**《数据结构》实验报告**

**姓名： 杨孟衡 学号： 8002118240**

**班级： 软件工程1809班 专业： 软件工程**

**报告日期： 2019 年 10 月 10 日**

**实验三 链表**

**一、问题描述与分析**

1. 设某线性数据元素类型为浮点型，以线性顺序表为存储结构。试编程实现：

1. 熟练掌握动态链表结构及有关算法的设计方法；
2. 理解带头结点的单链表的特点，掌握采用这种结构的算法设计；
3. 熟练掌握运用带头结点链表表示特定形式的数据的方法，并设计出有关算法。

2.分析

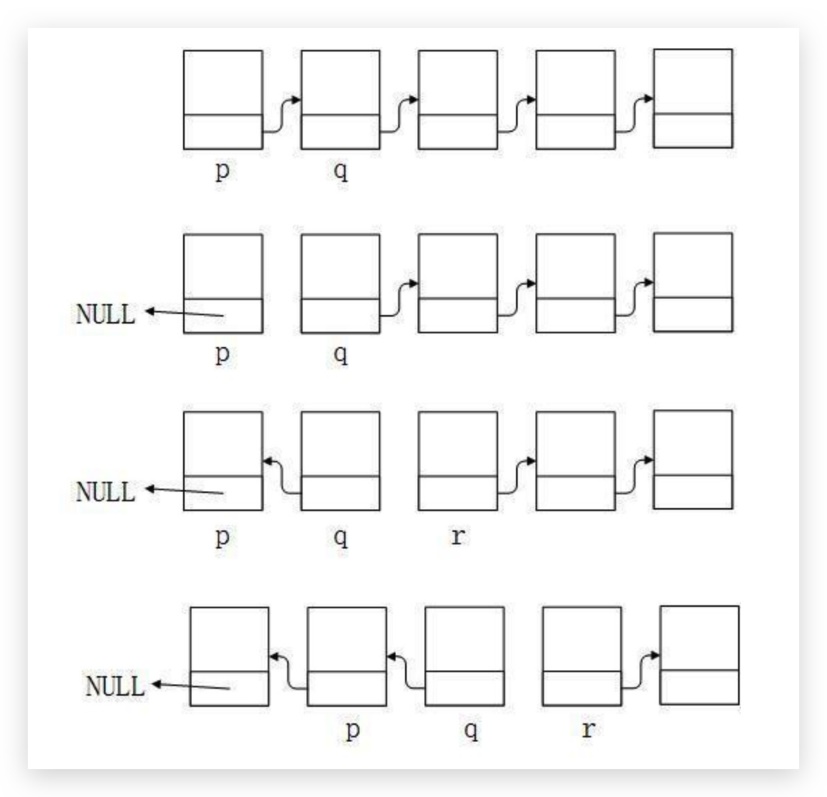
已知带头结点的链表结构定义及头插法建表、尾插法建表和打印链表等函数定义如书page 265 slnklist.h，修改slnklist.h头文件中头插法、尾插法建立函数及链表输出函数，使其能正确应用于带头结点的单链表，在此基础上，完成小项1~小项9。

