**2019秋季学期**

**数据结构实验报告**

**实验一 线性表实验**

**班级：\_\_\_\_\_计实验19\_\_\_\_\_\_\_**

**学号：\_\_\_\_\_19101130122\_\_\_\_\_\_\_**

**姓名：\_\_\_\_\_陈旻杰\_\_\_\_\_\_\_**

|  |
| --- |
| **评语：**  **日期：** |

**实验目的：**

熟悉线性表的基本运算在顺序存储结构和链式存储结构上的实现，其中重点掌握线性表的链式表示时各种操作的实现。

1、约瑟夫

**问题描述**：

约瑟夫（Joseph）问题的一种描述是：编号为1，2，3，…，n的n个人按顺时针方向围坐一圈，每人持有一个密码〈正整数〉，一开始任选一个正整数作为报数上限值m，从第一个人开始按顺时针方向自1开始顺序报数，报到m时停止报数，报m的人出列，将他的密码作为新的m值，从他在顺时针方向上的下一个人开始重新从1报数，如此下去，直至所有人全部出列为止。试设计一个程序求出出列顺序。

**需求分析：**（包括对问题的理解，解决问题的策略、方法描述）

题意：这题相当于一堆人站一圈，初始权值，然后循环报数，报数结束时所在人出局，并以出局的人的值为新的权值。依次循环，直到所有人出局游戏结束。

题解：建立单向循环链表，从头结点的下一个节点开始遍历，遍查找权值数的节点并删除该节点，进行下一次查找，直到该链表为空。步骤：建立单向循环链表->查找->删除。

**系统设计：**（包括数据结构定义、抽象出基本操作描述、主程序模块处理过程描述）

数据结构定义：

struct pos

{

int num,pos;

struct pos\* next;

};

抽象出基本操作描述

1.初始化链表

初始条件：head不存在

操作结果：创建空链表头

2.链表赋值

初始条件：head存在

操作结果：每次输入数据就创建一个新节点并与先前的链表链接

3.查找目标节点

初始条件：head存在且非空

操作结果：查找n个结点的head中从结点p起报数为m的元素，并标记

4.删除目标节点

初始条件：head存在且非空

操作结果：删除标记的节点，然后更改报数数值m

5.释放内存

主程序模块处理过程描述

1.创建线性链表L。

1. 输入m,n

3.随着数据输入创建新节点，并赋值为num和pos

4.查找head中报数为m的结点,输出该元素的pos值，并把其num值赋给m,删除head中的上述元素所在节点，并释放掉

**基本操作的实现：**（对各基本操作实现的描述）

#include <iostream>

#include <cstring>

#include <stdlib.h>

using namespace std;

struct pos

{

int num,pos;

struct pos\* next;

};

struct pos\* head,\* tail;

int main()

{

int m,n;

cin>>m>>n;

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

{

int t;

cin>>t;

struct pos\* now=(struct pos\*)malloc(sizeof(struct pos));

now->num=t;

now->pos=i;

if(i==1)

{

head=now;

tail=now;

tail->next=head;

continue;

}

tail->next=now;

tail=now;

tail->next=head;

}

int ans=1;

int k=n;

//int b=1;

struct pos\* now=head;

struct pos\* pri=tail;

while(k!=1)

{

if(ans==m)

{

pri->next=now->next;

cout<<now->pos<<' ';

m=now->num;

ans=1;

free(now);

now=pri->next;

k--;

continue;

}

ans++;

pri=now;

now=now->next;

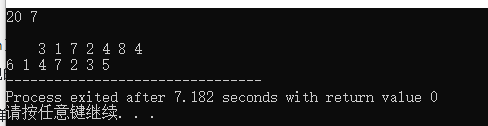
}

cout<<now->pos;

return 0;

}

**测试结果：**（输入的测试数据及运行结果、正确性、在线测试情况）



**调试分析：**（包括调试过程中对原设计的修改，以及遇到的问题和解决的方法）

1.自定义类型不要和STL重复。

2、窗口问题

**问题描述**：

在某图形操作系统中，有N个窗口，每个窗口都是一个两边与坐标轴分别平行的矩形区域。窗口的边界上的点也属于该窗口。窗口之间有层次的区别，在多于一个窗口重叠的区域里，只会显示位于顶层的窗口里的内容。

