# 数据结构实验一实验报告

## 需求分析

1. 从键盘输入整数m，通过create函数生成一个具有m个结点的单向环表。环表中的结点编号依次为1，2，……，m。

对于问题1来说，需要我先创立一个结构体包含：int data 来存储结点的编号，然后再创立一个指针来指向下一个结点。

而这里面值得注意的是本次链表要求是单向环表，所以我还需要将最后一个结点指向头结点，即p->next=head;

1. 从键盘输入整数s（1<=s<=m）和n，从环表的第s个结点开始计数为1，当计数到第n个结点时，输出该第n结点对应的编号，将该结点从环表中消除，从输出结点的下一个结点开始重新计数到n，这样，不断进行计数，不断进行输出，直到输出了这个环表的全部结点为止。

例如，m=10，s=3，n=4。则输出序列为：6，10，4，9，5，2，1，3，8，7。

对于问题二，需要我们在已经调用create函数创建单向环表后读入整数s和n，然后记第s个结点为第一个结点，从s开始计数到第n个结点后输出该结点的编号，并且调用free()函数消除该结点，然后继续从下一个结点开始重新计数到n不断计数输出，直到输出环表的全部结点。

## 概要设计

对于该程序，我设计了一个包含数字类型int的数值符号data，指向下一个结构体的指针next的结构体NODE。

然后设计了创建单向环表的函数NODE\* create(int m)，来创建一个包含m个结点，编号依次为1、2、3......m的单向环表。

再设计函数void josephus\_circle(NODE\* head, int s, int n)，用于输出约瑟夫环。

最后在主函数main里面先读入m，s，n，并且对m，s，n的格式是否满足题目要求做出判断，对于错误情况做出提示并且重新读入，然后调用create函数创建单向环表，再调用josephus\_circle输出约瑟夫环，代码完毕。

## 详细设计

对于结构体NODE，我先使用int data把data定义为整数型，然后使用struct node\* next将next定义为指向下一个结构体的指针。

对于函数NODE create(int m),我先定义了表头head，然后定义了两个结点p和q，使用malloc函数对表头赋存储空间，并且先定义表头所对应的编号head->data=1，此后令p=head，开始用p来创建单向环表。

由于已经给head->data赋初值1，并且函数前提已经得到链表总长度m，故我使用for(int i=2;i<=m;i++){

q=(NODE\*)malloc(sizeof(NODE));

q->data=i;

p->next=q;

p=q;

}

这一代码来创建单向环表，首先令i从2开始，并且设i小于等于m，每一次循环都先给q分配一个存储空间，然后将q的data值赋予对应的标号，并且令p的指针指向q，这样将q表示为p的下一个结点，然后再令p=q，即令p为下一个结点来继续循环函数。值得注意的是，我在循环结束后令p->next=head，这样可以成功的将表尾的指针指向表头，使得链表构成单向环表。

最后，create函数返回的是表头的地址。

对于void josephus\_circle(NODE\* head, int s, int n)函数，首先我们已经得到了表头，s和n，这时我们设置一个结点pre为表头的地址，再设一个空节点p，首先我们需要做的是找到初始的s，即找到初始编号为1的结点，我们使用 for(int i=1;i<s;i++){

pre=pre->next;

}

这一代码可以得到编号为1的结点，他令i=1，i<s，将pre从表头开始以此迭代最终得到初始1.然后我使用代码 while(pre->next!=pre){

for(int i=1;i<n;i++){

p=pre;

pre=pre->next;

}

printf("%d ",pre->data);

p->next=pre->next;

free(pre);

pre=p->next;

}

来输出除最后一个外的所有编号。他的判定条件pre->next!=pre的原理为，当pre->next!=pre时，说明该链表还有大于等于2个的结点，但是如果pre->next==pre，那么说明该链表只剩一个结点了。

我们讨论链表还有大于等于2个的结点的情况，在这种情况下，我们设i=1；i<n，i++，在这个循环下依次得到第n个编号的结点，我们使用p=pre；pre=pre->next，可以使得当得到第n个编号的结点时pre为第n个编号的结点，而p为第n-1个编号的结点，这使得p->next=pre->next;这一代码可以成功的在消除pre（即第n个结点）的情况下依然能够链接整个链表，我们先输出该结点的编号，然后链接剩余结点，最后用free函数删除结点，当到了最后一个结点时，直接 printf("%d\n",pre->data);

