**数据结构实验报告**

**实验〇：顺序表与链表**

专 业： 网络工程

班 级： 1501

姓 名： 邓泽先

学 号： 150200325

指导教师： 王梅

日 期： 2016-9-26

**一、实验目的**

（1）掌握顺序表的存储结构形式及其描述和基本运算的实现。

（2）熟练掌握动态链表结构及有关算法的设计。

（3）掌握用链表表示特定形式的数据的方法，并能编写出有关运算的算法。

（4）理解单循环链表及双循环链表的特点。

**二、实验内容**

**1、顺序表的应用**

(1)．输入一组整型元素序列，建立顺序表。

(2)．实现该顺序表的遍历。

(3)．在该顺序表中顺序查找某一元素,查找成功返回1,否则返回0。

(4)．判断该顺序表中元素是否对称,对称返回1,否则返回0。

(5)．实现把该表中所有奇数排在偶数之前,即表的前面为奇数,后面为偶数。

(6)．输入整型元素序列利用有序表插入算法建立一个有序表。

(7)．利用算法6建立两个非递减有序表并把它们合并成一个非递减有序表。

(8)．编写一个主函数,调试上述算法。

**2、链表的应用**

(1)．键盘输入一组元素，建立一个无头结点的单向链表（无序）。

(2)．遍历（打印）单向链表。

(3)．把单向链表中元素逆置（不允许申请新的结点空间）。

(4)．在单向链表中删除所有的偶数元素结点。

(5)．对链表排序，排序后链表元素按照非递减方式排列（注意：排序时如果要交换两个结点的顺序，不得通过交换结点的内容，而需要使用改变指针的方式交换结点的位置。建议使用直接插入排序算法）。

(6)．利用算法5建立两个非递减有序单向链表，然后合并成一个非递减链表。

(7)．利用算法1建立的链表，删除链表中的重复元素。

(8)．利用算法1建立的链表，实现将其分解成两个链表，其中一个全部为奇数，另一个全部为偶数（尽量利用已知的存储空间）。

(9)．判断算法1和算法5生成单链表所表示的集合是否相等。

(10)．在主函数中设计一个简单的菜单，分别调试上述算法。

**三、需求分析**

1.输入的形式和输入值的范围

建立链表时输入的都是整数,输入0代表输入的结束，插入元素时需要输入插入的位置和元素的值，删除元素时输入删除元素的值。

2.输出形式

所有操作在出现错误时都会有提示，并且在任意一个操作结束后都会输出操作后的顺序表或链表。

3.程序所能达到的功能

完成顺序表和链表的生成，并且能够实现实验内容中所包含的功能

4.测试数据：见第八部分。

**四、概要设计**

(1) 本程序包含的函数

顺序表：主函数 main()

建立顺序表函数 createArray()

顺序表的遍历函数 printfAllOfArray(int \* a)

在顺序表中查找某元素函数 findInArray(int \* a, int num)

顺序表是否对称函数 isSymmetry(int \* a)

把该表中所有奇数排在偶数之前的函数 five(int \* a)

建立有序表函数 createOrderlyForm()

合并成递减有序表的函数 createBigOrderlyForm()

链表：

主函数 main()

建立链表 createList()

遍历链表 printfList(struct Num \* head)

逆置链表 convert(struct Num \* head)

删除偶数节点函数 deleteEvenNumber(struct Num \* head)

排序函数 sortList(struct Num \* head)

合并函数 createBigList()

递归函数 digui(struct Num \* p1, struct Num \* p2)

删除重复节点函数 deleteNode(struct Num \* head)

分离链表函数 separate(struct Num \* head, struct Num \* odd, struct Num \* even)

判断链表集合是否相等函数 isEqual(struct Num \* head1, struct Num \* head2)