当你点击屏幕上一个点的时候，你就选择了处于被点击位置的最顶层窗口，并且这个窗口就会被移到所有窗口的最顶层，而剩余的窗口的层次顺序不变。如果你点击的位置不属于任何窗口，则系统会忽略你这次点击。

现在我们希望你写一个程序模拟点击窗口的过程。

**需求分析：**（包括对问题的理解，解决问题的策略、方法描述）

题意：若干窗口，满足条件的窗口放到第一位；

题解：运用单向链表，倒序输入，正序遍历，满足条件的节点断开并放在第一位。

**系统设计：**（包括数据结构定义、抽象出基本操作描述、主程序模块处理过程描述）

数据结构定义：

struct window

{

int x1,x2,y1,y2,num;

struct window\* next;

struct window\* prior;

};

抽象出基本操作描述

1.初始化链表

初始条件：p不存在

操作结果：创建一个链表头p

2.链表赋值

初始条件：p存在

操作结果：随着数据输入创建新的结构体并赋值，然后与先前链表链接

3.查找目标节点

初始条件：L存在且非空

操作结果：查找n个结点的p中满足条件的节点（p->x1<=x&&p->x2>=x&&p->y1<=y&&p->y2>=y）

4.断开并将目标节点放在链表头结点next位置

初始条件：p存在且非空

操作结果：将目标节点断开并放在头节点next位置

5.释放内存

主程序模块处理过程描述

1.创建线性链表p。

1. 输入m,n

3.生成n个结点，输入这n个数据元素的坐标值 ， 并定义 num 值

4.查找L中满足条件的节点的结点 （p->x1<=x&&p->x2>=x&&p->y1<=y&&p->y2>=y）

5. 断开并将目标节点放在链表头结点next位置

6．根据find函数的返回值确定输出为框体，还是“IGNORED”

**基本操作的实现：**（对各基本操作实现的描述）

#include <iostream>

#include <algorithm>

#include <cstring>

using namespace std;

struct window

{

int x1,x2,y1,y2,num;

struct window\* next;

struct window\* prior;

};

struct window\* now;

int find(int x,int y)

{

struct window\* p;

p=now;

while(p!=NULL)

{

if(p->x1<=x&&p->x2>=x&&p->y1<=y&&p->y2>=y)

{

if(p->prior==NULL)

{

p->next->prior=p->prior;

p->prior=now;

p->next=now->next;

now->next=p;

now=p;

return p->num;

}

if(p->next==NULL)

{

return p->num;

}

p->prior->next=p->next;

p->next->prior=p->prior;

p->next=now->next;

p->prior=now;

now->next=p;

now=p;

return p->num;

}

p=p->prior;

}

if(p==NULL)

{

return 0;

}

}

int main()

{

int m,n;

cin>>m>>n;

for(int i=1;i<=m;i++)

{

int x1,x2,y1,y2;

cin>>x1>>y1>>x2>>y2;

struct window\* body = (struct window\*)malloc(sizeof(struct window));

if(i==1)

{

now=body;

body->prior=NULL;

body->num=i;

body->next=NULL;

body->x1=x1;

body->x2=x2;

body->y1=y1;

body->y2=y2;

continue;

}

body->prior=now;

now->next=body;

body->num=i;

body->next=NULL;

body->x1=x1;

body->x2=x2;

body->y1=y1;

body->y2=y2;

now=body;

}

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

{

int x,y,flag;

cin>>x>>y;

flag=find(x,y);

if(!flag)

{

cout<<"IGNORED"<<'\n';

}

else

{

cout<<flag<<'\n';

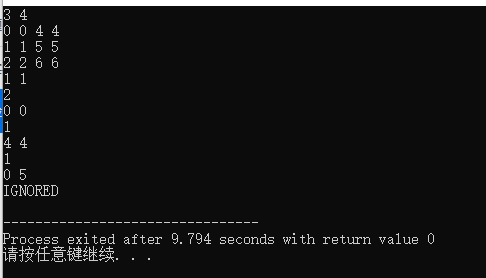
}

}

return 0;

}

**测试结果：**（输入的测试数据及运行结果、正确性、在线测试情况）



**调试分析：**（包括调试过程中对原设计的修改，以及遇到的问题和解决的方法）

1.建立为双向链表在断开指针指向时，要确定所有指针全部被断开，而不是部分断开，保持数据结构干净