**二、数据结构与算法设计**

1.数据结构：

静态链表采用基本数据类型数组作为基本元素构成，搭配一个指向自己本类的指针next。代码如下：

链表图示如下：



**三、算法复杂度分析**

⑴ 插入一个值为x的结点

循环找最后一个结点，算法复杂度为O(n)。

⑵ 链表就地逆置

多重单循环即可，算法复杂度为O（n）。

（3）删除一个值为x的结点

同插入，算法复杂度为O(n)。

(4)删除所有值等于x的结点

同（3），只需要单重循环即可实现，算法复杂度为O（n）。

（5）单链表的排序

采用最简单的选择排序法，算法复杂度为O（n^2）。

（6）单链表的合并

先将两个链表合并为一个链表，再利用选择排序进行升序和降序排序，，算法复杂度为O（n^2）。

（7）单链表的交集

嵌套循环寻找交集元素，算法复杂度为O（n^2）。

（8）单链表的调整

单重循环，算法复杂度为O（n）。

（9）单链表的访问

单重循环，算法复杂度为O（n）。

**四、测试计划**

1.编写目的

加深对数据结构链表的存储结构和其算法想的理解。

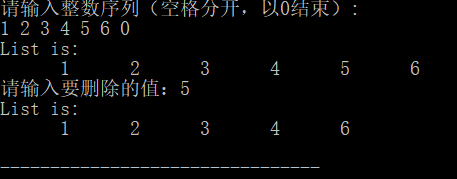
2.开发及运行环境

Dev c++

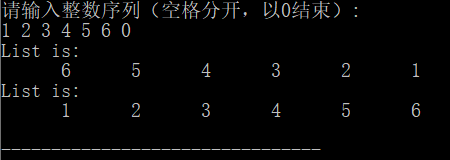
Visual Stdio Code 1.39

3.小项测试截图

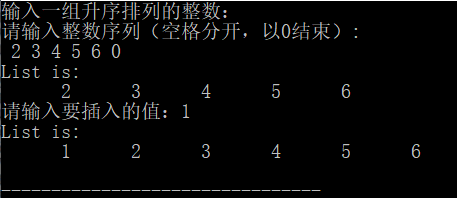
（1）删除值为x的结点

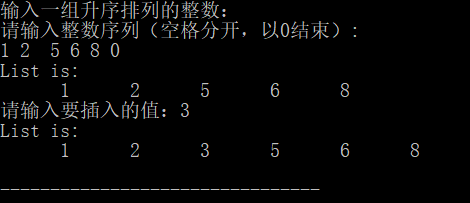
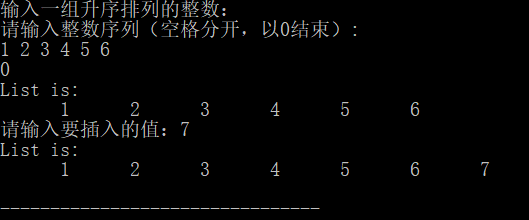


