第二次作业总结&&题解

注意：本文档中所有代码均已提交过，如果后面开放补交时提交该代码，会被查重！！！

第一题 五子棋危险判断

值得注意的点：

- 数组大小开略大一点，以防越界
- 注意起始位置的判断条件是行优先，列作为第二判断条件，所以在判断斜线的情况时，可以以当前旗子为起点，向右下和左下方进行延伸判断
- 注意判断时数组下标不能越界，如a [i+1] [j] 还需要判断 i+1< n

```
#include <stdio.h>
#include <ctype.h>
#define MAX 25
int chess[MAX][MAX];
int max = 19;
void read() {
    int i, j;
    char c;
    for(i = 1; i <= max; i++) {
        for(j = 1; j <= max; j++) {
            while(!isdigit(c = getchar()));
            chess[i][j] = c - '0';
        }
    }
}
int row(int x, int y) {
    int res = chess[x][y];
    for(int i = 1; i <= 3; i++) {
        if(res != chess[x][y + i])
            return 0;
    }
    if((!chess[x][y - 1] && y > 1) || (!chess[x][y + 4] && y + 4 <= max))
        return res;
    else return 0;
}
int col(int x, int y) {
    int res = chess[x][y];
    for(int i = 1; i <= 3; i++) {
        if(res != chess[x + i][y])
            return 0;
    }
    if((!chess[x - 1][y] && x > 1) || (!chess[x + 4][y] && x + 4 <= max))
        return res;
    else
        return 0;
}
int fir(int x, int y) {
```

```
    int res = chess[x][y];
    for(int i = 1; i <= 3; i++) {
        if(res != chess[x + i][y + i])
            return 0;
    }
    if((!chess[x - 1][y - 1] && x > 1 && y > 1) || (!chess[x + 4][y + 4] && y + 4 <=
max && x + 4 <= max))
        return res;
    else return 0;
}
int sec(int x, int y) {
    int res = chess[x][y];
    for(int i = 1; i <= 3; i++) {
        if(res != chess[x + i][y - i])
            return 0;
    }
    if((!chess[x - 1][y + 1] && x > 1 && y + 1 <= max) ||  (!chess[x + 4][y - 4] &&
x + 4 <= max && y > 4))
        return res;
    else
        return 0;
}
void judge() {
    for(int i = 1; i <= max; i++) {
        for(int j = 1; j <= max; j++) {
            if(chess[i][j]) {
                int res = row(i, j) | col(i, j) | fir(i, j) | sec(i, j);
                if(res) {
                    printf("%d:%d,%d", chess[i][j], i, j);
                    return ;
                }
            }
        }
    }
    printf("No");
}
int main() {
    int key = 0, ans;
    read();
    judge();
    return 0;
}
```

第二题 字符串替换（新）

关于字符串输入输出

结尾符

字符串一定要以结尾符结尾，遵循这一点原则非常重要

对字符串的操作，一些情况下可以不关心字符串的长度，而只要判断是否遇到了结尾符。这也就是

为什么字符串库函数传参时不需要传入字符串长度

在自己构造、修改字符串完成时，一定要保证结尾符紧跟着内容

不需要关心结尾符后面的内容，除非特殊情况也不要去使用

常用库函数

- `unsigned int strlen(char* str)`：

strlen 所作的是一个计数器的工作，它从内存的某个位置（可以是字符串开头，中间某个位置，甚至是某个不确定的内存区域）开始扫描，直到碰到第一个字符串结束符 \0 为止，然后返回计数器值（长度不包含 \0 ）

不要在循环的截止条件里调用该函数，这样会极大降低程序的性能，像这样：

```
for(int i = 0; i < strlen(str); i++) {
/*do something*/
}
```

因为该函数本身就是遍历字符串判断长度，如果自己在循环时每次都调用这个函数，则复杂度将增加

注意该函数的返回值，是个**无符号整数**，如果 str1 比 str2 短，那么 strlen(str1)-strlen(str2) 并不是一个负数，而是一个非常大的正数

- `int strcmp(char* str1, char* str2)`

逐字符比较两个字符串的内容。这里强调：该函数的返回值不仅限于0, 1和-1！

- `char* strcpy(char* destination, char* source)` 和 `char* strncpy(char* dest, char*src, unsigned int num)`

