

重 庆 大 学

学 生 实 验 报 告

实验课程名称 数据结构与算法

开课实验室 DS1501

学 院 软件学院 年级 2021 专业班 软件 X 班

学 生 姓 名 XXX 学 号 2021XXXX

开 课 时 间 2022 至 2023 学年第 1 学期

总 成 绩	
教师签名	XXX

软件学院制

《数据结构与算法》实验报告

开课实验室：DS1501

2022 年 9 月 22 日

学院	软件学院	年级、专业、班	2021 级软件工 程 X 班	姓名	XXX	成绩	
课程 名称	数据结构与算法	实验项目 名 称	2022-2023 学年第一学期数 据结构与算法上机练习	指导教师	XXX		
教 师 评 语	教师签名： 年 月 日						

一、实验目的

- 掌握线性表基本概念，包括理解 ADT 与具体数据结构实现的关系，当前位置概念、栅栏概念等，并实际根据书本图等键入代码到编译器中去调试。
- 熟练使用线性表的两种标准实现方法：顺序表和链表
- 上机实验先需对本章节的基本数据结构中的相关方法进行实现，然后将这些方法作为实验基础完成实验任务
- 学会在使用算法时注意对其运行时间的要求
- **主要任务：**完成实现习题 4.1、4.3、4.4、4.7。

二、使用仪器、材料

PC 微机；
Windows 操作系统，VS2010 编译环境；

三、实验步骤

- 1、先编写 List，之后派生它的实现方法类 AList 与 LList，写 LList 之前还得写个结点类 Link。以及用于调试的 Main 函数
- 2、编写相应数据结构里的 Print（）函数来输出数据，后期调试也需要不断修改该函数
- 3、编写 Main 函数来调试顺序表、链表的基本功能
- 4、习题 4.1 的解答
- 5、习题 4.3 的解答
- 6、习题 4.4 的解答
- 7、习题 4.7 的解答
- 8、完成实验报告并提交

四、实验过程原始记录(数据、图表、计算等)

1. 先编写 List，之后派生它的实现方法类 AList 与 LList，写 LList 之前还得写个结点类 Link。以及用于调试的 Main 函数；

(1)List:

```
template <typename E> class List {
private:
    void operator=(const List&) {}
    List(const List&) {}

public:
    List() {}
    virtual ~List() {}
    virtual void clear() = 0;
    virtual void insert(const E& item) = 0;
    virtual void append(const E& item) = 0;
    virtual E remove() = 0;
    virtual void moveToStart() = 0;
    virtual void moveToEnd() = 0;
    virtual void prev() = 0;
    virtual void next() = 0;
    virtual int length() const = 0;
    virtual int currPos() const = 0;
    virtual void moveToPos(int pos) = 0;
    virtual const E& getValue() const = 0;
};
```

(2)AList:

```
template<typename E> class AList : public List<E> {
private:
    int maxSize;
    int listSize;
    int curr;
    E* listArray;

public:
    AList(int size = defaultSize) {
        maxSize = size;
        listSize = curr = 0;
        listArray = new E[maxSize];
    }

    ~AList() { delete[] listArray; }

    void clear() {
```

```

        delete[] listArray;
        listSize = curr = 0;
        listArray = new E[maxSize];
    }
    void insert(const E& it) {
        Assert(listSize < maxSize, "List capacity exceeded");
        for (int i = listSize; i > curr; i--)
            listArray[i] = listArray[i - 1];
        listArray[curr] = it;
        listSize++;
    }
    void append(const E& it) {
        Assert(listSize < maxSize, "List capacity exceeded");
        listArray[listSize++] = it;
    }
    E remove() {
        Assert((curr >= 0) && (curr < listSize), "No element");
        E it = listArray[curr];
        for (int i = curr; i < listSize - 1; i++) {
            listArray[i] = listArray[i + 1];
            listSize--;
        }
        return it;
    }
    void moveToStart() { curr = 0; }
    void moveToEnd() { curr = listSize; }
    void prev() { if (curr != 0) curr--; }
    void next() { if (curr < listSize) curr++; }

    int length() const { return listSize }
    int currPos() const { return curr }

    void moveToPos(int pos) {
        Assert((pos >= 0) && (pos <= listSize), "Pos out of range");
        curr = pos;
    }

    const E& gstValue() const {
        Assert((curr >= 0) && (curr < listSize), "No current element");
        return listArray[curr];
    }
};

