11;
        if ((fp = fopen("file.txt", "w")) == NULL)
    {
```

```
        printf("打开文件失败！\n");
        exit(EXIT_FAILURE);//exit() 需要stdlib.h
    }
    fwrite(book_for_write,sizeof(struct Book),1,fp);
    fclose(fp);
    if ((fp = fopen("file.txt", "r")) == NULL)
    {
        printf("打开文件失败！\n");
        exit(EXIT_FAILURE);//exit() 需要stdlib.h
    }
    fread(book_for_read,sizeof(struct Book),1,fp);
    printf("书名：%s\n",book_for_read->name);
    printf("作者：%s\n",book_for_read->author);
    printf("出版社：%s\n",book_for_read->publisher);
    printf("出版日期：%d-%d-%d\n",book_for_read->date.year,book_for_read->date.month,book_for_read->date.day);
    fclose(fp);
    return 0;
}
```

P59随机读写文件

ftell返回给定流 stream 的当前文件位置。

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   FILE *fp;
   if((fp=fopen("hello.txt","w"))==NULL){
    printf("文件打开失败！\n");
    exit(EXIT_FAILURE);
       }
       printf("%ld\n",ftell(fp));
       fputc('F',fp);
       printf("%ld\n",ftell(fp));
       fputs("ishC\n",fp);
       printf("%ld\n",ftell(fp));
       fclose(fp);
       return 0;
}
//0开头
```

```
//1
//7因为win系统换行 \r\n 所以1+6
```

rewind移动到文件头

上面程序添加

```
rewind(fp);
fputs("Hello",fp);
```

会把原来的内容覆盖掉

fseek用于设置文件流的位置指示器

```
#include<stdio.h>
int fseek(FILE *stream,long int offset,int whence);
```

stream -- 这是指向 FILE 对象的指针，该 FILE 对象标识了流。

offset -- 这是相对 whence 的偏移量，以字节为单位。

whence -- 这是表示开始添加偏移 offset 的位置。它一般指定为下列常量之一：

SEEK_SET 文件的开头

SEEK_CUR 文件指针的当前位置

SEEK_END 文件的末尾

返回值

如果成功，则该函数返回零，否则返回非零值。

```
#include <stdio.h>
#include <stdlib.h>
#define N 4
struct Stu{
        char name[24];
        int num;
        float score;
}stu[N],sb;
int main()
{
    FILE *fp;
    int i;
    if((fp=fopen("score.txt","w"))==NULL){
     printf("打开文件失败！\n");
     exit(EXIT_FAILURE);
        }
        for(i=0;i<N;i++){
                printf("请开始录入成绩(格式:姓名 学号 成绩)");
                scanf("%s %d %f",stu[i].name,&stu[i].num,&stu[i].score);
```

```
    }
    fwrite(stu,sizeof(struct Stu),N,fp);
    fclose(fp);
    if((fp=fopen("score.txt","rb"))==NULL){
        printf("打开文件失败！");
        exit(EXIT_FAILURE);
    }
    fseek(fp,sizeof(struct Stu),SEEK_SET);
    fread(&sb,sizeof(struct Stu),1,fp);
    printf("%s(%d)的成绩是：%.2f\n",sb.name,sb.num,sb.score);
    fclose(fp);
    return 0;
}
```

可移植性问题

对于以二进制模式打开的文件，fseek在某些操作系统中可能不支持SEEK_END位置。

对于以文本模式打开的文件，feek函数的whence参数只能取SEEK_SET才是有意义的，并且传递给offset参数的值要么是0，要么是上一次对同一个文件调用ftell函数获取的返回值。

