

课程设计（论文）

题 目：数据结构课程设计（1.5+2.2）

专业：计算机科学与技术 指导教师：刘娜、路莹

学生姓名：梁宇龙 班级学号：计算机 182-07

序号	班级学号	姓名	完成题目及工作概述
1	计算机 182-07	梁宇龙	算法设计、编写、调试、运行、论文写作
2			

2019 年 12 月 27 日

摘 要

《数据结构》课程设计是本学习在进行 64 学时理论课程讲解及 8 学时实验后对课程内容的回顾与工程化项目建立的设计课程。

在本次课程设计中，本人共建立“文字编辑”与“约瑟夫环”两个项目的设计，主要运用《数据结构》路莹主编版第四章串及模式匹配和第三张第四节队列的相关知识，以及本课程中其他知识及 C 语言进行辅助。

在“文字编辑”项目中，运用 ASCII 码区间分别字母、数字与符号获取操作对象的信息并采用 KMP 算法进行子串对母本的模式比配，而后借助顺序表的功能实现母本中目标信息的删除。

“约瑟夫环”项目中，建立单循环链表，并根据链表的删除功能实现特定元素出队列并修改原队列的项目设计。

关键字：数据结构、字符串的模式匹配、单链表、队列

Abstract

The course design of data structure is a design course for the review of course content and the establishment of engineering project after 64 class hours of theoretical course explanation and 8 class hours of experiment.

In this course design, I set up two projects of "text editing" and "Josephus ring", mainly using the related knowledge of the fourth chapter string and pattern matching and the fourth section queue of the third page of data structure edited by Lu Ying, as well as other knowledge in this course and C language.

In the "text editing" project, the ASCII code interval is used to obtain the information of the operation object respectively by letters, numbers and symbols, and KMP algorithm is used to compare the mode of substring to the master, and then the target information in the master is deleted by the function of sequence table.

In the "Josephus ring" project, a single loop linked list is established, and the specific elements are queued and the original queue design is modified according to the deletion function of the linked list.

Key Words: Data structure, pattern matching of string, single chain table, queue

目 录

摘 要.....	I
Abstract.....	II
第一章 题目 1.5：文字编辑.....	1
1 任务.....	1
2 要求分析.....	1
3 概要设计.....	1
4 详细设计.....	3
5 调试分析.....	3
6 测试结果.....	4
7 用户使用说明.....	5
第二章 题目 2.2：约瑟夫环.....	1
1 要求分析.....	7
2 概要设计.....	7
3 详细设计.....	8
4 调试分析.....	8
5 测试结果.....	9
6 用户使用说明.....	9
课设总结.....	10
附录 A 源代码.....	11

第一章 题目 1.5：文字编辑

1 任务

输入多行字符，可以是大写、小写的英文字母、任何数字及标点符号。

2 要求分析

(1) 分行输出用户输入的各行字符；

要求分行，则不能用传统的“enter 键入”作为输入完成的标志，因此在设计中采用#键代替“\n”作为键入完成的判断条件。

(2) 分别统计“英文字母数”，“数字个数”，“空格个数”，“文章总字数”；

区分字母和数字等在元素在数据中采用 ASCII 码中上述几项均在不同数值区间内的特性来完成对不同元素的统计方法。

(3) 利用 KMP 算法，统计某一字符串在文章中出现的次数，并输出该次数；

利用 KMP 算法，则先求出子串的 next 值与 nextval 值以便于提高子串与母本匹配的速度与精确度，在该算法中每成功一次匹配要进行一次记录，最后根据记录数值确定子串在母本中的出现次数。