```

(3)Link:

```

template <typename E> class Link {
public:
    E element;

```

```

    Link* next;
    Link(const E& elemval, Link* nextval = NULL)
    { element = elemval; next = nextval; }
    Link(Link* nextval = NULL) { next = nextval; }
};

```

(4)LList:

```

template <typename E> class LList :public List<E> {
private:
    Link<E>* head;
    Link<E>* tail;
    Link<E>* curr;
    int cnt;
    void init() {
        curr = tail = head = new Link<E>;
        cnt = 0;
    }
    void removeall() {
        while (head != NULL)
        {
            curr = head;
            head = head->next;
            delete curr;
        }
    }
public:
    LList(int size = defaultSize) { init(); }//constructor
    ~LList() { removeall(); }
    void print()const;
    void clear() { removeall(); init(); }
    void insert(const E& it) {
        curr->next = new Link<E>(it, curr->next);
        if (tail == curr)tail = curr->next;
        cnt++
    }
    void append(const E& it) {
        tail = tail->next = new Link<E>(it, NULL);
        cnt++
    }
    E remove() {
        Assert(curr->next != NULL, "No element");
        E it = curr->next->element;
        Link<E>* ltemp = cur->next;
        if (tail == curr->next)tail = curr;
        curr->next = curr->next->next;
    }
};

```

```

        delete ltemp;
        cnt--;
        return it;
    }
    void moveToStrat()
    {
        curr = head;
    }
    void moveToEnd()
    {
        curr = tail;
    }

    void prev()
    {
        if (curr == head) return;
        Link<E>* temp = head;
        while (temp->next != curr; i++)
            temp = temp->next;
        curr = temp;
    }
    void next()
    {
        if (curr != tail)
            curr = curr->next;
    }
    int length() const
    {
        return cnt;
    }
    int currPos() const
    {
        Link<E>* temp = head;
        int i;
        for (i = 0; curr != temp; i++)
            temp = temp->next;
        return i;
    }

    void moveToPos(int pos)
    {
        Assert((pos >= 0) && (pos <= cnt), "Position out of range");
        curr = head;
        for (int i = 0; i < pos; i++)
            curr = curr->next;
    }

```

```

const E& getValue() const
{
    Assert(curr->next != NULL, "No value");
    return curr->next->element;
}
};

```

(5)main_a:

```

int main()
{
    Alist<int> La(100);
    La.clear();
    La.append(12);
    La.append(23);
    La.append(15);
    La.append(5);
    La.append(9);
    La.print();
    cout << endl;
    La.moveToPos(3);
    La.remove();
    La.prev();
    La.insert(6);
    La.moveToEnd();
    La.insert(10);
    La.print();
    cout << endl;
    cout << "length:" << La.length() << endl << "currPos:" << La.currPos() << endl;
    cout << "currValue:" << La.getValue() << endl;
}

```

(6)main_l:

```

int main()
{
    LList<int> Ll(100);
    Ll.clear();
    Ll.append(12);
    Ll.append(23);
    Ll.append(15);
    Ll.append(5);
    Ll.append(9);
    Ll.print();
    Ll.moveToPos(3);
    Ll.remove();
    Ll.prev();
    Ll.insert(6);
    Ll.moveToEnd();
}

```

```

Ll.insert(10);
Ll.print();
cout << "length:" << Ll.length() << endl << "currPos:" << Ll.currPos() << endl;
cout << "currValue:" << Ll.getValue() << endl;
}

```

2.编写相应数据结构里的 Print () 函数来输出数据，后期调试也需要不断修改该函数;