输出该结点的编号，然后free(pre)删除结点，该函数无需返回值。

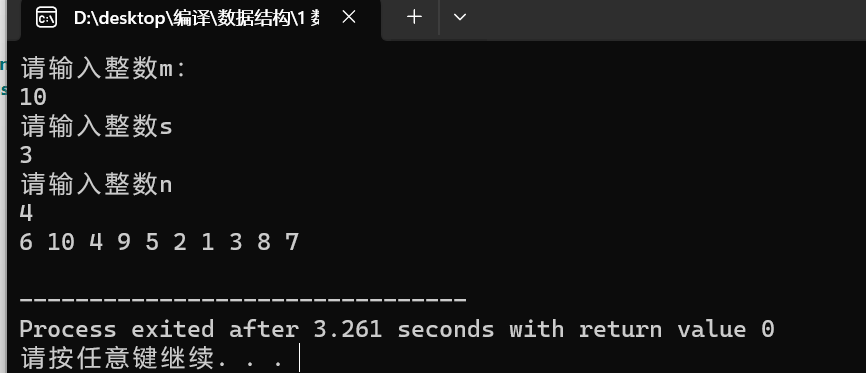
对于main函数，我设置了三个状态，ff、ss和nn，分别对应读入m、读入s、读入n，首先根据提示“请输入整数m：”读入大于等于0的整数m，如果符合条件则继续，如果不符合条件则返回ff重新读入m，并且输出报错“m输入错误——你应该输入大于零的自然数”。然后根据提示“请输入整数s”输入 整数s，注意s必须取值在1到m之间，如果s小于1或者s大于m，那么输出“s输入错误，请输入在1~m间的整数s”报错并且返回状态ss。最后根据提示“请输入整数n”输入n，注意n必须取值在1到m之间，如果n小于1或者n大于m，那么输出“n输入错误，请输入在1~m间的整数n”报错并返回状态nn。

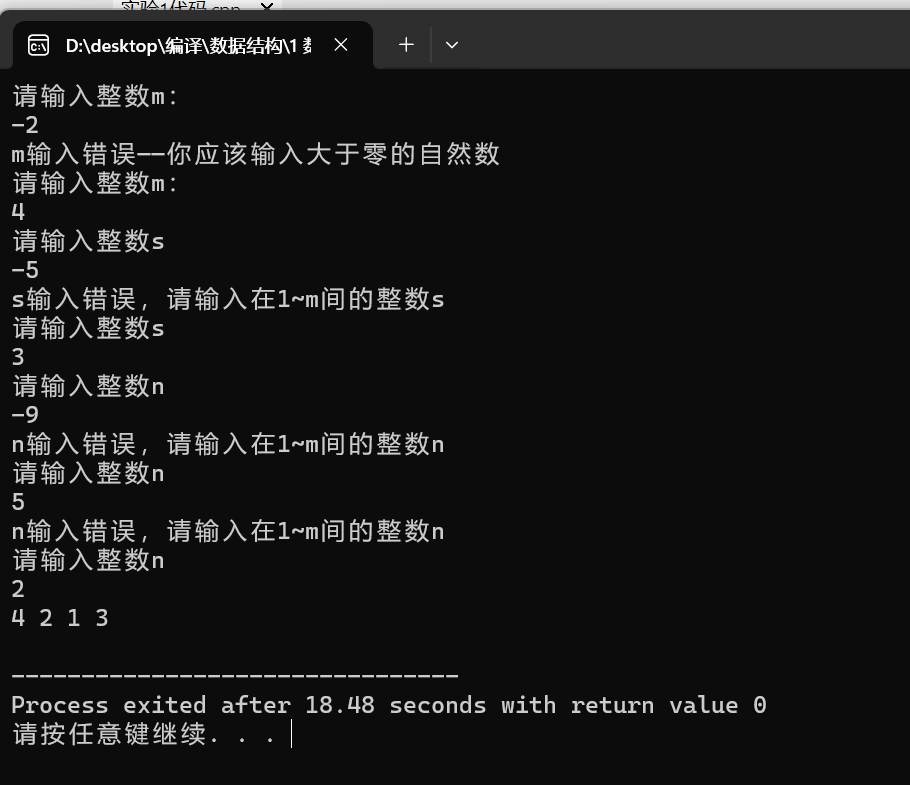
在这之后调用 NODE \*head =create(m);函数创建单向环表，然后调用 josephus\_circle(head,s,n);函数输出约瑟夫环，然后结束函数。