(4) 删除某一子串，并将后面的字符前移，输出删除后剩下的字符。

删除子串则需先判定子串在母本中的位置，先使用 (3) 算法找到并定位子串在母本中的位置，而后利用顺序表的数值覆盖完成对子串的删除功能。

3 概要设计

(1) 数据类型的定义与含义：

结构体 line：由字符型指针 data 和结构体 line 类型指针 next 组成，作为文章每一行头指针和行内字符索引线索。

字符型数组 q[50]、str[50]：存储查找或删除的字符存储。

整数类型指针 next、nextval：存储子串的 next 值和 nextval 值，便于进行 KMP 算法

结构体 line 型指针 head、p：用于查找文章以及各行。

此外还有整型变量 i 用于进行循环控制。

(2) 主程序流程：

- ①定义相应需要变量，调用 Creat 函数进行文章输入与存储
- ②对输入的文章进行输出
- ③调用 Letter 函数对字母个数、数字个数等进行统计并输出
- ④输入要查到的子串并调用 StrLength 函数进行计数
- ⑤调用 SearchStr 函数查找出现次数，并输出次数
- ⑥输入要删除的子串并调用 DeleteStr 函数进行删除操作
- ⑦输出删除后的文章，程序结束。

(3) 各程序模块功能：

StrLength 函数：计算字符串长度

strAssgin 函数：将输入的字符串存储进文章中

GetNext 函数：求出子串的 next 值

Index_KMP 函数：用 KMP 算法求出子串所在位置

Creat 函数：进行文章的输入与存储

Letter 函数：对文章字母、数字、空格和总字数进行统计

SearchStr 函数：进行查找目标串在文章中出现的次数

DeleteStr 函数：将目标子串在文章中删除

(4) 层次调用关系：

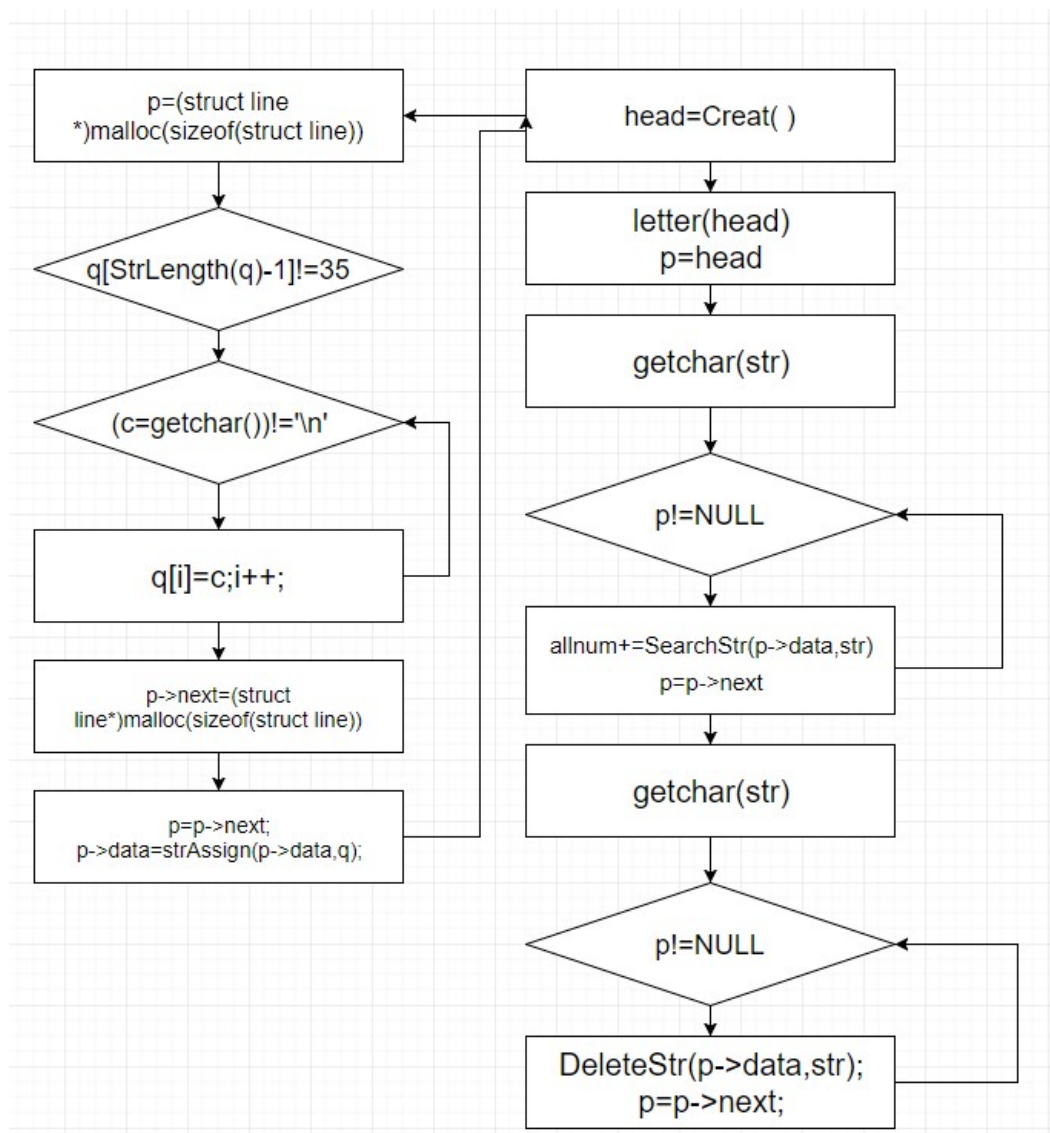
Creat 函数中调用 StrLength、strAssign 函数

Letter 函数中调用 StrLength 函数

SearchStr 函数中调用 StrLength、GetNext、Index_KMP 函数

DeleteStr 函数中调用 StrLength、GetNext、Index_KMP 函数

4 详细设计



5 调试分析

① 正确输入与输出测试：

输入文章为 asdasdasd#，输出文章正确，字母数、数字数、空格数与文章总字数输出正确。输入 asd 为查找字符串，输出出现次数正确。输出删除字符串 asd，输出结果正确。通过正确调试。

② 试错输入测试：

(1) 错误输入文章测试：输入 6 个回车键后#测试，输出结果正常，文章总字数出现错误。输入查找字符串输出结果正确，输入删除字符串输出结果正确。