(2)各函数的功能设计为：

顺序表：

int \* createArray() 建立顺序表,输入值为0结束.返回顺序表的地址

void printfAllOfArray(int \* a) 传递进去一个顺序表的地址，遍历该顺序表

int findInArray(int \* a, int num) 传递进去被检验的顺序表和需要检验的数字，看看该数字是否在改顺序表里.在里面返回1，不在里面返回0

int isSymmetry(int \* a) 传递进去一个顺序表的地址，检验该顺序表是否对称，对称返回1，不对称返回0

void five(int \* a) 传递进去一个顺序表的地址，把该表中所有奇数排在偶数之前。

int \* createOrderlyForm() 输入整型元素序列利用有序表插入算法建立一个有序表。返回顺序表的地址。

int \* createBigOrderlyForm()利用算法6建立两个非递减有序表并把它们合并成一个非递减有序表，返回合并后表的地址

main(void) 调试上述函数

链表：

struct Num \* createList() 键盘输入一组元素，建立一个无头结点的单向链表（无序），返回链表第一个节点的地址

void printfList(struct Num \* head) 接收一个链表的第一个节点的地址，遍历该链表

struct Num \* convert(struct Num \* head) 接收一个链表的第一个节点的地址，将该链表逆置；返回链表第一个节点的地址

void deleteEvenNumber(struct Num \* head) 接收一个链表的第一个节点的地址，删除所有的偶数元素结点。

struct Num \* sortList(struct Num \* head) 接收一个链表的第一个节点的地址，对链表排序，排序后链表元素按照非递减方式排列；返回链表第一个节点的地址

struct Num \* createBigList() 利用算法5建立两个非递减有序单向链表，然后合并成一个非递减链表，返回该链表第一个节点的地址（这里会用到下面那个递归函数）

struct Num \* digui(struct Num \* p1, struct Num \* p2) 递归函数，不再赘述

void deleteNode(struct Num \* head) 接收一个链表的第一个节点的地址，删除重复的元素

void separate(struct Num \* head, struct Num \* odd, struct Num \* even) 接收一个链表的第一个节点的地址，储存奇数和偶数链表的引用，将该链表分到奇数和偶数链表里

int isEqual(struct Num \* head1, struct Num \* head2) head1来获取算法1得到的表，head2获取算法5得到的表

main(void) 调试上述函数

**五、详细设计**

(1)链表中结点类型和指针类型:

struct Num

{

int num;

struct Num \* next;

};

(2)顺序表和单链表的基本操作

主要操作就是利用步骤四里的函数处理顺序表和链表，详情见步骤四

**六、调试分析**

调试过程比较麻烦

**七、使用说明**

顺序表运行后会有以下的菜单:

