**数据结构习题**

**学号： 姓名：**

**1、**2-18（第二版）

#include <stdio.h>

#include <stdlib.h>

#define MAXSIZE 1024 // 顺序表最大长度设置为1024

typedef int Datatype; // 线性表数据类型设置为 int

typedef struct{ // 创建顺序表结构体

Datatype data[MAXSIZE];

int last;

}SequenList;

SequenList \*L;

SequenList \*Create() // 创建顺序表L

{

int n, i = 0;

L = (SequenList \*)malloc(sizeof(SequenList)); //分配顺序表空间

L -> last = -1;

printf("请输入数据，输入0时停止输入\n");

while(scanf("%d", &n) && n != 0)

{

L -> data[i ++ ] = n;

L -> last ++ ;

}

return L;

}

void Output(SequenList \*L) //输出顺序表元素

{

printf("\n顺序表中的元素为：");

for(int i = 0; i <= L -> last; i ++ )

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

printf("\n");

}

void Move(int k) // 移动顺序表元素

{

int tmp; // 临时变量，进行移动是储存顺序表尾元素

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

{

// 将除了顺序表尾的元素全部往右移动一位

tmp = L -> data[L -> last];

for(int j = L -> last; j > 0; j -- )

L -> data[j] = L -> data[j - 1];

L -> data[0] = tmp; // 将顺序表尾元素置于表首

}

}

int main()

{

int k;

L = Create();

printf("\n请输入右移的位数k:");

scanf("%d", &k);

Move(k);

Output(L);

return 0;

}

**2、**2-22（第二版）

**（一）、顺序表**

// 顺序表倒转

#include <stdio.h>

#include <stdlib.h>

#define MAXSIZE 1024 // 顺序表最大长度设置为1024

typedef int Datatype; // 线性表数据类型设置为 int

typedef struct{ // 创建顺序表结构体

Datatype data[MAXSIZE];

int last;

}SequenList;

SequenList \*L;

SequenList \*Create() // 创建顺序表L

{

int n, i = 0;

L = (SequenList \*)malloc(sizeof(SequenList)); //分配顺序表空间

L -> last = -1;

printf("请输入数据，输入0时停止输入\n");

while(scanf("%d", &n) && n != 0)

{

L -> data[i ++ ] = n;

L -> last ++ ;

}

return L;

}

void Output(SequenList \*L) //输出顺序表元素

{

printf("\n顺序表中的元素为：");

for(int i = 0; i <= L -> last; i ++ )

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

printf("\n");

}

void Reverse() // 倒转顺序表元素

{

int tmp;

for(int i = 0; i <= L -> last / 2; i ++ ) // 使顺序表头元素与表尾元素互换，对称位置互换，以此类推

{

tmp = L -> data[i];

L -> data[i] = L -> data[L -> last - i];

L -> data[L -> last - i] = tmp;

}

}

int main()

{

L = Create();

Reverse();

Output(L);

return 0;

}

**（二）、单链表**

// 链表倒转

#include <stdio.h>

#include <stdlib.h>

typedef struct node{ // 链表数组，其中有data元素

int data;

struct node \*next;

}LinkList;

LinkList \*Create() // 创建链表

{

int n;

LinkList \*head, \*s, \*r;

head = NULL; // 链表初值为空

r = NULL; // 尾指针初值为空

printf("请输入数据，输入0时停止输入\n");

while(scanf("%d", &n) && n != 0)

{

s = (LinkList\*)malloc(sizeof(LinkList)); //分配空间

s -> data = n;

if(head == NULL)

head = s; // 新结点s插入空表

else

r -> next = s; //非空表，新结点s插入到链表尾

r = s; // 尾指针指向新表尾

}

if(r != NULL) //非空表，将尾结点的指针指向空

r -> next = NULL;

return head; // 返回指针头

}

void Output(LinkList\* head) // 输出链表元素

{

printf("\n链表中的元素为：");

for(LinkList\* i = head; i != NULL; i = i -> next)

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

printf("\n");

}

LinkList \*Reverse(LinkList \*head) // 倒转链表元素，采用倒转指向的方法

{

if(head == NULL || head -> next == NULL) // 当链表为空或只有一个元素，返回该链表

return head;

