#### 1. 顺序表基本操作——学生成绩信息管理

【问题描述】

定义一个包含学生成绩（学号，姓名，成绩）的顺序表，读入相应的成绩信息来完成学生成绩表的创建。然后，（1）查找统计不及格的学生人数，（2）删除并以学号升序方式逐行输出不及格的学生信息。  
【输入形式】

输入n+1行，其中第1行是学生人数，后n行是n本图书的信息（学号，姓名，成绩），每条信息占一行，学号、姓名、成绩使用空格分隔，成绩之后没有空格。

【输出形式】

输出数据有m+1行，其中第1行是不及格学生人数，后m行是不及格学生信息，每条信息占一行，学号、姓名、成绩使用空格分隔，成绩之后没有空格。

【样例输入】

3

1001 zhangsan 34

1002 wangwu 66

1003 lisi 56  
【样例输出】

2

1001 zhangsan 34

1003 lisi 56

【样例说明】

如果不存在不及格同学，则输出提示信息“not found”

#### 2. 顺序表公共元素查找

【问题描述】给定两个顺序表，编写算法找出两表的公共结点。

【输入形式】

四行：

第一行：一个数字（第一个顺序表中的元素个数）

第二行：第一个顺序表中各个结点的值

第三行：一个数字（第二个顺序表中的元素个数）

第四行：第二个顺序表中各个结点的值

【输出形式】

两行：

第一行：两个单链表的公共元素个数

第二行：依次打印输出各个公共元素

【样例输入】

6

12 5 8 9 -23 16

4

16 21 9 3

【样例输出】

2

9 16

注：若两个链表无公共元素，则输出：

0

#include<stdio.h>

#include<malloc.h>

struct Lnode{

int data[20];

int last;

};//寤虹珛椤哄簭琛?

typedef struct Lnode \*list;

list makeempty(int n)

{

list ptrl;

ptrl=(list)malloc(sizeof(struct Lnode));

for(int i=0;i<n;i++)

{

scanf("%d",&ptrl->data[i]);

}

ptrl->last=n-1;

return ptrl;

}//寤虹珛绌鸿〃

int find(int n,list ptrl)

{

int i=0;

while(i<=ptrl->last&&ptrl->data[i]!=n)

{

i++;

}

if(i>ptrl->last)

{

return 0;

}

else

{

return n;

}

}//瀵绘壘鐩稿悓鍏冪礌

int main()

{

int n1,n2,i,n,s=0,a[20],j=0,t,q,p;

list p1,p2;

scanf("%d",&n1);

p1=makeempty(n1);

scanf("%d",&n2);

p2=makeempty(n2);

for(i=0;i<n1;i++)

{

n=p1->data[i];

t=find(n,p2);

if(t!=0)

{

a[j++]=t;

s+=1;

}

if(i==n1-1)

{

for(q=0;q<s;q++)

{

for(p=q+1;p<s;p++)

{

if(a[q]==a[p])

{

s-=1;

a[p]=0;

}

}

}

}

}//閫氳繃绗竴涓『搴忚〃鏌ヨ绗簩涓『搴忚〃涓浉鍚岀殑鍏冪礌骞跺瓨鍦╝[n]涓?

printf("%d\n",s);

if(s==0)

{

printf("没有公共元素");

}

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

{

if(a[i]!=0)

{printf("%d ",a[i]);}//杈撳嚭鐩稿悓鍏冪礌

}

return 0;

}

#### 1. 找出两个链表的公共结点

【问题描述】给定两个单链表，编写算法找出两个链表的公共结点。  
【输入形式】

四行：

第一行：一个数字（第一个单链表中的元素个数）

第二行：第一个单链表中各个结点的值

第三行：一个数字（第二个单链表中的元素个数）

第四行：第二个单链表中各个结点的值

【输出形式】

两行：

第一行：两个单链表的公共元素个数

第二行：依次打印输出各个公共元素  
【样例输入】

6

12 5 8 9 -23 16

4

16 21 9 3  
【样例输出】

2

9 16

注：若两个链表无公共元素，则输出：

0

没有公共元素

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct Node{

int data;

struct Node \*next;