(2) 错误输入查找字符串、删除字符串：输入文章输出结果正确，以###作为查找字符串和删除字符串，输出结果正常。

③对于错误输出的调整：

将判断文章总字数条件从判断字符串长度改为判断字母与数字个数。进行再次试错调试：输入六个回车后#进行测试，字母数、数字数、空格数、文章总字数输出正确，输入查找字符串输出正确，输入删除字符串目标正确。

④时间复杂度分析：

文章中字符串的最大长度均为 50，行数 n 动态开辟，若对每一个元素进行操作，则算法复杂对应为 $O(50n)$ ，在算法设计中，为减小时间复杂度，在行内将循环判定条件均改为以遇到 '\0' 结束循环来降低不必要位置对算法执行次数的影响。

⑤空间复杂度分析：

本程序设计采用多自定义算法嵌套执行模式。所有循环变量与搜索指针均定义为自定义函数内变量，减小不必要空间。影响本程序空间复杂度的因素为每行输入元素个数与预定义 50 个的差，通过分析思考，希望实现通过判定每一行输入个数动态开辟行内空间，但还没实现算法，接下来会持续跟进。

6 测试结果

测试计划：

进行 10 次测试，分别为 5 次正确输入测试及 5 次试错测试。试错测试包括输入文章错误测试、输入查找字符串错误测试、输入删除字符串错误测试、文章行内输入超过规定字数测试、大数据量测试。

测试结果（部分）：

C:\Users\yulon\Desktop\梁_1.5: 文章编辑.exe

请输入一页文章，以#结尾（每行最多50字符）：
qwertyuiop
asdfghjkl
zxcvbnm#

输出的文章为：
qwertyuiop
asdfghjkl
zxcvbnm

全部字母数为26

数字个数为0

空格个数为0

文章总字数为26

请输入要查找的字符串：asd

目标字符串出现次数为1

请输入要删除的某一字符串：asd

删除后的文章为：
qwertyuiop
fghjkl
zxcvbnm

C:\Users\yulon\Desktop\梁_1.5: 文章编辑.exe

请输入一页文章，以#结尾（每行最多50字符）：
asdadasdasd#

输出的文章为：
asdadasdasd

全部字母数为12

数字个数为0

空格个数为0

文章总字数为12

请输入要查找的字符串：asd

目标字符串出现次数为4

请输入要删除的某一字符串：asd

删除后的文章为：

C:\Users\yulon\Desktop\梁_1.5: 文章编辑.exe

请输入一页文章，以#结尾（每行最多50字符）：
jahjdjhjashdjkdhasjdhasjkdhasjhdjashdhasjkdhasjkdjhjashdjasdh
每行最多输入50字符！