(1)顺序表:

```

template <class E>
class Alist : public List<E> {
private:
    int listSize;
    int curr;
    E* listArray;
public:
    // 构造函数
    Alist(int size) {
        listSize = 0;
        curr = 0;
        listArray = new E[size];
    }
    // 析构函数
    ~Alist() { delete[] listArray; }
    void print()
    {
        for (int i = 0; i < listSize; i++)
            cout << listArray[i] << " ";

    }
    // 清空顺序表函数
    void clear() {
        delete[] listArray;
        listSize = curr = 0;
        listArray = new T[maxSize];
    }
    // 在当前位置插入值 x
    void insert(const E& x) {
        if (listSize >= maxSize || curr < 0) {
            cout << "List capacity exceeded";
            return;
        }
        for (int j = listSize; j >= curr; j--)
            listArray[j] = listArray[j - 1];
        listArray[curr] = x;
        listSize++;
    }
}

```



```

}

// 在顺序表末尾插入 x
void append(const E& x) {
    if (listSize >= maxSize) {
        cout << "List capacity exceeded";
        return;
    }
    listArray[listSize++] = x;
}

// 删除当前元素
E remove() {
    if (curr < 0 || curr >= listSize) {
        cout << "No element";
        return 0;
    }
    E x = listArray[curr];
    for (int j = curr; j < listSize - 1; j++)
        listArray[j] = listArray[j + 1];
    listSize--;
    return x;
}

// 将当前位置移动到头部
void moveToStart() { curr = 0; }
// 将当前位置移动到尾部
void moveToEnd() { curr = listSize; }
// 将当前位置向前移动一格

void prev() { if (curr != 0) curr--; }
// 将当前位置向后移动一格
void next() { if (curr < listSize) curr++; }
// 返回顺序表长度
int length() const { return listSize; }
// 返回当前位置
int currPos() const { return curr; }
// 将当前位置移动到 pos
void moveToPos(int pos) {
    if (pos < 0 || pos > listSize) {
        cout << "Pos out of range";
        return;
    }
    curr = pos;
}

// 返回当前位置的值
T getValue() const {
    if (curr < 0 || curr >= listSize) {
        cout << "No current element";
    }
}

```

```

        return 0;
    }
    return listArray[curr];
}
};

```

(2)链表:

```

template <class E>
class LList : public List<E>
{
private:
    Link<E>* head;
    Link<E>* tail;
    Link<E>* curr;
    int cnt;

    void init()
    {
        curr = tail = head = new Link<E>;
        cnt = 0;
    }

    void removeall()
    {
        while (head != NULL)
        {
            curr = head;
            head = head->next;
            delete curr;
        }
    }

public:
    LList(int size) { init(); }
    ~LList() { removeall(); }
    void print() const
    {
        Link<E>* ltemp = head->next;
        while(ltemp != NULL)
        {
            cout << ltemp->element << " ";
            ltemp = ltemp->next;
        }
        cout << endl;
        delete ltemp;
    }

    void clear() { removeall(); init(); }

```

```

void insert(const E& x)
{
    curr->next = new Link<E>(x, curr->next);
    if (tail == curr)tail = curr->next;// 若当前位置结点之前为尾结点，则新插入结点为新尾
    结点
    cnt++;
}
void append(const T& x)
{
    tail = tail->next = new Link<E>(x, NULL);
    cnt++;
}
T remove()
{
    assert(curr->next != NULL, "No element");
    T x = curr->next->element;// 复制当前位置结点值
    Link<T>* ltemp = curr->next;// 复制当前位置结点指针
    if (tail == curr->next)tail = curr;//若要删除的结点为尾结点，需要指定 curr 为新尾结点
    curr->next = curr->next->next;
    delete ltemp;
    cnt--;
    return x;
}
void moveToStart()
{
    curr = head;
}
void moveToEnd()
{
    curr = tail;
}
void prev()
{
    if (curr == head)return;
    Link<T>* temp = head;
    while (temp->next != curr) temp = temp->next;
    curr = temp;
}
void next()
{
    if (curr == tail)return;
    if (curr != tail)curr = curr->next;
}
int length()const { return cnt; }
int currPos()const
{

```

```

    Link<E>* temp = head;
    int i;
    for (i = 0; temp != curr; i++)
        temp = temp->next;
    return i;
}

void moveToPos(int pos)
{
    assert((pos >= 0) && (pos <= cnt), "Position out of range");
    curr = head;
    for (int i = 0; i < pos; i++) curr = curr->next;

}

T getValue()const
{
    assert(curr->next != NULL, "No value");
    return curr->next->element;
}

};