}Node,\*List;

void InitList(List &L,int a[],int n){

L=(List)malloc(sizeof(Node));

Node \*s,\*r=L;

for(int i=0;i<n;i++){

s=(Node\*)malloc(sizeof(Node));

s->data=a[i];

r->next=s;

r=s;

}

r->next=NULL;

}

void dayin(List L1,List L2){

int data[100],i=0;

Node \*Head=L2;

L1=L1->next;

L2=L2->next;

while(L1){

L2=Head->next;

while(L2){

if(L1->data==L2->data){

data[i]=L1->data;

i++;

}

L2=L2->next;

}

L1=L1->next;

}

if(i==0){

printf("0\n");

printf("没有公共元素");}

else

printf("%d",i);

printf("\n");

for(int j=0;j<i;j++)

printf("%d ",data[j]);

}

int main()

{

int n1,n2,a[100],b[100],i,n;

List L1,L2;

scanf("%d",&n1);

for(i=0;i<n1;i++)

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

scanf("%d",&n2);

for(i=0;i<n2;i++)

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

InitList(L1,a,n1);

InitList(L2,b,n2);

dayin(L1,L2);

}

#### 2. 单链表重排列：S＝ (a1，an，a2，an-1，.…）

【问题描述】: 设线性表L = (a1 ,a2，a3，.…，an-1，an）采用带头结点的单链表保存，请设计一个空间复杂度为 0(1）且时间上尽可能高效的算法，重新排列 L 中的各结点，得到线性表S＝ (a1，an，a2，an-1，.…）。

【输入形式】：多组数据，每组第一行为n，表示链表的长度。接下来为链表中的元素。不超过5组数据。

【输出形式】：每组数据，输出重新排列后的序列。

【样例输入】

  10

  11 13 15 17 19 2 4 6 8 20

【样例输出】

  11 20 13 8 15 6 17 4 19 2  
【样例说明】  
【评分标准】

#include <stdio.h>

#include <stdlib.h>

#define M 100

typedef struct Lnode {

int data;

struct Lnode\* next;

}LNode;

LNode\* CreateLinkedList(int a[], int len) {

LNode\* head = NULL;

LNode\* tail = NULL;

int i;

for (i = 0; i < len; i++) {

LNode\* L = (LNode\*)malloc(sizeof(LNode));

L->data = a[i];

L->next = NULL;

if (head == NULL) {

head = L;

tail = L;

} else {

tail->next = L;

tail = L;

}

}

return head;

}

int isLinkedListEmpty(LNode\* head) {

return (head == NULL);

}

void PrintLinkedList(LNode\* head) {

LNode\* p = head;

while (p != NULL) {

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

p = p->next;

}

}

int GetLinkedListLength(LNode\* head) {

int len = 0;

LNode\* p = head;

while (p != NULL) {

len++;

p = p->next;

}

return len;

}

LNode\* GetLinkedListMiddleNode(LNode\* head) {

if (head == NULL) {

return NULL;

}

LNode\* fast = head;

LNode\* slow = head;

while (fast->next != NULL && fast->next->next != NULL) {

fast = fast->next->next;

slow = slow->next;

}

return slow;

}

LNode\* ReverseLinkedList(LNode\* head) {

LNode\* prev = NULL;

LNode\* curr = head;

LNode\* next = NULL;

while (curr != NULL) {

next = curr->next;

curr->next = prev;

prev = curr;

curr = next;

}

return prev;

}

LNode\* MergeLinkedLists(LNode\* head1,LNode\* head2) {

LNode\* dummyHead = (LNode\*)malloc(sizeof(LNode));

dummyHead->next = NULL;

LNode\* p = dummyHead;

LNode\* p1 = head1;

