# 实习报告

题目：用顺序表解决约瑟夫问题

班级：电信1705 姓名：刘羿 学号：U201713371完成日期：2018.4.8

## 一：需求分析

1. 约瑟夫问题的一种描述是：编号为1,2,.....,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的上限值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。
2. 本演示程序中，程序从用户获取总的人数n，每个人的密码，第一次报数的时候的m的初始值。输出出局的人的编号序列。
3. 程序执行的命令包括：

1)构造线性表; 2)依次输出出列者的序号 3)结束

4.测试数据

例1： 输入n=7，密码=[3,1,7,2,4,8,4],m=6.输出6 1 4 7 2 3 5

例2： 输入n=6，密码=[5,6,4,2,6,2],m=2.输出2 3 1 5 4 6

## 二：概要设计

1. 为实现上述程序功能，以顺序表表示集合，以数组实现顺序表

2.循环输出出列者的编号，已经出列的人密码定义为-1

3.全部人出列后，循环结束

## 三：详细设计

1. 元素类型

线性表->数组

1. 接受每个人的密码

{

printf("please enter the first key! ");

int key,i;

scanf("%d",&key);

int ring[N]; //定义线性表

for (i=0;i<N;++i)

{

printf("please enter the %d people's password! ",i+1);

scanf("%d",&ring[i]);

}

}

1. 循环输出出列者的编号

int m=0,k=0,count=1; //定义已经出列的人数，下一个报数者的编号，下一人的报数

while(m<N)

{

while(1)

{

k=k%N;

if(ring[k]>0) //找到未出列的人

{

if(count == key)

{

printf("%d ",k+1); //输出出列者的编号

key = ring[k];

ring[k] = -1;

count=1;

++k;

break;

}

else

{

count++;

k++;

}

}

else

++k;

}

m++;

}

## 四：调试分析

1. 刚开始打算以递归的形式实现，后发现需要传输的参数过多
2. 由于程序模块较少，并没有做模块划分，所以本程序的模块划分不够合理
3. 算法的时空分析
4. 1)空间复杂度：一个长度为N的数组
5. 2)时间复杂度：O(n^2)

## 五：用户手册

1. #define N 7 修改n的值需要在程序首修改N的值
2. 程序运行后首先输入m的初始值，接下来依次输入每个人的密码
3. 程序最后会输出出局的人的编号序列。

## 六：测试结果

## result

## 七：附录

1.源文件文件名清单：

Joseph\_ring

Main.c

2.源文件储存网址<https://github.com/liuyi12138/DataStructure>

1. 其他问题的代码及运行结果

1）求链表的交集病机

#include <stdio.h>

#include <stdlib.h>

typedef struct Node{

int data;

struct Node \*next;

}LinkedNode,\*LinkedList;

void CreateLinkedList(LinkedList \*L, int n);

void GetIntersection(LinkedList \*L1,LinkedList \*L2,LinkedList \*L3);

void GetUnion(LinkedList \*L1,LinkedList \*L2,LinkedList \*L4);

void FreeLink(LinkedList \*L);

int main()

{

int n1,n2;

printf("Now,we will create LinkedList A! please enter the number of LinkedNode you want in A ");

scanf("%d",&n1); //输入链表A中节点的个数

LinkedList \*Ahead = NULL;

printf("please enter the numbers From small to large!\n");

CreateLinkedList(&Ahead, n1); //创建链表A

printf("Now,we will create LinkedList B! please enter the number of LinkedNode you want in B ");

scanf("%d",&n2); //输入链表B中节点的个数

LinkedList \*Bhead = NULL;

printf("please enter the numbers From small to large!\n");

CreateLinkedList(&Bhead, n2); //创建链表B

LinkedList \*L3 = NULL; //定义新链表L3为AB两链表的交集

LinkedList \*L4 = NULL; //定义新链表L4为AB两链表的并集

printf("A and B have been created now,so we will get the intersection and union of them \n");

GetIntersection(&Ahead,&Bhead,&L3); //创建L3并输出其中的元素

GetUnion(&Ahead,&Bhead,&L4); //创建L4并输出其中的元素

FreeLink(&Ahead);

FreeLink(&Bhead);

FreeLink(&L3);

FreeLink(&L4);

return 1;

}

void CreateLinkedList(LinkedList \*L, int n) //创建新链表

{

LinkedNode \*now,\*p;

int i;

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

{

printf("please enter the %d data ",i);

now=(LinkedNode\*)malloc(sizeof(LinkedNode));

now->next=NULL;

scanf("%d",&now->data);

if((\*L)==NULL)

(\*L)=now;

else

p->next=now;

now->next=NULL;

p=now;

}

return ;

}

void GetIntersection(LinkedList \*L1,LinkedList \*L2,LinkedList \*L3) //得到两链表的交集并将数据存入新链表中

{

LinkedNode \*p = (\*L1);

LinkedNode \*q = (\*L2);

LinkedNode \*now1,\*k;

while((p!=NULL)&&(q!=NULL))

{

if(p->data==q->data)

{

now1=(LinkedNode\*)malloc(sizeof(LinkedNode));

now1->data=p->data;

now1->next=NULL;

if((\*L3)==NULL)