使用要求：

1. source 指针和 destination 指针都不能为空，且指向的字符串都必须以空终止符结尾（即 '\0' ）。
2. destination 所指向的空间必须足够大。
3. source 字符串和 destination 字符串不可以有重叠

- `char* strstr(char* str1, char* str2)`

在 str1 中查找 str2。如果查找到了，返回一个指向 str1 中的指针，这个指针指向 str2 第一次在 str1 中出现的位置，如果没有查找到，返回 NULL。查找过程不会匹配空字符 '\0' ，并且会在该处停止查找。

代码

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int judge(char *substr, char *str) {
    char *s = substr, *p = str;
    while(*s) {
        if(*s != *p && *s != *p + 32 && *s != *p - 32)
            return 0;
        s++, p++;
    }
    return 1;
}
char buf[BUFSIZ], buf2[BUFSIZ];
char s1[BUFSIZ], s2[BUFSIZ];
int main() {
    FILE *in = NULL, *out = NULL;
    char *p = NULL;
    int l1, l2;
    out = fopen("fileout.txt", "w");
    in = fopen("filein.txt", "r");
    scanf("%s", s1);
    scanf("%s", s2);
    l1 = strlen(s1), l2 = strlen(s2);
    while(fgets(buf, BUFSIZ, in) != NULL) {
        p = buf;
        while(*p) {
            if(judge(s1, p)) {
                //从p开始长度为l1的字符串需要被替换
                *p = '\0';
                //置0是为了后面拼接s2
                strcpy(buf2, p + l1);
                //将后续部分先复制到buf2暂存
                strcat(buf, s2);
                strcat(buf, buf2);
                //s2数组自带结束符，因此可以直接将buf2接在后面
                memset(buf2, 0, BUFSIZ);
                //建议不需要考虑效率的时候多多清零，否则面对极端数据很容易出现问题
                p += l2;
                //直接偏移l2个位置，避免不必要的检查
            }
            else
                p++;
        }
        fprintf(out, "%s", buf);
        memset(buf, 0, BUFSIZ);
        //同上
    }
```

```
    fclose(in);
    fclose(out);
    return 0;
}
```

第三题 加密文件

- 使用 used 数组判断每个字符是否出现过，如之前已出现字符'a'，标记flag[a-97]=1，而不是每次都遍历一遍去判断，虽然现在感受不深，是因为数据规模小，当大作业来到的时候大家可以清晰感受到不同写法带来效率上的极大差别
- 使用数组下标的时候要统一，不要一会是97-122('a'-'z')，一会是0-25，这样容易给自己错误的心理暗示，有些问题也不易发现

```
#include <ctype.h>
#define MAX 50
char buf[BUFSIZ];
char key[MAX];
char str[MAX];
int used[MAX];
int main() {
    int i, pst = 0, ch, p;
    FILE *in = NULL, *out = NULL;
    in = fopen("encrypt.txt", "r");
    out = fopen("output.txt", "w");
    scanf("%s", str);
    for(i = 0; str[i]; i++) {
        p = str[i];
        if(!used[p - 'a']) {
            key[pst++] = p;
            used[p - 'a'] = 1;
        }
    }
    for(p = 'z'; p >= 'a'; p--) {
        if(!used[p - 'a'])
            key[pst++] = p;
    }
    while((ch = fgetc(in)) != EOF) {
        if(isalpha(ch))
            ch = key[ch - 'a'];
        fputc(ch, out);
    }
    fclose(in);
    fclose(out);
    return 0;
}
```

第四题 通讯录整理

- 利用双层循环来找到重复项并标号，每记录一条，就搜索先前所有的记录
- 如果搜索到 条相同名字的记录，那么将当前记录加上后缀 _n
- 如果搜索到名字和电话号都相同的记录，那么不保存当前记录
- 排序时以名字作为第一关键字，后缀作为第二关键字

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
typedef struct {
    char name[30];
    long long num;
    int repeat; // 重复次数，默认为0
} Person;
Person ar[101];
int cmp(const void *p1, const void *p2)
{
    Person *p = (Person *)p1, *q = (Person *)p2;
    int res = strcmp(p->name, q->name);
    return res ? res : p->repeat - q->repeat; // 名字如果不同则按名字的字典序排列，相同则按后缀递增排列
}
int main() {
    int t, cnt = 0;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%s %lld", ar[cnt].name, &ar[cnt].num);
        for (int i = 0; i < cnt; i++)
            if (!strcmp(ar[i].name, ar[cnt].name))
            {
                if (ar[i].num == ar[cnt].num)
                {
                    cnt--; // 如果电话号也一样则将记录个数减 1 （即删除当前记录）
                    break;
                }
                ar[cnt].repeat++;
            }
        cnt++;
    }
    qsort(ar, cnt, sizeof(ar[0]), cmp);
    for (int i = 0; i < cnt; i++)
    {
        printf("%s", ar[i].name);
        if (ar[i].repeat)
            printf("_%d", ar[i].repeat);
        printf(" %lld\n", ar[i].num);
    }
    return 0;
```

```
}
```