LNode\* p2 = head2;

while (p1 != NULL && p2 != NULL) {

p->next = p1;

p1 = p1->next;

p = p->next;

p->next = p2;

p2 = p2->next;

p = p->next;

}

if (p1 != NULL) {

p->next = p1;

}

if (p2 != NULL) {

p->next = p2;

}

LNode\* head = dummyHead->next;

free(dummyHead);

return head;

}

void ReorderLinkedList(LNode\* head) {

if (isLinkedListEmpty(head)) {

return;

}

LNode\* middleNode = GetLinkedListMiddleNode(head);

LNode\* head1 = head;

LNode\* head2 = middleNode->next;

middleNode->next = NULL;

head2 = ReverseLinkedList(head2);

head = MergeLinkedLists(head1, head2);

}

int main() {

int a[M];

int n,i;

scanf("%d",&n);

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

{

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

}

LNode\* head = CreateLinkedList(a,n);

ReorderLinkedList(head);

PrintLinkedList(head);

return 0;

}

#### 3. 删除单链表所有值为x的结点

【问题描述】在带头结点的单链表L中，删除所有值为x的结点，并释放其空间，假设值为x的结点不唯一，编写算法实现上述操作。  
【输入形式】

三行数据：

第一行：链表中元素个数n

第二行：n个数据，数据之间用空格分隔

第三行：待删除的数据  
【输出形式】

输出后的单链表  
【样例输入】

10

12 2 31 5 8 2 9 2 9 2

2  
【样例输出】

12 31 5 8 9 9

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

typedef struct Node{

int data;

struct Node \*next;

}Node,\*List;

void InitList(List &L,int n){

L=(List)malloc(sizeof(Node));

L->next=NULL;

Node \*r=L,\*head=L,\*p;

for(int i=0;i<n;i++){

p=(List)malloc(sizeof(Node));

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

r->next=p;

r=p;

}

r->next=NULL;

L=head->next;

}

void DeleteList(Node \*L,int \*x){

Node \*r=L,\*head=L->next;

while(r->next!=NULL){

Node \*p=r->next;

if(p->data==(\*x)){

r->next=p->next;

free(p);

}

else{

r=r->next;

}

}

L=head;

}

int main()

{

int n,i,x;

List L;

scanf("%d",&n);

InitList(L,n);

scanf("%d",&x);

DeleteList(L,&x);

while(L!=NULL){

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

L=L->next;

}

}

#### 1. 队列元素逆置

【问题描述】设Q是一个队列，S是一个空栈，实现将队列中的元素逆置的算法。（假设队列中的元素为字符型）

【输入形式】队列中的元素依次入队

【输出形式】依次输出队列中的元素

【样例输入】abcd

【样例输出】dcba

【样例说明】

【评分标准】

【提示信息】元素顺序入栈，然后依次出栈，出栈元素立即入队，最后打印出队序列。

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct{

char data[100];

int front,rear;

}SqQueue;

typedef struct{

char data[100];

int top;

}SqStack;

void InitStack(SqStack &S){

S.top=-1;

}

void InitQueue(SqQueue &Q){

Q.front=Q.rear=0;

}

void PushQueue(SqQueue &Q,char x){

Q.data[Q.rear]=x;

Q.rear++;

}

void cr(SqQueue &Q,SqStack &S){

S.top++;

S.data[S.top]=Q.data[Q.front];

Q.front++;

}

void print(SqStack S){

while(S.top>=0){

printf("%c",S.data[S.top]);

S.top--;

}

}

int main(){

SqQueue Q;

SqStack S;

int n,i;

char a[100];

gets(a);

n=strlen(a);

InitStack(S);

InitQueue(Q);

for(i=0;i<n;i++){

PushQueue(Q,a[i]);

}

for(i=0;i<n;i++){

cr(Q,S);

}

print(S);

}

#### 2. 字符中心对称判断——栈的应用

【问题描述】设单链表的表头指针为L，结点结构为data和next两个域构成，其中data域为字符型。设计算法判断链表的n个字符是否是中心对称的。例如xyx、xyyx都是中心对称的。  
【输入形式】字符序列  
【输出形式】yes 或者 no  
【样例输入】