P60标准流和错误处理

标准流

标准输入stdin

标准输出stdout

标准错误输出stderr

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fp;
    if((fp=fopen("压根都不存在的文件.txt","r"))==NULL){
        fputs("打开文件失败！\n",stderr);
        exit(EXIT_FAILURE);
    }
    fclose(fp);
    return 0;
}
```

错误处理

错误指示器ferror

使用clearerr函数可以人为的清除文件末尾指示器和错误指示器状态

ferror函数只能检测是否出错，但无法获取错误原因。不过，大多数系统函数在出现错误时会将错误原因就在errno中。

perror函数可以直观的打印出错误原因。会自己加"冒号空格错误原因"。例如perror("错误原因是");输出为"错误原因是: Bad file descriptor"

strerror函数直接返回错误码对应的错误消息。

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include<string.h>
int main()
{
	FILE *fp;
	int ch;
	if((fp=fopen("hello.txt","r"))==NULL){
		fputs("打开文件失败！\n",stderr);
		exit(EXIT_FAILURE);
	}
	while(1){
		ch=fgetc(fp);
		if(feof(fp)){
			break;
		}
		putchar(ch);
	}
	fputc('c',fp);//因为只读的，所以会失败，触发ferro
	if(ferror(fp)){
		fputs("出错了，这条消息输出到错误输出流\n",stderr);
		printf("使用errno得到的错误代码:%d\n",errno); //errno.h
		perror("使用perror得到的错误原因是"); //stdio.h
		//fputs("出错了,，原因是%s\n",stderr);
		printf("使用strerror得到的错误原因是:%s\n",strerror(errno));//strerror需要<string.h>
	}
	clearerr(fp);
	if(feof(fp)||ferror(fp)){
		printf("123456");//不打印。末尾指示器和错误指示器被clear清除掉，不会触发
	}
	fclose(fp);
```

```
    return 0;
}
```

输出结果：

Hello

出错了，这条消息输出到错误输出流

使用errno得到的错误代码:9

使用perror得到的错误原因是: Bad file descriptor

使用strerror得到的错误原因是:Bad file descriptor

P61 IO缓冲区

IO缓冲区

```
#include<stdio.h>
#include<stdlib.h>
int main(){
    FILE *fp;
    if((fp=fopen("output.txt","w"))==NULL){
        perror("打开文件失败，原因");
        exit(EXIT_FAILURE);
    }
    fputs("I Love FishC.com\n",fp);//写在了缓冲区里，并没有写入文件
    getchar();//不输入字符，这时看output是空的 。输入后 ，下一步执行fclose才写入文件
    fclose(fp);
    return 0;
}
```

标准IO提供三种类型的缓冲模式

按块缓存 也称为全缓存，在填满缓冲区后才进行实际的设备读写操作；

按行缓存 是指在接收到换行符\n之前，数据都是先缓存在缓冲区的；

不缓存 允许直接读写设备上的数据

setvbuf()定义流 stream 应如何缓冲。

```
#include<stdio.h>
int setvbuf(FILE *stream, char *buffer, int mode, size_t size)
```

参数

stream -- 这是指向 FILE 对象的指针，该 FILE 对象标识了一个打开的流。

buffer -- 这是分配给用户的缓冲。如果设置为 NULL，该函数会自动分配一个指定大小的缓冲。

mode -- 这指定了文件缓冲的模式：

模式 描述

_IOFBF 全缓冲：对于输出，数据在缓冲填满时被一次性写入。对于输入，缓冲会在请求输入且缓冲为空时被填充。

_IOLBF 行缓冲：对于输出，数据在遇到换行符或者在缓冲填满时被写入，具体视情况而定。对于输入，缓冲会在请求输入且缓冲为空时被填充，直到遇到下一个换行符。

_IONBF 无缓冲：不使用缓冲。每个 I/O 操作都被即时写入。buffer 和 size 参数被忽略。

size --这是缓冲的大小，以字节为单位。

返回值

如果成功，则该函数返回 0，否则返回非零值。

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main(){
    char buff[1024];
    memset(buff,'\0',sizeof(buff));//memset使用一个常量字节填充内存空间。需要string.h
    setvbuf(stdout,buff,_IOFBF,1024);//修改为_IONBF 后会都输出
    fprintf(stdout,"welcome to fishc.com");
    fflush(stdout);// 刷新缓冲区会立即输出
    fprintf(stdout,"输入任意字符后才会显示该行字符\n");
    getchar();
    return 0;
}
```


C语言 小甲鱼 带你学C带你飞

来自文集：C语言/C++/数据结构/算法

< 上一篇 已经是开头了~ | 查看目录 1/2 | 下一篇 > 小甲鱼C语言《带你学C带你飞》学习笔...

5 评论

loading...

何炅悉心挑选“四小天鹅”，主角沦为“小透明”，毫不起眼的配角却越来越火
12-7 阅读2.3万

学习C语言对我以后的工作到底有没有帮助
9-24 阅读18

查看更多

更多

专栏投稿
前去写文章

专栏帮助
查看专栏使用