C:\Users\yulon\Desktop\梁_1.5: 文章编辑.exe

请输入一页文章，以#结尾（每行最多50字符）：
12312i3u123u12u3i12u3iolui2ui3o1#

输出的文章为：
12312i3u123u12u3i12u3iolui2ui3o1

全部字母数为13

数字个数为19

空格个数为0

文章总字数为32

请输入要查找的字符串：123

目标字符串出现次数为2

请输入要删除的某一字符串：123

删除后的文章为：
12i3uu12u3i12u3iolui2ui3o1

C:\Users\yulon\Desktop\梁_1.5: 文章编辑.exe

请输入一页文章，以#结尾（每行最多50字符）：
123123123123123#

输出的文章为：
123123123123123

全部字母数为0

数字个数为15

空格个数为0

文章总字数为15

请输入要查找的字符串：123

目标字符串出现次数为5

请输入要删除的某一字符串：123

删除后的文章为：

7 用户使用说明

本程序均含有提示语句引导用户操作，具体操作步骤如下。

- (1) 程序界面提示“请输入一页文章，以#结束”，引导用户输入文章。若超过每行规定字符，则提示“超出 50 字符”并退出程序。
- (2) 用户完成输入文章操作后，程序自动进行文章输出，并统计字母数、数字数、空格数、文章总字数并输出。
- (3) 系统提示用户输入要查找的字符串，引导用户输入。
- (4) 用户进行查找字符串输入，完成后程序输出字符串出现次数。
- (5) 系统提示用户输入要删除的字符串，引导用户输入。
- (6) 用户进行删除字符串输入，完成后程序输出删除目标字符串后的文章，所有程序执行结束。

第二章 题目 2.2：约瑟夫环

1 要求分析

要求：约瑟夫环问题是一个数学的应用问题：已知 n 个人（以编号 $1, 2, 3 \dots n$ 分别表示）围坐在一张圆桌周围。从编号为 k 的人开始报数，数到 m 的那个人出列，他的下一个人又开始报数，数到 m 的那个人又出列，依次规律重复下去，圆桌周围的人全部出列。要求采用循环链表实现约瑟夫环。

分析：建立循环单链表，当指定元素出队列后使出队列元素的前一个元素的线索指向出元素的线索，实现单链表的改造，当该结点线索指向自身时，即证明链表中只剩一个元素，输出后即可实现约瑟夫环的输出效果。

2 概要设计

（1）数据类型的定义与含义：

整型变量 n 、 m 、 k 用于定义总人数、炸弹数字以及起始人

结构体 `node`：包含整型变量 `data` 和结构体 `node` 型变量 `next`，作为单链表的一个结点。

（2）主程序流程：

- ①对整型变量定义并输入总人数 n 、炸弹数字 m 、起始人 k
- ②根据总人数 n 调用 `creat` 函数建立循环单链表
- ③根据 k 的值找到对应的 `p->data`，并将指针 `p` 指向起始点
- ④调用 `result` 函数输出约瑟夫环输出结果，程序结果