样例1：abbba

样例2：abdcba  
【样例输出】

样例1输出：yes

样例2输出：no  
【样例说明】  
【评分标准】

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

typedef struct Node{

char data;

struct Node \*next;

}Node,\*List;

typedef struct{

char data[100];

int top;

}SqStack;

void push(List &L,SqStack &S)

{

S.top++;

S.data[S.top]=L->data;

L=L->next;

}

void InitList(List &L,int n,char a[]){

L=(List)malloc(sizeof(Node));

L->next=NULL;

Node \*r=L,\*p,\*head=L;

for(int i=0;i<n;i++){

p=(List)malloc(sizeof(Node));

p->data=a[i];

r->next=p;

r=p;}

r->next=NULL;

L=head->next;

}

void dc(List &L,int n){

int i;

char s[n/2];

SqStack S;

S.top=-1;

for(i=0;i<n/2;i++){

push(L,S);

}

if(n%2==1)

{ L=L->next;

while(L){

if(S.data[S.top]==L->data){

S.top--;

L=L->next;

i--;

}

else

goto leep;

}

}

else

{

while(L)

{

if(S.data[S.top]==L->data){

S.top--;

L=L->next;

i--;

}

else

goto leep;

}

}

leep:

if(i==0){

printf("yes");

}

else

printf("no");

}

int main(){

List L;

int i;

char a[100];

gets(a);

i=strlen(a);

InitList(L,i,a);

dc(L,i);

}

#### 3. 共享栈

【问题描述】：设有两个找 s1、s2都采用顺序找方式，并共享一个存储区[0, ...，maxsize-1]，为了 尽量利用空间，减少溢出的可能，可采用栈顶相向、迎面增长的存储方式。试设计 s1、s2有关入栈和出找的操作算法。

【输入形式】：多组数据，第一行为n，表示共享栈的容量，第二行为栈中序列的元素，#表示空闲位置。第三行至第n行为顺序执行的进栈操作（栈号 进栈元素）,栈号为0为进左端栈，栈号为1为进右端栈。

【输出形式】：若栈未满，则打印信息有一行为数据序列，#表示空闲位置。如栈已满则打印信息有两行，第一行为数据序列，第二行为提示信息“栈满”。

【样例说明】

|  |  |
| --- | --- |
| 样例输入 | 样例输出 |
| 9    11 13 15 # # # 6 8 20    0 16    1  4 | 11 13 15 16 # 4 6 8 20 |
| 7    11 13 15 # 6 8 20    0 16    1  4 | 11 13 15 16 6 8 20  满栈 |
|  |  |

【评分标准】

#include<stdio.h>

#include<string.h>

int main(){

//栈容量

int n;

scanf("%d",&n);

//栈存储结构：二维字符数组，

char da[n][4];

//栈顶指针

int top1=-1,top2=n;

int i;

/\*\*

栈的数据输入,一行放栈的一个数据值，例如

11

12

13

#

#

16

18

\*/

for(i=0;i<n;i++){

scanf("%s",da[i]);

}

//初始化top1

for(i=0;i<n;i++){

if(da[i][0]=='#'){

top1=i-1;

break;

}

}

//初始化top2

for(;i<n;i++){

if(da[i][0]!='#'){

top2=i;

break;

}

}

//EOF用法介绍：https://blog.csdn.net/qq\_44657062/article/details/105428393

int tag;

char x[4];

//循环：多行输入入栈信息，如：0 16

while(scanf("%d",&tag)!=EOF){

scanf("%s",x);

//共享栈进栈，非满即根据栈号tag进栈

if(da[top1+1][0]=='#'||da[top2-1][0]=='#'){

if(tag==0){

//字符串的互相赋值，不能使用赋值号，只能是strcpy，字符串拷贝函数

strcpy(da[++top1],x);

}else{

strcpy(da[--top2],x);