1.建立顺序表,输入值为0结束

2.实现顺序表的遍历

3.在该顺序表查找某一元素，找到返回1，否则0

4.判断该顺序表中元素是否对称,对称返回1,否则返回0

5.实现把该表中所有奇数排在偶数之前,即表的前面为奇数,后面为偶数

6.输入整型元素序列利用有序表插入算法建立一个有序表

7.利用算法6建立两个非递减有序表并把它们合并成一个非递减有序 表

输入0退出程序

请输入你的选择：

链表运行后会出现以下的菜单：

(1)．键盘输入一组元素，建立一个无头结点的单向链表（无序）

(2)．遍历（打印）单向链表

(3)．把单向链表中元素逆置

(4)．在单向链表中删除所有的偶数元素结点

(5)：对链表排序，排序后链表元素按照非递减方式排列

(6)．利用算法5建立两个非递减有序单向链表，然后合并成一个非递减链表

(7)．利用算法1建立的链表，删除链表中的重复元素

(8)．利用算法1建立的链表，实现将其分解成两个链表，其中一个全部为奇数，另一个全部为偶数

(9)．判断算法1和算法5生成单链表所表示的集合是否相等

输入0退出程序

请输入您的选择：

选择要进行的操作，如果出现了非法的操作，如没有建立链表就进行了操作，可能会有提示：该链表为空。

**八、测试结果**

顺序表：

1. 建立顺序表：

选择1，输入6 2 3 4 9 0，得到顺序表（6，2，3，4，9）

1. 遍历：

选择2，输出顺序表6 2 3 4 9

1. 在顺序表中查找某元素

选择3，输入查找6

得到结果：该元素在这个表里

1. 判断该顺序表是否对称

选择4，得到结果：这个表不对称

1. 把这个表的奇数排到偶数前面

选择5，得到结果3 9 6 2 4

1. 输入整型元素序列利用有序表插入算法建立一个有序表

选择6，输入6 3 9 7 0，得到结果3 6 7 9

1. 利用算法6建立两个非递减有序表并把它们合并成一个非递减有序表

选择7，分别输入9 8 7 0 和 6 5 4 0，得到结果4 5 6 7 8 9

链表：

1. 建立链表：

选择1，输入数据6 8 1 2 3 0，建立链表

1. 遍历链表

选择2，输入数据6 8 1 2 3

1. 将链表逆置

选择3，得到数据3 2 1 8 6

1. 删除链表中的偶数节点

选择4，得到数据3 1

1. 将链表按照非递减的顺序排序

选择5，得到数据1 3

1. 利用算法5建立两个链表，合并成一个

选择6，分别输入数据6 7 5 0和3 6 5 0，得到数据3 5 5 6 6 7

1. 删除算法1生成的链表中的重复元素

选择7，得到数据6 8 1 2 3 0

1. 将算法1生成的链表分成奇数和偶数两部分

选择8，得到数据1 3和数据6 8 2

1. 判断算法1和算法5生成单链表所表示的集合是否相等

选择9，输入3 6 8 5 0和3 6 9 5 0得到结果算法1和算法5生成单链表所表示的集合不相等的

再次选择9，输入6 8 5 11 0和11 6 5 8 0得到结果算法1和算法5生成单链表所表示的集合是相等的

**九、附录：部分代码**

**顺序表：**

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#define NumberOfArray 500

//这里是顺序表的地方

int getLength(int a[])

{

int i = 0;

int length = 0;

while(a[i] != 0)

{

i++;

}

length = i;

return length;

}

//1.建立顺序表,输入值为0结束

int \* createArray()

{

int a[NumberOfArray];

int i = 0;

do{

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

}while(a[i++] != 0);

return a;

}

//2.实现该顺序表的遍历

void printfAllOfArray(int \* a)

{

if(a[0] == 0)

{

printf("这是个空顺序表！！\n");

return;

}

int i = 0;

for(i = 0; i < getLength(a); i++)

{

printf("%d ", a[i]);

}

printf("\n");

}

//3.在该顺序表查找某一元素，找到返回1，否则0o

int findInArray(int \* a, int num)

{

if(a[0] == 0)

{

printf("这是个空顺序表！！\n");

return 0;

}

int i = 0;

for(i = 0; i < 500 && a[i] != 0; i++)

{

if(a[i] == num)

{

return 1;

}

}

return 0;

}

//4.判断该顺序表中元素是否对称,对称返回1,否则返回0

int isSymmetry(int \* a)

{

if(a[0] == 0)

{

printf("这是个空顺序表！！\n");

return 0;

}

int i = 0;

int length = getLength(a);

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

{

if(a[i] != a[length-i-1])

{

return 0;

}

}

return 1;

}

//5.实现把该表中所有奇数排在偶数之前,即表的前面为奇数,后面为偶数

void five(int \* a)

{

if(a[0] == 0)

{

printf("这是个空顺序表！！\n");

return;

}

int length = getLength(a);

int ji[NumberOfArray/2];

int ou[NumberOfArray/2];

int j = 0;

int k = 0;

int i = 0;

for(; i < length; i++)

{

if(a[i] % 2 == 1)

{

ji[j++] = a[i];

}

else

{

ou[k++] = a[i];

}

}

i--;

--j;

--k;

while(k > -1)

{

a[i--] = ou[k--];

}

while(j > -1)

{

a[i--] = ji[j--];

}

}

//6.输入整型元素序列利用有序表插入算法建立一个有序表

int \* createOrderlyForm()