```

3. 编写 Main 函数来调试顺序表、链表的基本功能;

(1)顺序表:

```

int main()
{
    Alist<int> La(100);
    La.clear();
    La.append(12);
    La.append(23);
    La.append(15);
    La.append(5);
    La.append(9);
    La.print();
    cout << endl;
    La.moveToPos(3);
    La.remove();
    La.prev();
    La.insert(6);
    La.moveToEnd();
    La.insert(10);
    La.print();
    cout << endl;
    cout << "length:" << La.length() << endl << "currPos:" << La.currPos() << endl;
    cout << "currValue:" << La.getValue() << endl;
}

```

结果:

```
C:\> 选择Microsoft Visual Studio 调试控制台
12 23 15 5 9
12 23 6 15 9 10
length:6
currPos:5
currValue:10

D:\source\repos\array based list\Debug\
要在调试停止时自动关闭控制台，请启用 “工
按任意键关闭此窗口. . .
```

(2)链表:

```
int main()
{
    LList<int> L1(100);
    L1.clear();
    L1.append(12);
    L1.append(23);
    L1.append(15);
    L1.append(5);
    L1.append(9);
    L1.print();
    L1.moveToPos(3);
    L1.remove();
    L1.prev();
    L1.insert(6);
    L1.moveToEnd();
    L1.insert(10);
    L1.print();
    cout << "length:" << L1.length() << endl << "currPos:" << L1.currPos() << endl;
    cout << "currValue:" << L1.getValue() << endl;
}
```

结果:

```
C:\> Microsoft Visual Studio 调试控制台
12 23 15 5 9
12 23 6 15 9 10
length:6
currPos:5
currValue:10

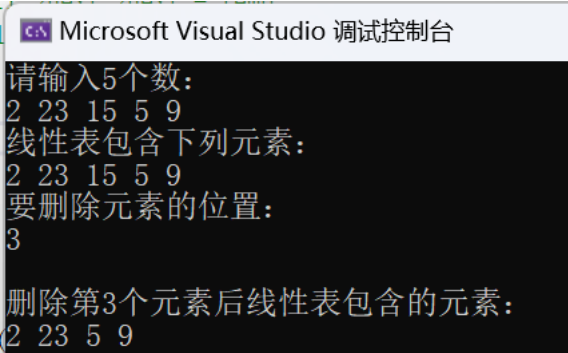
D:\source\repos\link_list\Debug\
要在调试停止时自动关闭控制台，请
按任意键关闭此窗口. . .
```

4. 习题 4.1;

(1) 关键代码:

```
int main()
{
    int num;
    int i;
    LList<int>L1(10);
    L1.clear();
    cout << "请输入5个数: " << endl;
    for (i = 0; i < 5; i++) {
        cin >> num;
        L1.append(num);    //在线性表的尾部插入节点
    }
    cout << "线性表包含下列元素: " << endl;
    L1.print();            //打印线性表的所有元素
    cout << endl;
    int k;
    cout << "要删除元素的位置: " << endl;
    cin >> k;
    L1.moveToPos(k-1);    //将curr移动到要删除的元素前
    L1.remove();          //删除curr的后一个节点
    cout << endl << "删除第"<<k<<"个元素后线性表包含的元素: " << endl;
    L1.print();
    return 0;
}
```

(2) 程序运行结果:



```
Microsoft Visual Studio 调试控制台
请输入5个数:
2 23 15 5 9
线性表包含下列元素:
2 23 15 5 9
要删除元素的位置:
3

删除第3个元素后线性表包含的元素:
2 23 5 9
```

5. 习题 4.3;

(1) 关键代码:

```
int main()
{
    int num;
    int i;
    LList<int>L1(20);
    L1.clear();
    cout << "请输入: " << endl;
    for (i = 0; i < 5; i++) {
        cin >> num;
```

```

        L1.append(num);    //在线性表的尾部插入节点
    }
    cout << "线性表包含下列元素：" << endl;
    L1.print();            //打印线性表的所有元素
    cout << endl;
    return 0;
}

```

(2) 程序运行结果:



The screenshot shows the Microsoft Visual Studio Debug Console. It displays the following text:

请输入:

2

23

15

5

9

线性表包含下列元素:

2 23 15 5 9

6. 习题 4.4;

(1) 关键代码:

```

int LinkSort(LList& l, ElemType e) {
    LList p, q, s;
    int i = 1;
    p = l;
    q = p->next;
    s = q->next;
    if (q->data == e)
        return 1;
    while (s) {
        if (s->data != e) {
            p = p->next;
            q = q->next;
            s = s->next;
            i++;
        }
        else {
            p->next = s;
            q->next = s->next;
            s->next = q;
            return i;
        }
    }
    return 0;
}

```

```

int main() {
    LList L;
    ElemType x;
    cout << "输入元素: ";
    InitList1(L);
    cout << "输入最后一位元素: ";
    cin >> x;
    cout << "初始状态的表: ";
    print(L);
    if (LinkSort(L, x) != 0) {
        cout << "交换后的表: ";
        print(L);
    }
    else {
        cout << "没有找到该元素。" << endl;
    }
    return 0;
}

```

(2) 程序运行结果:



```

Microsoft Visual Studio 调试控制台
输入元素: 6 7 12 42 19 87
输入最后一位元素: 87
初始状态的表: 6 7 12 42 19 87
交换后的表: 6 7 12 42 87 19

```

7. 习题 4.7;

(1) 关键代码:

```

LNode* LList::createLinkList(int n)
{
    int value;
    linkListCnt = 0;
    for (int i = 1; i <= n; ++i)
    {
        printf("请输入链表中的第%d个数: ", i);
        scanf_s("%d", &value);
        tail->next = new Node(value, NULL);
        tail = tail->next;
        linkListCnt++;
    }
    return head;
}

LNode* LList::bubbleSortLinkList(LNode* headNode)
{
    LNode* tempNode1 = NULL;
    LNode* tempNode2 = NULL;

```



```

if (headNode == tail)
    return NULL;
for (tempNode1 = headNode->next; tempNode1 != NULL; tempNode1 = tempNode1->next)
{
    for (tempNode2 = headNode->next; tempNode2 != NULL; tempNode2 = tempNode2->next)
    {
        if (tempNode1->data < tempNode2->data)
        {
            int temp = tempNode1->data;
            tempNode1->data = tempNode2->data;
            tempNode2->data = temp;
        }
    }
}
return headNode;
}

```

```

LNode* mergeLinkList(LNode* list1Node, LNode* list2Node)
{
    LNode* tempNode = list1Node;
    for (; tempNode->next != NULL; tempNode = tempNode->next) { ; }

    tempNode->next = list2Node->next;
    list2Node->next = NULL;
    return list1Node;
}

```

(2) 主程序:

```

int main()
{
    int a;
    int b;
    LList list1(10), list2(10);
    cout << "请设置链表一的元素个数 (不超过10): " << endl;
    cin >> a;
    LNode* list1Node = list1.createLinkList(a);
    cout << "请设置链表二的元素个数 (不超过10): " << endl;
    cin >> b;
    LNode* list2Node = list2.createLinkList(b);
    cout << "两个链表连接后的链表: " << endl;
    LNode* tempNode = mergeLinkList(list1Node, list2Node);
    list1.bubbleSortLinkList(tempNode);
    list1.print(tempNode);
    return 0;
}

```

(3) 程序运行结果:

```
Microsoft Visual Studio 调试控制台
请设置链表一的元素个数（不超过10）：
6
请输入链表中的第1个数： 1
请输入链表中的第2个数： 3
请输入链表中的第3个数： 4
请输入链表中的第4个数： 2
请输入链表中的第5个数： 5
请输入链表中的第6个数： 6
请设置链表二的元素个数（不超过10）：
7
请输入链表中的第1个数： 13
请输入链表中的第2个数： 24
请输入链表中的第3个数： 43
请输入链表中的第4个数： 65
请输入链表中的第5个数： 34
请输入链表中的第6个数： 78
请输入链表中的第7个数： 9
两个链表连接后的链表：
1 2 3 4 5 6 9 13 24 34 43 65 78
```

五、实验结果及分析

结果都已对应显示在原始数据记录中，结果都与预期的分析符合。