第五题 小型图书管理系统

- 本题一种做法是链表做法

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct node
{
    char title[55];
    char author[25];
    char ss[35];
    char year[15];
    struct node *next;
}*List;
List search1(List L,char a[])
{
    while(L!=NULL)
    {
        if(L->next==NULL||(strcmp(L->title,a)<0&&strcmp(L->next->title,a)>0))
            return L;
        L=L->next;
    }
    return NULL;
}
void check(List L,char b[])
{
    while(L!=NULL)
    {
        if(strstr(L->title,b))
        printf("%-50s%-20s%-30s%-10s\n",L->title,L->author,L->ss,L->year);
        L=L->next;
    }
}
List del(List L,char b[])
{
    List p=NULL,head=L;
    while(L!=NULL)
    {
        if(strstr(L->title,b))
        {
            if(p==NULL)
            {
                head=L->next;
            }
            else
            {
```

```
                p->next=L->next;
            }
        }
        else p=L;
        L=L->next;
    }
    return head;
}
int main()
{
    List p = NULL,head = NULL,q = NULL;
    FILE *in,*out;
    in=fopen("books.txt","r");
    out=fopen("ordered.txt","w");
    int i=0,j,n;
    p=(List)malloc(sizeof(struct node));
    while(~fscanf(in,"%s%s%s%s",p->title,p->author,p->ss,p->year))
    {
        p->next=NULL;//important
        if(head==NULL)
            head=p;
        else
        {
            if(strcmp(head->title,p->title)>0)
            {
                p->next=head;
                head=p;//
            }
            else
            {
                q=search1(head,p->title);
                p->next=q->next;
                q->next=p;
            }
        }
        i++;
        p=(List)malloc(sizeof(struct node));
    }
    n=i;

    int ol;
    char s[55];
    while(1)
    {
        scanf("%d",&ol);
        if(ol==0)
        {
            q=head;
            while(q!=NULL)
            {
                fprintf(out,"%-50s%-20s%-30s%-10s\n",q->title,q->author,q->ss,q-
>year);
```

```
                q=q->next;
            }
            break;
        }
        else if(ol==1)
        {
            p=(List)malloc(sizeof(struct node));
            scanf("%s%s%s%s",p->title,p->author,p->ss,p->year);
            p->next=NULL;//important
            if(head==NULL)
            head=p;
            else
            {
                //如果要插入结点书名的字典序比头结点小
                if(strcmp(head->title,p->title)>0)
                {
                    p->next=head;
                    head=p;
                }
                else
                {
                    //找到第一个下一个结点的书名字典序比p大的结点q
                    q=search1(head,p->title);
                    p->next=q->next;
                    q->next=p;
                }
            }
        }
        else if(ol==2)
        {
            scanf("%s",s);
            check(head,s);
        }
        else if(ol==3)
        {
            scanf("%s",s);
            head=del(head,s);
        }
    }
    fclose(in);
    fclose(out);
    return 0;
}
```

- 还有一种暴力做法，每进行一次操作都将所有图书排序
- 在维护图书信息时，利用变量 cnt 保存图书总个数，并且只关心下标小于 cnt 的记录

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