LinkList \*BePointed = head, \*Point = head -> next, \*tmp = Point -> next; // 定义三个结构体变量，一个为被指向结点，一个为指向结点，一个为原链表中指向结点的下一个结点

while(tmp != NULL) // 当指向结点的下一个结点不为空，即指向结点不是链表尾时

{

Point -> next = BePointed; // 调转原链表的指向关系

BePointed = Point;

Point = tmp;

tmp = tmp -> next; // 更新 BePointed， Point， tmp

}

Point -> next = BePointed; // 将新链表头指向新链表第二个元素

head -> next = NULL; // 原链表头，即新链表尾，指向空

return Point; // 返回新链表头

}

int main()

{

LinkList \*head;

head = Create();

head = Reverse(head);

Output(head);

return 0;

}

**3、**2-28（第二版）

#include <stdio.h>

#include <stdlib.h>

typedef struct node{ // 链表数组，其中有data元素

int data;

struct node \*next;

}LinkList;

LinkList \*Create() // 创建链表

{

int n;

LinkList \*head, \*s, \*r;

head = NULL; // 链表初值为空

r = NULL; // 尾指针初值为空

printf("请输入数据，输入0时停止输入\n");

while(scanf("%d", &n) && n != 0)

{

s = (LinkList\*)malloc(sizeof(LinkList)); //分配空间

s -> data = n;

if(head == NULL)

head = s; // 新结点s插入空表

else

r -> next = s; //非空表，新结点s插入到链表尾

r = s; // 尾指针指向新表尾

}

if(r != NULL) //非空表，将尾结点的指针指向空

r -> next = NULL;

return head; // 返回指针头

}

void Output(LinkList\* head) // 输出链表元素

{

printf("\n链表中的元素为：");

for(LinkList\* i = head; i != NULL; i = i -> next)

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

printf("\n");

}

LinkList \*Reverse(LinkList \*head) // 倒转链表元素，采用倒转指向的方法

{

if(head == NULL || head -> next == NULL) // 当链表为空或只有一个元素，返回该链表

return head;

LinkList \*BePointed = head, \*Point = head -> next, \*tmp = Point -> next; // 定义三个结构体变量，一个为被指向结点，一个为指向结点，一个为原链表中指向结点的下一个结点

while(tmp != NULL) // 当指向结点的下一个结点不为空，即指向结点不是链表尾时

{

Point -> next = BePointed; // 调转原链表的指向关系

BePointed = Point;

Point = tmp;

tmp = tmp -> next; // 更新 BePointed， Point， tmp

}

Point -> next = BePointed; // 将新链表头指向新链表第二个元素

head -> next = NULL; // 原链表头，即新链表尾，指向空

return Point; // 返回新链表头

}

LinkList \*Union(LinkList \*head1, LinkList \*head2)

{

LinkList \*head = NULL, \*r = NULL, \*s;

while(head1 != NULL)

{

if(head2 == NULL)

break;

if(head1 -> data <= head2 -> data)

{

s = (LinkList\*)malloc(sizeof(LinkList)); //分配空间

s -> data = head1 -> data;

if(head == NULL)

head = s; // 新结点s插入空表

else

r -> next = s; //非空表，新结点s插入到链表尾

r = s; // 尾指针指向新表尾

if(head1 -> data == head2 -> data)

head2 = head2 -> next;

head1 = head1 -> next;

}

else

{

s = (LinkList\*)malloc(sizeof(LinkList)); //分配空间

s -> data = head2 -> data;

if(head == NULL)

head = s; // 新结点s插入空表

else

r -> next = s; //非空表，新结点s插入到链表尾

r = s; // 尾指针指向新表尾

head2 = head2 -> next;

}

}

if(head1 == NULL)

while(head2 != NULL)

{

s = (LinkList\*)malloc(sizeof(LinkList)); //分配空间

s -> data = head2 -> data;

if(head == NULL)

head = s; // 新结点s插入空表

else

r -> next = s; //非空表，新结点s插入到链表尾

r = s; // 尾指针指向新表尾

head2 = head2 -> next;