（2）链表的倒置

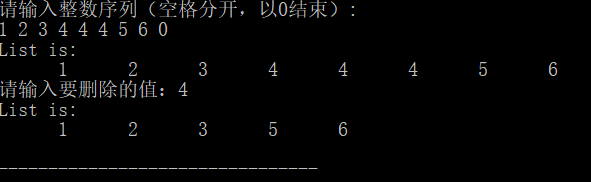


（3）插入一个值为x的结点

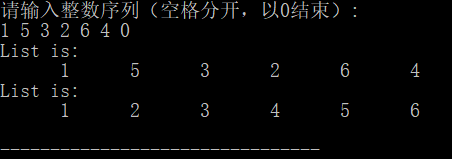




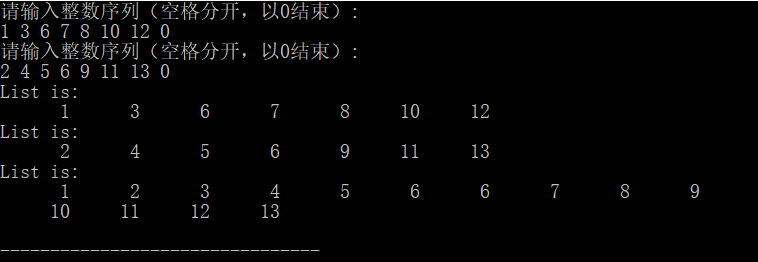
（4）单链表的删除

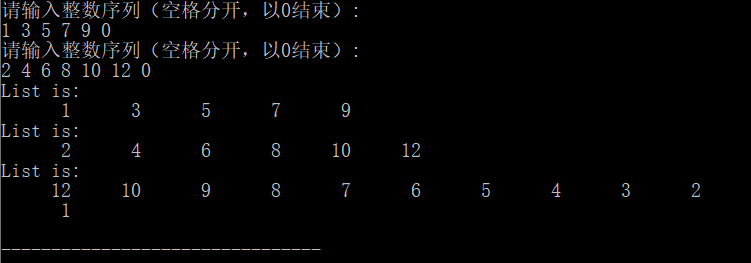


（5）单链表的排序

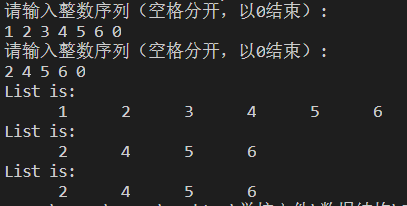


（6）链表的合并

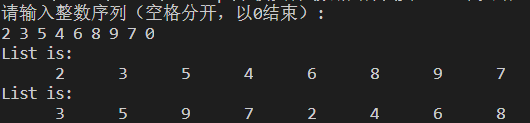




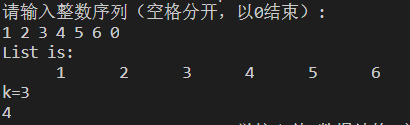
（7）链表的交集



（8）链表的调整



（9）链表的搜索



**五、源程序(仅贴算法函数部分)**

**<lab3\_01>:**

\*编写函数void delx(linklist head, datatype x)，删除带头结点单链表head中第一个值为x 的结点。

并构造测试用例进行测试。

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*文件名称：lab3\_01.c                 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "slnklist.h"

/\*请将本函数补充完整，并进行测试\*/

linklist delx(linklist head,datatype x)

{

    linklist temp = head->next;

    linklist pre = head;

    while(temp && (temp->info != x))

    {

        pre = temp;

        temp = temp->next;

    }

    if(temp)

    {

        pre->next = temp->next;

        free(temp);

    }

    return head;

}

**<lab3\_02>:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*文件名称：lab3\_02.c                 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

假设线性表（a1,a2,a3,…an）采用带头结点的单链表存储，请设计算法函数linklist reverse(linklist  head)，

将带头结点的单链表head就地倒置，使表变成（an,an-1,…a3.a2,a1）。并构造测试用例进行测试。

\*/

#include "slnklist.h"

/\*请将本函数补充完整，并进行测试\*/

linklist reverse(linklist head)

{

    linklist last2 = head->next;

    linklist last1 = head->next;

    linklist last = head->next->next;

    last1 = last;

    last = last->next;

    last1->next = last2;

    last2->next = NULL;

    last2 = last1;

    while(last->next != NULL)

    {

        last1 = last;

        last = last->next;

        last1->next = last2;

        last2 = last1;

    }

    last->next = last1;

    head->next = last;

    return head;

}

**<lab3\_03>:**

/\*

假设带头结点的单链表head是升序排列的，设计算法函数linklist insert(linklist head,datatype x)，

将值为x的结点插入到链表head中，并保持链表有序性。

分别构造插入到表头、表中和表尾三种情况的测试用例进行测试。

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*文件名称：lab3\_03.c                 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "slnklist.h"

/\*请将本函数补充完整，并进行测试\*/

linklist insert(linklist head ,datatype x)

{

    linklist pre = head;

    linklist data = malloc(sizeof(node));

    data->info = x;

    while(pre->next != NULL)

    {

        if(pre->next->info > x)

        {

            data->next = pre->next;

            pre->next = data;

            break;

        }

        pre = pre->next;

    }

    if(pre->next == NULL)

    {

        pre->next = data;

        data->next = NULL;

    }

    return head;

}

**<lab3\_04>:**

\*

编写算法函数linklist delallx(linklist head, int x)，删除带头结点单链表head中所有值为x的结点。

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*文件名称：lab3\_04.c                 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "slnklist.h"

/\*请将本函数补充完整，并进行测试\*/

linklist delallx(linklist head,int x)

{

    linklist temp = head->next;

    linklist pre = head;

    linklist del = temp;

    while(temp)

    {

        if(temp->info == x)

        {

            pre->next = temp->next;

            del = temp;

            temp = temp->next;

            free(del);

        }

        else

        {

            temp = temp->next;

            pre = pre->next;

        }

    }

    return head;