（3）各程序模块功能：

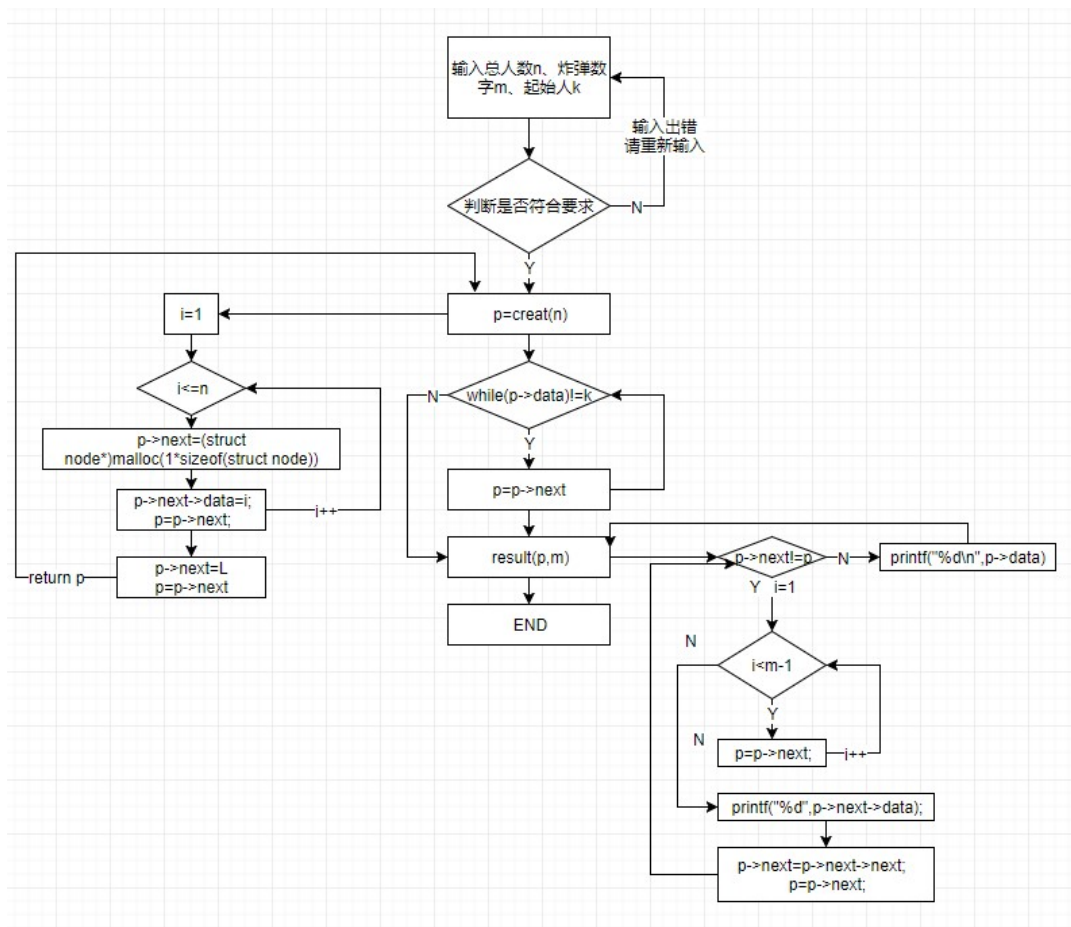
`creat` 函数：建立循环单链表，并对对应 `data` 进行赋值

`result` 函数：根据循环单链表，按约瑟夫环输出规则进行值的输出与单链表的更改。

（4）层次调用关系：

`main` 函数中调用 `creat` 函数和 `result` 函数。

3 详细设计



4 调试分析

①正确输入与输出测试：

输入总人数为 6、炸弹数字为 4、起始人为 2：输出约瑟夫环结果正常。

②试错输入测试：

(1) 错误输入总人数测试：输入总人数为-3，程序提示“总人数出错，请重新输入”，正确输入后恢复正常操作。

(2) 错误输入炸弹数字测试：输入总人数为 10，炸弹数字为-8，程序提示“炸弹数字出错，请重新输入”，正确输入后恢复正常操作。

(3) 错误输出起始人测试：输入总人数为 10，炸弹数字为 6，起始人为 19，程序提示“起始人出错，请重新输入”，正确输入后恢复正常操作。

③时间复杂度分析:

在 creat 算法中, 对于创建结点的算法时间复杂度为 $O(n)$; 对于输出约瑟夫环的结果算法时间复杂度为 $O(mn)$ 。

④空间复杂度分析:

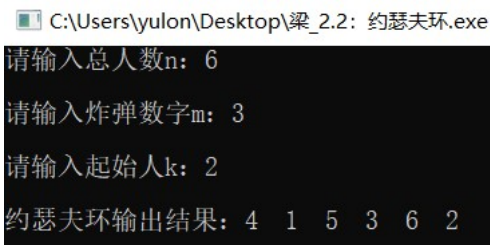
本程序使用单链表作为存储元素的方法, 每个结点均通过 n 的值动态开辟, 极大程度上降低了本次程序的空间复杂度。