{

int a[NumberOfArray]= {0};

int num;

int count = 0;

int flag;

do{

flag = 0;

int i;

scanf("%d", &num);

if(num != 0)

{

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

{

if(num < a[i])

{

flag = 1;

int j = 0;

for(j = 0; j <= count-i-1; j++)

{

a[count-j] = a[count-j-1];

}

a[i] = num;

break;

}

}

}

if(!flag)

{

a[count] = num;

}

count++;

}while(num != 0);

return a;

}

//7.利用算法6建立两个非递减有序表并把它们合并成一个非递减有序表

int \* createBigOrderlyForm()

{

int \* a = createOrderlyForm();

int c[2\*NumberOfArray] = {0};

int i = 0;

int j = 0;

while(a[i] != 0)

{

c[j++] = a[i++];

}

i = 0;

printf("请再输入值：\n");

a = createOrderlyForm();

while(a[i] != 0)

{

c[j++] = a[i++];

}

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

{

int num;

int k = 0;

for(k = i+1; k < j; k++)

{

if(c[i] > c[k])

{

num = c[i];

c[i] = c[k];

c[k] = num;

}

}

}

return c;

}

int main(void)

{

int num = -1;

int a = 0;

int \* p = &a;

do{

printf("1.建立顺序表,输入值为0结束\n2.实现顺序表的遍历\n3.在该顺序表查找某一元素，找到返回1，否则0\n4.判断该顺序表中元素是否对称,对称返回1,否则返回0\n5.实现把该表中所有奇数排在偶数之前,即表的前面为奇数,后面为偶数");

printf("\n6.输入整型元素序列利用有序表插入算法建立一个有序表\n7.利用算法6建立两个非递减有序表并把它们合并成一个非递减有序表\n输入0退出程序\n请输入你的选择：\n");

scanf("%d", &num);

switch(num)

{

case 1:

printf("请输入值：\n");

p = createArray();

printfAllOfArray(p);

break;

case 2:

printf("下面是遍历的值:\n");

printfAllOfArray(p);

break;

case 3:

printf("请输入要查找的元素\n");

scanf("%d", &num);

printf("该元素%s这个表里\n", findInArray(p, num)?"在":"不在");

break;

case 4:

printf("这个表%s对称\n", isSymmetry(p)?"":"不");

break;

case 5:

five(p);

printfAllOfArray(p);

break;

case 6:

printf("请输入值：\n");

p = createOrderlyForm();

printf("有序表为：\n");

printfAllOfArray(p);

break;

case 7:

printf("请输入值：\n");

p = createBigOrderlyForm();

printf("合成的有序表为：\n");

printfAllOfArray(p);

break;

case 0:

break;

default:

printf("输入的数字不正确！！");

p = createArray();

break;

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n");

}while(num != 0);

return 0;

}

**链表：**

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

struct Num

{

int num;

struct Num \* next;

};

//(1)．键盘输入一组元素，建立一个无头结点的单向链表（无序）

struct Num \* createList()

{

struct Num \* head = NULL;

struct Num \* p1, \* p2;

int n = 0;

p1 = p2 = (struct Num \*)malloc(sizeof(struct Num));

scanf("%d", &p1->num);

head = NULL;

while(p1->num != 0)

{

n++;

if(n == 1)

{

head = p1;

}

else

{

p2->next = p1;

}

p2 = p1;

p1 = (struct Num \*)malloc(sizeof(struct Num));

scanf("%d", &p1->num);

}

p2->next = NULL;

return head;

}

//(2)．遍历（打印）单向链表。

void printfList(struct Num \* head)

{

struct Num \* p;

p = head;

if(head == NULL)

{

printf("该链表为空！！\n");

}

else

{

while(p != NULL)

{

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

p = p->next;

}

printf("\n");

}

}

//(3)．把单向链表中元素逆置（不允许申请新的结点空间）。

struct Num \* convert(struct Num \* head)