}

**<lab3\_05>:**

/\*

已知线性表存储在带头结点的单链表head中，请设计算法函数void sort(linklist head)，将head中的结点按结点值升序排列。

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*文件名称：lab3\_05.c                 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "slnklist.h"

/\*请将本函数补充完整，并进行测试\*/

void  sort(linklist head)

{

    linklist min, cp, all, pos;

    datatype data;

    if (head->next && head->next->next) //有两个以上实际结点

    {

        all = head->next;

        min = head->next;

        pos = head->next->next;

        while(all->next != NULL)

        {

            cp = pos;

            while(cp->next != NULL)

            {

                if(cp->info > min->info)

                {

                    data = cp->info;

                    cp->info = min->info;

                    min->info = data;

                }

                cp = cp->next;

            }

            if(cp->info > min->info)

            {

                data = cp->info;

                cp->info = min->info;

                min->info = data;

            }

            min = min->next;

            all = all->next;

            pos = pos->next;

        }

    }

}

**<lab3\_06>:**

/\*

已知两个带头结点的单链表L1和L2中的结点值均已按升序排序，设计算法函数

linklist mergeAscend (linklist L1,linklist L2)将L1和L2合并成一个升序的

带头结单链表作为函数的返回结果；

设计算法函数linklist mergeDescend (linklist L1,linklist L2)

将L1和L2合并成一个降序的带头结单链表作为函数的返回结果；

并设计main()函数进行测试。

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*文件名称：lab3\_06.c                 \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "slnklist.h"

/\*请将本函数补充完整，并进行测试\*/

linklist mergeAscend(linklist L1,linklist L2)

{

    linklist min, cp, all, pos, p, q;

    datatype data;

    int flag = -1;

    p = L1;

    q = L2;

    if(q->next->info < p->next->info)

    {

        flag = 2;

    }

    else flag = 1;

    while(q->next != NULL)

    {

        q = q->next;

    }

    while(p->next != NULL)

    {

        p = p->next;

    }

    switch(flag)

    {

        case 1:

            {

                while(p->next != NULL)

                {

                    p = p->next;

                }

                p->next = L2->next;

                p = L1;

                free(L2);

                all = p->next;

                min = p->next;

                pos = p->next->next;

                while(all->next != NULL)

                {

                    cp = pos;

                    while(cp->next != NULL)

                    {

                        if(cp->info < min->info)

                        {

                            data = cp->info;

                            cp->info = min->info;

                            min->info = data;

                        }

                        cp = cp->next;

                    }

                    if(cp->info < min->info)

                    {

                        data = cp->info;

                        cp->info = min->info;

                        min->info = data;

                    }

                    min = min->next;

                    all = all->next;

                    pos = pos->next;

                }

                return L1;

            }

        case 2:

            {

                while(q->next != NULL)

                {

                    q = q->next;

                }

                q->next = L1->next;

                q = L2;

                free(L1);

                all = q->next;

                min = q->next;

                pos = q->next->next;

                while(all->next != NULL)

                {

                    cp = pos;

                    while(cp->next != NULL)

                    {

                        if(cp->info < min->info)

                        {

                            data = cp->info;

                            cp->info = min->info;

                            min->info = data;

                        }

                        cp = cp->next;

                    }

                    if(cp->info < min->info)

                    {

                        data = cp->info;

                        cp->info = min->info;

                        min->info = data;

                    }

                    min = min->next;

                    all = all->next;

                    pos = pos->next;

                }

                return L2;

            }

 }

}

linklist mergeDescend(linklist L1,linklist L2)

{

linklist max, cp, all, pos, p, q;

    datatype data;

    int flag = -1;

    p = L1;

    q = L2;

    if(q->next->info > p->next->info)

    {

        flag = 2;

    }

    else flag = 1;

    while(q->next != NULL)

    {

        q = q->next;

    }

    while(p->next != NULL)

    {

        p = p->next;

    }

    switch(flag)

    {

        case 1:

            {

                while(p->next != NULL)

                {

                    p = p->next;

                }

                p->next = L2->next;

                p = L1;

                free(L2);

                all = p->next;

                max = p->next;

                pos = p->next->next;

                while(all->next != NULL)