}

if(head2 == NULL)

while(head1 != NULL)

{

s = (LinkList\*)malloc(sizeof(LinkList)); //分配空间

s -> data = head1 -> data;

if(head == NULL)

head = s; // 新结点s插入空表

else

r -> next = s; //非空表，新结点s插入到链表尾

r = s; // 尾指针指向新表尾

head1 = head1 -> next;

}

if(r != NULL) //非空表，将尾结点的指针指向空

r -> next = NULL;

return Reverse(head);

}

int main()

{

LinkList \*head1 = Create();

LinkList \*head2 = Create();

LinkList \*head = Union(head1, head2);

Output(head);

return 0;

}

**4、**2-30（第二版）

#include <stdio.h>

#include <stdlib.h>

typedef struct node

{

char data;

struct node \*next;

}LinkList;

LinkList \*ALPH = NULL, \*NUM = NULL, \*OTHERS = NULL; // 定义三个循环链表头结点

int cntA = 0, cntN = 0, cntO = 0; // 定义三个计数点

LinkList \*Create() // 创建链表

{

char n;

LinkList \*head, \*s, \*r;

head = NULL; // 链表初值为空

r = NULL; // 尾指针初值为空

printf("请输入数据，输入0时停止输入\n");

while(scanf("%c", &n) && n != '0')

{

s = (LinkList\*)malloc(sizeof(LinkList)); //分配空间

s -> data = n;

if(head == NULL)

head = s; // 新结点s插入空表

else

r -> next = s; //非空表，新结点s插入到链表尾

r = s; // 尾指针指向新表尾

}

if(r != NULL) //非空表，将尾结点的指针指向空

r -> next = NULL;

return head; // 返回指针头

}

void ConverToCircleList(LinkList \*L) // 将单链表转化为循环链表

{

LinkList \*p = L;

for(; p -> next != NULL; p = p -> next);

p -> next = L;

}

void Add(LinkList \*L, LinkList \*p, int cnt) // 单链表添加元素

{

LinkList \*tmp = L;

for(int i = 0; i < cnt; i ++ ) // 采用计数的遍历形式

tmp = tmp -> next;

tmp -> next = p;

}

void classify(LinkList \*head) // 对给定链表进行分类

{

LinkList \*p = head;

while(p)

{

if(p -> data >= 'a' && p -> data <= 'z' || p -> data >= 'A' && p -> data <= 'Z') // 字母类

{

if(ALPH == NULL) // 为空时填充头结点

{

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

s -> data = p -> data;

s -> next = NULL;

ALPH = s;

}

else // 否则添加进入单链表，并对新增链表元素个数

{

Add(ALPH, p, cntA);

cntA ++ ;

}

}

else if(p -> data >= '0' && p -> data <= '9') // 数字类

{

if(NUM == NULL)

{

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

s -> data = p -> data;

s -> next = NULL;

NUM = s;

}

else

{

Add(NUM, p, cntN);

cntN ++ ;

}

}

else

{

if(OTHERS == NULL) // 其他类

{

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

s -> data = p -> data;

s -> next = NULL;

OTHERS = s;

}

else

{

Add(OTHERS, p, cntO);

cntO ++ ;

}

}

p = p -> next; // 更新结点

}

}

void EmptyRedunNodes(LinkList \*L, int cnt) // 将分好类后的链表最后一个元素的next变为NULL

{

LinkList \*tmp = L;

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

tmp = tmp -> next;

tmp -> next = NULL;

}

void Output(LinkList \*L) // 输出链表元素

{

printf("\n链表中的元素为：");

for(LinkList\* i = L; i != NULL; i = i -> next)

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

printf("\n");

}

int main()

{

LinkList \*head = Create();

classify(head);//, ALPH, NUM, OTHERS);

EmptyRedunNodes(ALPH, cntA);

EmptyRedunNodes(NUM, cntN);

EmptyRedunNodes(OTHERS, cntO);

Output(ALPH);

Output(NUM);

Output(OTHERS);

ConverToCircleList(ALPH);

ConverToCircleList(NUM);

ConverToCircleList(OTHERS);

return 0;

}