{

struct Num \* p1, \* p2, \*p3;

int n = 0;

p1 = head;

if(head == NULL)

{

printf("这是个空链表！！\n");

return NULL;

}

p2 = p1->next;

p1->next = NULL;

while(p2 != NULL)

{

p3 = p2->next;

p2->next = p1;

p1 = p2;

p2 = p3;

}

return p1;

}

//(4)．在单向链表中删除所有的偶数元素结点。

void deleteEvenNumber(struct Num \* head)

{

struct Num \* p1, \* p2, \* p3;

int n = 0;

p1 = head;

if(head == NULL)

{

printf("这是一个空链表！！\n");

return;

}

do{

n++;

p2 = p1->next;

if(p1->num % 2 == 0)

{

if(n == 1)

{

free(p1);

}

else

{

free(p1);

p1 = p3;

p3->next = p2;

}

}

p3 = p1;

p1 = p2;

}while(p2 != NULL);

}

//(5)：对链表排序，排序后链表元素按照非递减方式排列（注意：排序时如果要交换两个结点的顺序，不得通过交换结点的内容，而需要使用改变指针的方式交换结点的位置。建议使用直接插入排序算法）。

struct Num \* sortList(struct Num \* head)

{

struct Num \* target, \* p1, \* p2;

struct Num \* q1, \* q2;//存放需要比较的节点和该节点的前一个节点

if(head == NULL)

{

printf("这是个空链表!!\n");

return NULL;

}

target = head->next;

p1 = head;

p2 = head->next->next;

while(target != NULL)

{

q1 = head;

q2 = head->next;

if(target->num < head->num)

{

target->next = q1;

p1->next = p2;

head = target;

target = p2;

if(p2 != NULL)

p2 = p2->next;

continue;

}

int flag = 0;//检验是否交换过

while(q2 != NULL)

{

if(target == q2)

break;

if(target->num < q2->num)

{

flag = 1;

p1->next = p2;

q1->next = target;

target->next = q2;

target = p2;

if(p2 != NULL)

p2 = p2->next;

break;

}

q1 = q2;

if(q2 != NULL)

q2 = q2->next;

}

if(flag)

continue;

p1 = p1->next;

if(p2 != NULL)

p2 = p2->next;

target = target->next;

}

return head;

}

//(6)．利用算法5建立两个非递减有序单向链表，然后合并成一个非递减链表。

struct Num \* digui(struct Num \* p1, struct Num \* p2)//不得已用的递归

{

if(p1->next == NULL)

p1->next = p2;

else

digui(p1->next, p2);

return p1;

}

struct Num \* createBigList()

{

struct Num \* p1, \* p2;

struct Num num;

printf("请输入第一组：\n");

p1 = createList();

num = \*p1;

p1 = sortList(p1);

printfList(p1);

printf("请输入第二组：\n");

p2 = createList();

p2 = sortList(p2);

printfList(p2);

p1 = digui(p1, p2);//这里用了各种循环方法都是失败，改成递归后就成功了

printfList(p1);

p1 = sortList(p1);

}

//(7)．利用算法1建立的链表，删除链表中的重复元素。

void deleteNode(struct Num \* head)

{

struct Num \* p1, \* p2, \*p3;

if(head == NULL)

{

printf("这是个空链表！！\n");

return;

}

p1 = p1 = head;

while(p1 != NULL)

{

p3 = p1;

p2 = p1->next;

while(p2 != NULL)

{

if(p1->num == p2->num)

{

p3->next = p2->next;

free(p2);

p2 = p3->next;

}

else

{

p3 = p2;

p2 = p2->next;

}

}

p1 = p1->next;

}

}

//(8)．利用算法1建立的链表，实现将其分解成两个链表，其中一个全部为奇数，另一个全部为偶数（尽量利用已知的存储空间）。

void separate(struct Num \* head, struct Num \* odd, struct Num \* even)

{

struct Num \* ji, \* ou;

struct Num \* p1;

p1 = head;

ji = ou = NULL;

if(p1 == NULL)