5 测试结果

测试计划:

由于在调试过程中已经进行全部输入试错测试, 因此本次测试不再进行试错测试, 故本次测试进行 8 次正常数值测试与 2 次较大数据测试。

测试结果 (部分):



C:\Users\yulon\Desktop\梁_2.2: 约瑟夫环.exe
请输入总人数n: 6
请输入炸弹数字m: 3
请输入起始人k: 2
约瑟夫环输出结果: 4 1 5 3 6 2



C:\Users\yulon\Desktop\梁_2.2: 约瑟夫环.exe
请输入总人数n: -3
总人数出错, 请重新输入: 4
请输入炸弹数字m: -3
炸弹数字出错, 请重新输入: 2
请输入起始人k: 9
起始人出错, 请重新输入: 2
约瑟夫环输出结果: 3 1 4 2

6 用户使用说明

本程序均含有提示语句引导用户操作, 具体操作步骤如下。

(1) 程序界面提示“请输入总人数 n:”, 引导用户输入数值。如果输入数值不符合要求, 系统提示重新进行输入。

(2) 用户完成输入操作后, 系统提示用户“请输入炸弹数字 m:” 引导用户输入数值。如果输入数值不符合要求, 系统提示重新进行输入。

(3) 用户完成输入操作后, 系统提示用户“请输入起始人 k:” 引导用户输入数值。如果输入数值不符合要求, 系统提示重新进行输入。

(4) 输出约瑟夫环输出结果, 程序结果。

课设总结

本次课程设计针对本学期计算机专业基础课数据结构中的部分章节进行了知识回顾和算法实现，不仅回顾了学过的知识，也增强了自己在程序设计和算法构建过程的能力。

在本次课程设计中主要遇到的问题如下：

(1) 对于数据存储：在“文字编辑”项目中，开始由于自己思考较少，采用一元数组进行存储，但在查找和删除操作带来了许多不便利和错误的操作，后通过资料查阅和分析，确定以头指针索引，行内元素索引为存储模式进行存储，解决了母本查找的不便利性，也在一定程度上降低了空间复杂度。

(2) 对于算法的调用问题：在“文字编辑”项目中，在进行查找算法和删除算法中需调用 `next` 函数和 `Index_KMP` 函数，开始时对函数调用关系进行了混淆，导致了死循环。后来经过制作函数调用关系图，确定了函数的调用，进而对程序进行改写，突破了在该问题上的瓶颈。

(3) 对于程序连通问题：在“文字编辑”项目中，文章输入并存储结束后，所剩的辅助空间在后续操作中并不需要，且仍需开辟新的辅助空间，导致了空间的浪费以及函数连通性降低。解决办法：采用自定义函数的多级调用，使得辅助空间在该函数结束后直接释放。

在本次课程设计中，我对于程序设计有了全新的理解，在程序设计中，最重要的并不是如何快速的将代码通过编译和运行，而是在程序设计过程中多考虑可能出现的情况，并分功能设计出函数框架，在通过函数间关系设计逐渐充实程序，最终完成连通性的修改，使得代码能够运行，完成程序的设计。

对课程的认知：通过本次课程设计，我再次理解了在课程初老师所说的数据结构的定义：数据间的关系。我所完成的两个项目均是线性表的运用，充分的思考了程序中对于数值间前驱和后继的关系，收货颇丰，也为自己在本领域有了更深的见解。

附录 A 源代码

题目 1.5：文字编辑

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

struct line{                                     //定义line链表作为头指针分行和
行内字符输入
    char *data;
    struct line *next;
};

int StrLength(char *str){                        //该函数用来计算字符串的长度
    int i=0;
    while(str[i]!='\0'){                        //以字符串结尾标识“\0”作为结束
标志
        i++;
    };
    return i;
}

char *strAssign(char *s, char *t){              //将输入的字符串存储进文章中
    int i, j;
    int len=StrLength(t);
    s=(char*)malloc((len+1)*sizeof(char));      //根据字符串长度动态开辟连续存
储单元
    if(s!=NULL){
        for(i=0; i<len; i++){
            s[i]=t[i];
        }
        s[len]='\0';                            //存储结束以“\0”结尾
    }
    return s;
}

int *GetNext(char *t){                          //该函数计算子串的next值
    int *nextj;
    int i=1, j=0;
    nextj=(int *)malloc((StrLength(t)-1)*sizeof(int)); //根据字符串长度动态开辟连
续存储单元
```