```c
#include <stdlib.h>
typedef struct {
    char name[51];
    char author[21];
    char press[31];
    char date[11];
} Book;
Book lib[505];
int cmp(const void *p1, const void *p2) {
    return strcmp(((Book *)p1)->name, ((Book *)p2)->name);
}
int main() {
    FILE *fin = fopen("books.txt", "r"), *fout = fopen("ordered.txt", "w");
    char sub[51];
    int cnt = 0, op, ended = 0;
    while (~fscanf(fin, "%s %s %s %s", lib[cnt].name,
lib[cnt].author,lib[cnt].press, lib[cnt].date))
        cnt++;
    while (!ended) {
        qsort(lib, cnt, sizeof(Book), cmp); // 每轮都重新排序
        scanf("%d", &op);
        switch (op) {
            case 0: // 结束操作，输出所有图书信息
                ended = 1;
                for (int i = 0; i < cnt; i++)
                    fprintf(fout, "%-50s%-20s%-30s%-10s\n",
lib[i].name,lib[i].author, lib[i].press, lib[i].date);
                break;
            case 1: // 直接插在最后，利用 qsort 来维护顺序
                scanf("%s %s %s %s", lib[cnt].name, lib[cnt].author,lib[cnt].press,
lib[cnt].date);
                cnt++;
                break;
            case 2: // 利用 strstr 函数进行查找
                scanf("%s", sub);
                for (int i = 0; i < cnt; i++)
                    if (strstr(lib[i].name, sub))
                        printf("%-50s%-20s%-30s%-10s\n", lib[i].name,lib[i].author,
lib[i].press, lib[i].date);
                break;
            case 3: // 用最后一项记录覆盖待删除记录，并将总数目减 1，利用 qsort 来维护顺序
                scanf("%s", sub);
                for (int i = 0; i < cnt; i++)
                    if (strstr(lib[i].name, sub))
                        lib[i] = lib[--cnt], i--;
                break;
        }
    }
    return 0;
}
```

一些附加内容

如何处理空白符

常见的**空白符**主要有四种：**空格 、制表** \t **、换行** \n **、回车** \r

以第二次作业最后一题为例，每条图书信息包括了四个空格分隔的字符串，因此就需要在读入时处理空格和换行。

许多同学会先整行读入，再逐个字符赋值，并利用空格和换行判断何时结束：

这样的做法有许多不好之处：

- 代码冗长，复用较多
- 处理下标和字符串结尾符时容易出错
- 难以处理空白符不明确的情况，比如两字符串间可用任意多空格、制表符分隔，或者是行末可能出现 \r\n

对于这种情况，处理空白符最好的办法就是“不去处理他们”，而是交给 scanf 系列的函数来解决，这样原本可能要写几十行的代码只需要一行完成： fscanf(file, "%s%s%s%s", book.name,book.author, book.pub, book.date); ，而仅仅是一个 %s 就可以同时完成处理空白符、赋值、添加结尾符。

利用格式字符串

格式字符串是 printf 或 scanf 系列函数中用于控制输入输出格式的字符串，两者规则略有不同。这里只介绍如何利用 scanf 的格式字符串处理空白符。

在 scanf 的格式字符串中，**一个空白符可以匹配任意多个空白符**，也就是说形如 " " 、 "\n" 、 "\t\n" 的格式字符串可以匹配 、 \t \t 、 \r\n 等字符序列。但为了简单易读，只用一个空格即可。

上面的 fscanf 函数的格式字符串中虽然没有出现空白符，但也能达到和 "%s %s %s %s" 相同的效果，这是因为 %d 、 %s 等格式字符会自动匹配**前面**的所有空白符（但 %c 并不会， scanf("%c", &c) 和 getchar() 作用相同，如果需要处理空白符则要写成 scanf(" %c", &c) ）

关于选择scanf还是gets/fgets的建议

在编程题中，读入数据是必不可少的，通常是我们程序的第一步。我们最常使用的读入函数就是 scanf和 gets/fgets ，虽然一般情况下，使用任意一种读入方式都可以解决问题，但是针对不同的输入格式采取不同的读入函数，往往可以简化读入后的预处理，同时减小写出 bug 的风险。从刚刚的例子中我们也能感受一二。

scanf

scanf 的优势在于：

1. 可以有效处理空白符(第一部分所提及)
2. 直接读入对应变量，省掉后续的处理

因为这两点原因，在可以使用 scanf 读取的时候，尽量使用 scanf 读取， scanf 对于字符串的处理相对更为完善，比起大家后续处理，相对不容易出现问题。只要输入格式清晰明确，就尽量使用 scanf 来读入。

gets/fgets

1. scanf 无法处理
2. 不需要对输入内容做预处理

> ATTENTION!!!
>
> **gets ()函数并不读取换行符'\n'**,它会把换行符替换成空字符'\0'
>
> **而fgets()函数读入末尾换行符'\n'**