                {

                    cp = pos;

                    while(cp->next != NULL)

                    {

                        if(cp->info > max->info)

                        {

                            data = cp->info;

                            cp->info = max->info;

                            max->info = data;

                        }

                        cp = cp->next;

                    }

                    if(cp->info > max->info)

                    {

                        data = cp->info;

                        cp->info = max->info;

                        max->info = data;

                    }

                    max = max->next;

                    all = all->next;

                    pos = pos->next;

                }

                return L1;

            }

        case 2:

            {

                while(q->next != NULL)

                {

                    q = q->next;

                }

                q->next = L1->next;

                q = L2;

                free(L1);

                all = q->next;

                max = q->next;

                pos = q->next->next;

                while(all->next != NULL)

                {

                    cp = pos;

                    while(cp->next != NULL)

                    {

                        if(cp->info > max->info)

                        {

                            data = cp->info;

                            cp->info = max->info;

                            max->info = data;

                        }

                        cp = cp->next;

                    }

                    if(cp->info > max->info)

                    {

                        data = cp->info;

                        cp->info = max->info;

                        max->info = data;

                    }

                    max = max->next;

                    all = all->next;

                    pos = pos->next;

                }

                return L2;

            }

 }

}

**<lab3\_07>:**

linklist   interSection(linklist L1, linklist L2)

{

    linklist L3 = (linklist)malloc(sizeof(node));

    linklist r, s;

    linklist temp = L1->next;

    linklist temp1 = L2->next;

    int j = 1;  //记录L1链表有几个结点

    int i = 1;  //记录L2链表有几个结点

    int k = 0;  //循环变量

    int\* data = NULL;

    int length = 0;  //临时交集数组的长度

    while(temp->next)

    {

        temp = temp->next;

        i++;

    }

    while(temp1->next)

    {

        temp1 = temp1->next;

        j++;

    }

    //哪个链表元素多，就让哪个链表做基准数组

    temp = L1->next;

    temp1 = L2->next;

    if(i > j)

    {

        data = (int \*)malloc(j \* sizeof(datatype));

        for(;i > 0; i--)

        {

            temp1 = L2->next;

            for(k = 0;k < j;k++)

            {

                if(temp->info == temp1->info)

                {

                    data[length++] = temp->info;

                    break;

                }

                temp1 = temp1->next;

            }

            temp = temp->next;

        }

    }

    else

    {

        data = (int \*)malloc((i + 1) \* sizeof(datatype));

        for(;j > 0; j--)

        {

            temp = L1->next;

            for(k = 0;k < i;k++)

            {

                if(temp->info == temp1->info)

                {

                    data[length++] = temp->info;

                    break;

                }

                temp = temp->next;

            }

            temp1 = temp1->next;

        }

    }

    data[length] = 0;

    s = L3;

    L3->info = 0;

    for(k = 0; data[k] != 0; k++)

    {

        r = (linklist)malloc(sizeof(node));

        r->info = data[k];

        r->next = NULL;

        s->next = r;

        s = r;

    }

    return L3;

}

**<lab3\_08>:**

void partion(linklist head)

{

    linklist f = head->next;

    linklist right = head->next;

    linklist r;

    linklist left = head;

    while(right->next)

    {

        right = right->next;

    }

    r = right;

    while(f != r)

    {

        switch(f->info % 2)

        {

            case 1:

            {

                f = f->next;

                left = left->next;

                break;

            }

            case 0:

            {

                left->next = f->next;

                right->next = f;

                f->next = NULL;

                f = left->next;

                right = right->next;

                break;

            }

        }

    }

    return head;

}

**<lab3\_09>:**

linklist   search(linklist head,int k)

{

    linklist temp = head;

    int length = 1;

    while(temp->next)

    {

        length++;

        temp = temp->next;

    }

    length = length - k;

    if(length <= 0)

    {

        return NULL;

    }

    if(k == 1)

    {

        return temp;

    }

    temp = head;

    for(;length > 0; length--)

    {

        temp = temp->next;

    }

    return temp;

}