}

}

}

//在同一行内打印栈内数据，中间使用空格分隔

for(i=0;i<n-1;i++){

printf("%s ",da[i]);

}

//尾数字没有空格 ，因此单独打印

printf("%s",da[i]);

//换行，打印满栈提示信息

if(top1+1==top2){

printf("\n满栈");

}

}

#### 1. 二叉树的建立与遍历

【问题描述】按先序次序建立一棵二叉树，输入时，以井号表示空指针。对其进行后序遍历，并统计叶子结点的个数。  
【输入形式】二叉树的先序输入序列  
【输出形式】在屏幕上输出后序遍历的序列，并换行打印叶子的个数。  
【样例输入】

ABC##DE#G##F###  
【样例输出】

CGEFDBA

3  
【样例说明】  
【评分标准】

#include<stdio.h>

#include<stdlib.h>

typedef struct Node{

char data;

struct Node \*lchild,\*rchild;

}\*Tree,Node;

Tree build(){

char ch;

scanf("%c",&ch);

if(ch=='#')

return NULL;

Tree T=(Tree)malloc(sizeof(Node));

T->data=ch;

T->lchild=build();

T->rchild=build();

return T;

}

void order(Tree &T){

if(T==NULL)

return ;

order(T->lchild);

order(T->rchild);

printf("%c",T->data);

}

int count(Tree &T){

if(T==NULL)

return 0;

if(T->lchild==NULL&&T->rchild==NULL)

return 1;

else

return count(T->lchild)+count(T->rchild);

}

int main(){

Tree T=build();

order(T);

printf("\n");

printf("%d",count(T));

}

#include<stdio.h>

#include<ctype.h>

#include<string.h>

#include<stdlib.h>

typedef struct{

char vexs[100];

int edges[100][100];

int n, e;

}Graph;

int visited[100];

void initGraph(Graph \*g){

int i, j;

for(i=0; i<g->n; i++){

for(j=0;j<g->n;j++){

g->edges[i][j] = 0;

}

}

}

int findIndex(Graph \*g, char name){

int i;

for(i=0;i<g->n;i++){

if(g->vexs[i]==name){

return i;

}

}

return -1;

}

void insertEdge(Graph \*g, int i, int j, int w){

g->edges[i][j] = w;

g->edges[j][i] = w;

}

void output(Graph \*g){

int i, j;

for(i=0;i<g->n;i++){

for(j=0;j<g->n;j++){

printf(" %d",g->edges[i][j]);

}

printf("\n");

}

}

void DFS(Graph \*g, int v){

int i;

printf(" %c",g->vexs[v]);

visited[v] = 1;

for(i=0;i<g->n;i++){

if(g->edges[v][i]!=0 && !visited[i]){

DFS(g, i);

}

}

}

void BFS(Graph \*g, int v){

int queue[100];

int front = 0, rear = 0, temp;

visited[v] = 1;

printf(" %c",g->vexs[v]);

queue[rear] = v;

rear++;

while(front!=rear){

temp = queue[front];

front++;

for(int i=0;i<g->n;i++){

if(g->edges[temp][i]!=0 &&!visited[i]){

visited[i] = 1;

printf(" %c",g->vexs[i]);

queue[rear] = i;

rear++;

}

}

}

}

int main(){

Graph g;

int i, m, n, w;

char start, end, input;

char temp;

scanf("%d %d", &g.n, &g.e);

getchar();

initGraph(&g);

for(i=0;i<g.n;i++){

scanf("%c", &g.vexs[i]);

getchar();

}

for(i=0;i<g.e;i++){

scanf("%c %c %d", &start, &end, &w);

getchar();

insertEdge(&g, findIndex(&g, start), findIndex(&g, end), w);

}

scanf("%c", &input);

getchar();

memset(visited, 0, sizeof(visited));

DFS(&g, findIndex(&g, input));

printf("\n");

memset(visited, 0, sizeof(visited));

BFS(&g, findIndex(&g, input));

printf("\n");

return 0;

}