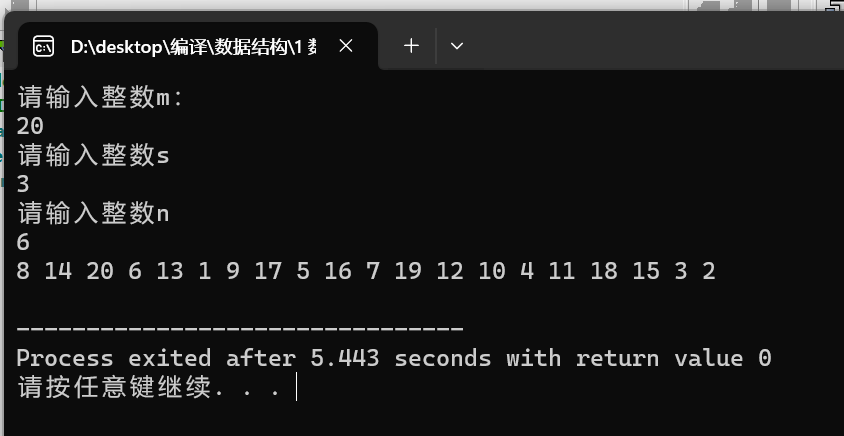
## 调试分析

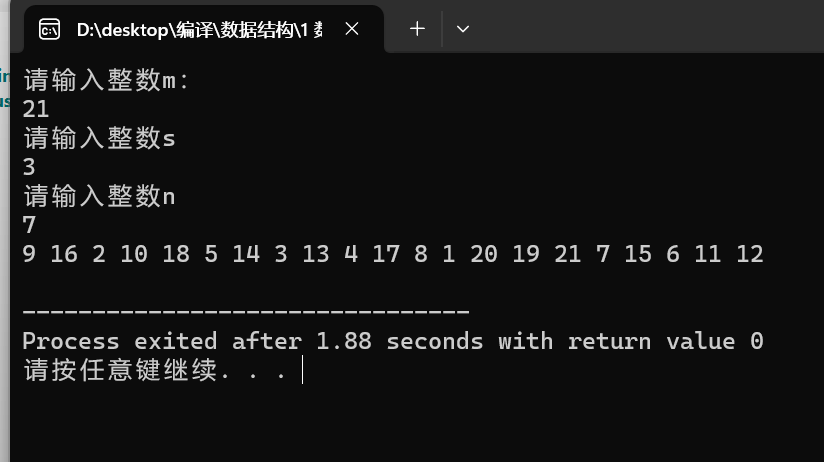
在代码运行的过程中，我也遇到了很多问题，首先在链表代码的写作中我们都经常遇到无输出的情况，经过调试和检查后，我发现链表无输出往往在于我们对于各个结点之间的链接不够，或者在无意中更改了头结点的地址，通过调试和验算后我最终成功的完成了代码！

## 测试结果

如图：







## 附录

#include <stdio.h>

#include <stdlib.h>

typedef struct node{

int data;

struct node\* next;

}NODE;

NODE\* create(int m){

NODE \*head=NULL;

NODE \*p=NULL;

NODE \*q=NULL;

head =(NODE\*)malloc(sizeof(NODE));

head->data=1;

p=head;

for(int i=2;i<=m;i++){

q=(NODE\*)malloc(sizeof(NODE));

q->data=i;

p->next=q;

p=q;

}

p->next=head;

return head;

}

void josephus\_circle(NODE\* head, int s, int n) {

NODE \*pre=head;

NODE \*p=NULL;

for(int i=1;i<s;i++){

pre=pre->next;

}

while(pre->next!=pre){

for(int i=1;i<n;i++){

p=pre;

pre=pre->next;

}

printf("%d ",pre->data);

p->next=pre->next;

free(pre);

pre=p->next;

}

printf("%d\n",pre->data);

free(pre);

}

int main(){

int m,s,n;

// 读入m

ff: printf("请输入整数m：\n");

if (scanf("%d", &m) == 1){

if(m<=0){

printf("m输入错误——你应该输入大于零的自然数\n");

goto ff;

}

}

// 读入s

ss: printf("请输入整数s\n");

if(scanf("%d", &s) == 1){

if(s<1||s>m){

printf("s输入错误，请输入在1~m间的整数s\n");

goto ss;

}

}

// 读入n

nn:printf("请输入整数n\n");

if(scanf("%d", &n) == 1){

if(n<1||n>m){

printf("n输入错误，请输入在1~m间的整数n\n");

goto nn;

}

}

NODE \*head =create(m);

josephus\_circle(head,s,n);

return 0;

}