```

nextj[1]=0;
while(i<t[0]) {
    if(j==0||t[i]==t[j]) {
        ++i;++j;
        nextj[i]=j;
    }
    else{
        j=nextj[j];
    }
}
return nextj;
}

int Index_KMP(char *s, char *t, int pos, int *nextj) { //该函数用KMP算法计算子串在
母本中的所在位置
    int i=pos, j=1, count=1;
    while(i<=s[0]&&j<=t[0]) { //进行子串与母本的匹配
        if(j==0||s[i]==t[j]) {
            ++i;++j;
        }
        else{
            j=nextj[j];
        }
    }
    if(j>t[0]) {
        return i-t[0];
    }
    else{
        return 0;
    }
}

struct line *Creat() { //进行文章创建函数
    struct line *p,*head;
    char q[50];
    int c, i=1;
    printf("请输入一页文章，以#结尾（每行最多50字符）：\n");
    p=(struct line *)malloc(sizeof(struct line)); //开辟存储每一行的索引结点
    head=p;
    while(1) {
        fflush(stdin);

```



```

        i=1;
        while((c=getchar())!='\n') {                                //输入行内元素
            q[i]=c;
            i++;
        }
        q[i]='\0';
        q[0]=StrLength(q)-1;
        if(StrLength(q)>50) {
            printf("每行最多输入50字符！");
            break;
        }
        if(q[0]==35)break;
        //如果发现#退出输入
        p->next=(struct line*)malloc(sizeof(struct line));
        //将指针指向下一行头指针
        p=p->next;
        p->data=strAssign(p->data, q);
        //将输入内容存入头指针中
        if(q[StrLength(q)-1]==35) {
            p->data[StrLength(q)-1]='\0';
            break;
        }
    }
    p->next=NULL;
    head=head->next;
    return head;
}

void Letter(struct line*head) {
    //进行文章的计数功能
    struct line *p=head;
    int i, let=0, num=0, spa=0, all=0;
    while(p!=NULL) {
        for(i=1; i<=StrLength(p->data); i++) {

            if((p->data[i]>='a' && p->data[i]<='z') || (p->data[i]>='A' && p->data[i]<='Z')) {
                //判断该字符为字母
                let++;all++;
            }
            else if(p->data[i]>=48&&p->data[i]<=57) {

```

```

//判断该字符为数字
        num++;all++;
    }
    else if(p->data[i]==32){
//判断该字符为空格
        spa++;
    }
}
p=p->next;
}
printf("\n全部字母数为%d\n", let);
printf("\n数字个数为%d\n", num);
printf("\n空格个数为%d\n", spa);
printf("\n文章总字数为%d\n", all);
}
int SearchStr(char *data, char *str) {
//进行子串在母本出现次数查询算法
    int j=StrLength(str)-1;
    int i;
    int time=0;
    int result;
    int k, count=1;
    int *nextj;
    do{
        nextj=GetNext(str);
//计算子串字符的next值
        result=Index_KMP(data, str, 1, nextj);
        while(result!=0){
            time++;
            result=Index_KMP(data, str, result+StrLength(str)-1, nextj); //利
用KMP算法找到子串所在位置
        }
    }while(result<StrLength(data)-1&&result!=0); //当
没有全部执行完时循环执行方便查找
    return time;
}
void DeleteStr(char *data, char *str) {
//进行目标子串在母本中的删除操作
    int j=StrLength(str)-1;

```