{

printf("这是个空链表！！\n");

return;

}

while(head != NULL)

{

if(head->num % 2)

{

head = head->next;

p1->next = NULL;

if(ji == NULL)

{

ji = p1;

odd = p1;

}

else

{

ji->next = p1;

ji = ji->next;

}

}

else

{

head = head->next;

p1->next = NULL;

if(ou == NULL)

{

ou = p1;

even = p1;

}

else

{

ou->next = p1;

ou = ou->next;

}

}

p1 = head;

}

printf("奇数表如下：\n");

printfList(odd);

printf("偶数表如下：\n");

printfList(even);

}

//(9)．判断算法1和算法5生成单链表所表示的集合是否相等。

int isEqual(struct Num \* head1, struct Num \* head2)//head1来获取算法1得到的表，head2获取算法5得到的表

{

struct Num \* p1, \* p2;

int flag = 0;

p1 = head1;

while(p1 != NULL)

{

p2 = head2;

flag = 0;

while(p2 != NULL)

{

if(p1->num != p2->num)

{

p2 = p2->next;

continue;

}

else

{

flag = 1;

break;

}

}

if(!flag)

{

return 0;

}

p1 = p1->next;

}

p2 = head2;

while(p2 != NULL)

{

p1 = head1;

flag = 0;

while(p1 != NULL)

{

if(p1->num != p2->num)

{

p1 = p1->next;

continue;

}

else

{

flag = 1;

break;

}

}

if(!flag)

{

return 0;

}

p2 = p2->next;

}

return 1;

}

//(10)．在主函数中设计一个简单的菜单，分别调试上述算法。

int main (void)

{

struct Num \* p1, \* odd, \* even;

p1 = odd = even = NULL;

int num = -1;

do{

printf("(1)．键盘输入一组元素，建立一个无头结点的单向链表（无序）\n");

printf("(2)．遍历（打印）单向链表\n");

printf("(3)．把单向链表中元素逆置\n");

printf("(4)．在单向链表中删除所有的偶数元素结点\n");

printf("(5)：对链表排序，排序后链表元素按照非递减方式排列\n");

printf("(6)．利用算法5建立两个非递减有序单向链表，然后合并成一个非递减链表\n");

printf("(7)．利用算法1建立的链表，删除链表中的重复元素\n");

printf("(8)．利用算法1建立的链表，实现将其分解成两个链表，其中一个全部为奇数，另一个全部为偶数\n");

printf("(9)．判断算法1和算法5生成单链表所表示的集合是否相等\n输入0退出程序\n");

printf("请输入您的选择：\n");

scanf("%d", &num);

switch(num)

{

case 1:

printf("请输入要键入的值：\n");

p1 = createList();

printf("这是您建立的链表：\n");

printfList(p1);

break;

case 2:

printf("这是您的链表：\n");

printfList(p1);

break;

case 3:

printf("逆置后的结果如下：\n");

p1 = convert(p1);

printfList(p1);

break;

case 4:

printf("删除后的结果如下：\n");

deleteEvenNumber(p1);

printfList(p1);

break;

case 5:

printf("排序后的结果如下：\n");

p1 = sortList(p1);

printfList(p1);

break;

case 6:

p1 = createBigList();

printf("所得结果如下：\n");

printfList(p1);

break;

case 7:

printf("所得结果如下：\n");

deleteNode(p1);

printfList(p1);

break;

case 8:

separate(p1, odd, even);

break;

case 9:

printf("请输入您要由算法1生成的链表的内容：\n");

struct Num \* p2, \* p3;

p2 = createList();

printf("请输入您要由算法5生成的链表的内容：\n");

p3 = createList();

p3 = sortList(p3);

printf("算法1和算法5生成单链表所表示的集合%s相等的", isEqual(p2, p3)?"是":"不");

break;

case 0:

break;

default:

printf("请输入正确的值！！\n\n\n");

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n");

}while(num != 0);

return 0;

}