{

(\*L3)=now1;

k=(\*L3);

}

else

k->next=now1;

k=now1;

p=p->next;

q=q->next;

}

else

{

if(p->data>q->data)

q=q->next;

else

p=p->next;

}

}

LinkedNode \*k1 = (\*L3);

printf("the intersection of A B is : ");

while(k1!=NULL)

{

printf("%d ",k1->data);

k1=k1->next;

}

printf("\n");

return ;

}

void GetUnion(LinkedList \*L1,LinkedList \*L2,LinkedList \*L4) //得到两链表的并集并将数据存入新链表中

{

LinkedNode \*p = (\*L1);

LinkedNode \*q = (\*L2);

LinkedNode \*now2,\*k,\*now3;

now2=(LinkedNode\*)malloc(sizeof(LinkedNode));

if(p->data>q->data)

{

now2->data=q->data;

q=q->next;

}

else

{

if(p->data<q->data)

{

now2->data=p->data;

p=p->next;

}

else

{

now2->data=p->data;

p=p->next;

q=q->next;

}

}

now2->next=NULL;

(\*L4)=now2;

k=now2;

while((p!=NULL)&&(q!=NULL))

{

now3=(LinkedNode\*)malloc(sizeof(LinkedNode));

if(p->data>q->data)

{

now3->data=q->data;

k->next=now3;

k=now3;

q=q->next;

}

else

{

if(p->data<q->data)

{

now3->data=p->data;

p=p->next;

}

else

{

now3->data=p->data;

p=p->next;

q=q->next;

}

}

k->next=now3;

k=now3;

}

if(q!=NULL)

k->next=q;

if(p!=NULL)

k->next=p;

LinkedNode \*k2 = (\*L4);

printf("the union of A B is : ");

while(k2!=NULL)

{

printf("%d ",k2->data);

k2=k2->next;

}

printf("\n");

return ;

}

void FreeLink(LinkedList \*L)

{

LinkedNode \*k3 = (\*L);

while((\*L)!=NULL)

{

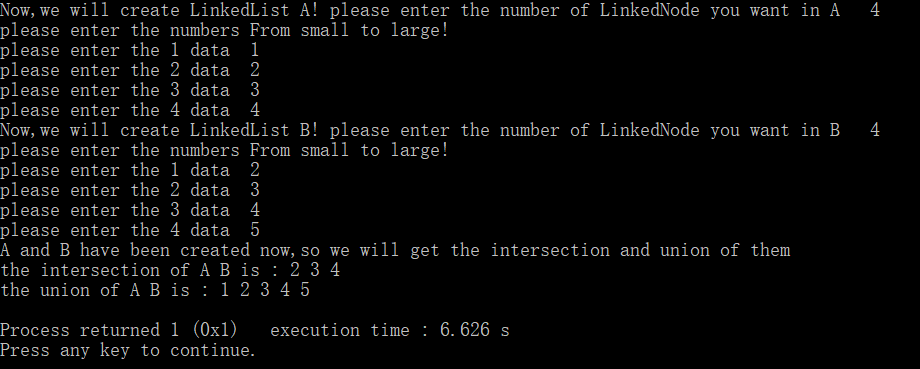
LinkedNode \*k3 = (\*L);

(\*L)=(\*L)->next;

free(k3);

}

}



1. 用栈实现十进制数转二进制数

#include <stdio.h>

#include <stdlib.h>

typedef struct Node{

int data;

struct Node \*next;

}InnNode,\*Inn;

void CreatInn(Inn \*base,Inn \*top,int n); //构建一个栈

void PullInn(Inn \*base,Inn \*top,int n); //输出栈中的数据

void FreeInn(Inn \*base,Inn \*top); //释放栈的空间

int main()

{

int n;

printf("Please enter a number! ");

scanf("%d",&n);

Inn \*top = NULL;

Inn \*base = NULL;

CreatInn(&base,&top,n);

PullInn(&base,&top,n);

FreeInn(&base,&top);

return 1;

}

void CreatInn(Inn \*base,Inn \*top,int n) //通过构建一种反向链表来实现栈

{

InnNode \*now;

now=(InnNode\*)malloc(sizeof(InnNode)); //为新的节点申请空间

now->data=n%2; //为新节点赋值

n=n/2;

(\*base)=now;

(\*top)=now;

while(n/2)

{

now=(InnNode\*)malloc(sizeof(InnNode));

now->data=(n%2);

n=n/2;

now->next=(\*top);

(\*top)=now;

}

if(n==1) //当n==1时，将最后一个数据压入栈中

{

now=(InnNode\*)malloc(sizeof(InnNode));

now->data=1;

now->next=(\*top);

(\*top)=now;

}

}

void PullInn(Inn \*base,Inn \*top,int n)

{

InnNode \*p=(\*top);

while(p!=(\*base)->next)

{

printf("%d",p->data); //依次输出栈中数据

p=p->next;

}

}

void FreeInn(Inn \*base,Inn \*top)

{

while((\*top)!=(\*base)->next)

{

InnNode \*p=(\*top);

(\*top)=(\*top)->next;

free(p);

}

}