```

int i;
int result;
int k, count=1;
int *nextj;
do{
    nextj=GetNext(str);
    //求出子串字符的next值
    result=Index_KMP(data, str, 1, nextj);
    //利用KMP算法找到子串所在位置
    if(result!=0){
        i=StrLength(data)-1;
        if((result+j)>i){
            data[result]='\0';
        }
        else{
            for(k=result+j;k<=i;k++){
                data[k-j]=data[k];
            }
            data[i-j+1]='\0';
        }
        data[0]=i-j;
    }
}while(result<StrLength(data)-1&&result!=0); //遍历所有字符，当没有全部执行完时循环执行方便查找
}

int main(){
    int i, c;
    struct line *head, *p;
    char str[50], str1[50], str2[50];
    int *next, *nextval, *result;
    head=Creat();
    //引入creat函数创建文章链表
    printf("\n输出的文章为: \n");
    p=head;
    while(p!=NULL){
        for(i=1; i<=p->data[0]; i++){
            printf("%c", p->data[i]);
        }
        printf("\n");
    }
}

```

```

        p=p->next;
    }
    //对文章进行输出
    Letter(head);
    printf("\n请输入要查找的字符串：");
    fflush(stdin);
    //删除键盘缓冲符
    i=1;
    while((c=getchar())!='\n'){
        //进行子串输入操作
        str[i]=c;
        i++;
    }
    str[i]='\0';
    str[0]=StringLength(str)-1;
    p=head;
    int allnum=0;
    while(p!=NULL){
        allnum+=SearchStr(p->data, str);
        //搜索每一行子串出现次数
        p=p->next;
    }
    printf("\n目标字符串出现次数为%d\n", allnum);
    p=head;
    printf("\n请输入要删除的某一字符串：");
    fflush(stdin);
    i=1;
    while((c=getchar())!='\n'){
        //输入子串
        str[i]=c;
        i++;
    }
    str[i]='\0';
    //使子串以“\0”结尾
    str[0]=StringLength(str)-1;
    p=head;
    while(p!=NULL){
        //进行子串删除操作
        DeleteStr(p->data, str);
    }

```

```

        p=p->next;
    }
    p=head;
    printf("\n删除后的文章为: \n");
    //输出删除后的文章
    while(p!=NULL) {
        for(i=1;i<=p->data[0];i++) {
            printf("%c", p->data[i]);
        }
        printf("\n");
        p=p->next;
    }
    return 0;
}

```

题目 2.2：约瑟夫环

```

#include<stdio.h>
#include<stdlib.h>
struct node{                                //定义单链表的
每个结点结构
    int data;
    struct node* next;
};
void result(struct node*p, int m){          //进行约瑟夫环
输出
    printf("\n约瑟夫环输出结果: ");
    while(p->next!=p) {
        for(int i=1;i<m-1;i++){            //使指针
指向目标结点的前一个节点
            p=p->next;
        }
        printf("%d ", p->next->data);        //输出目标结点
的值
        p->next=p->next->next;              //将输出结点移
除出单链表
        p=p->next;
    }
    printf("%d\n", p->data);

```

```

}
struct node* creat(int n) { //进行循环单链
表的创建
    struct node *p,*L;
    for(int i=1;i<=n;i++) {
        if(i==1) { //判断如
果是第一个开辟的结点，则将头指针赋值给该节点
            L=(struct node*)malloc(1*sizeof(struct node));
            p=L;
            p->data=i;
        }
        else{
            p->next=(struct node*)malloc(1*sizeof(struct node)); //根据结点个数动态
开辟结点
            p->next->data=i;
            p=p->next;
        }
    }
    p->next=L;
    p=p->next;
    return p;
}

int main() {
    int n;
    printf("请输入总人数n: "); //进行总人数输
入
    scanf("%d",&n);
    while(n<=0) { //如果输入不符
合规则，进行重新输入
        printf("\n总人数出错,请重新输入: ");
        scanf("%d",&n);
    }

    struct node* p=creat(n); //根据总人数n
开辟n个结点的循环单链表
    printf("\n请输入炸弹数字m: ");
    int m;
    scanf("%d",&m);
    while(m<=0) { //如果输入不符
合规则，进行重新输入

```

```

        printf("\n炸弹数字出错, 请重新输入: ");
        scanf("%d", &m);
    }
    printf("\n请输入起始人k: ");
    int k;
    scanf("%d", &k);
    while(k<=0 || k>=n) { //如果输
        入不符合规则, 进行重新输入
        printf("\n起始人出错, 请重新输入: ");
        scanf("%d", &k);
    }
    while(p->data!=k) {
        p=p->next;
    }
    result(p, m);
    return 0;
}

```