# 重庆交通大学信息科学与工程学院

# 《数据结构 A》实验报告

| | |
|---|---|
| 实验名称 | 线性表的链式存储结构 |
| 课程名称 | 数据结构 A |
| 专业班级 | 曙光 2101 班 |
| 学　　号 | 632107060510 |
| 姓　　名 | 王靖 |
| 指导教师 | 鲁云平 |

2022 年 10 月

# 《数据结构 A》综合性实验评分标准

| 评分等级 | 综合性实验评分标准 |
| --- | --- |
| 优 | 程序演示完全正确，界面美观，能正确回答 90%及以上的问题；报告规范，分析清楚，严格按照要求条目书写，阐述清楚。能针对综合性题目进行分析，并设计合适的解决方案，同时使用合适的编程平台进行编程、调试、测试和实现。 |
| 良 | 按要求完成 80%及以上功能，界面尚可，能正确回答 80%及以上的问题；报告规范，分析清楚，个别条目书写不完全符合要求，阐述基本清楚。能针对综合性题目进行分析，并设计较合适的解决方案，同时使用合适的编程平台进行编程、调试、测试和实现。 |
| 中 | 按要求完成 70%及以上功能，能回答 70%及以上的问题；报告基本规范，分析基本清楚，存在 30%以内条目书写不完全符合要求。在教师的指导下，能针对综合性题目进行分析，并设计较合适的解决方案，同时使用合适的编程平台进行编程、调试、测试和实现。 |
| 及格 | 按要求完成 60%及以上功能，能回答老师多数问题；报告基本规范，存在 40%以内条目书写不完全符合要求。在教师的指导下，能针对综合性题目进行分析和设计较合适的解决方案，并能使用合适的编程平台进行编程、调试、测试和实现，成功调试功能达 60%以上。 |
| 不及格 | 存在 40%以上功能未完成或抄袭；报告不规范或存在 40%以上条目书写不完全符合要求。无法采用正确的方法完成项目分析和解决方案设计或成功调试测试功能超过 40%未完成。 |

| | |
| --- | --- |
| **教师签名** | |
| **综合性实验成绩** | |

目录

## 一、实验目的

1、实现线性表的链式存储结构

2、熟悉 C++程序的基本结构，掌握程序中的头文件、实现文件和主文件之间的相互关系及各自的作用

3、熟悉链表的基本操作方式,掌握链表相关操作的具体实现

## 二、实验内容

对链式存储的线性表进行基本操作。包括：

（1）插入：操作方式为在指定元素前插入、在指定元素之后插入、在指定位置完成插入

（2）删除：操作方式可分为删除指定元素、删除指定位置的元素等，尝试实现逻辑删除操作。

（3）显示数据

（4）查找：查询指定的元素（可根据某个数据成员完成查询操作）

（5）定位操作：定位指定元素的序号

（6）排序操作：按升序排序

（7）反转操作：将链表结构逆转

（8）更新：修改指定元素的数据

（9）数据文件的读写操作

## 三、系统分析

(1) 数据方面

通过使用标准数据类型(int 等)和自定义数据类型利用自定义实现封装的顺序表模板进行基本操作

(2) 功能方面:

① 插入：操作方式为在指定元素前插入、在指定元素之后插入、在指定位置完成插入

② 删除：操作方式可分为删除指定元素、删除指定位置的元素等，尝试实现逻辑删除操作。

③ 显示数据

④ 查找：查询指定的元素（可根据某个数据成员完成查询操作）

⑤ 定位操作：定位指定元素的序号

⑥ 排序操作：按升序排序

⑦ 反转操作：将链表结构逆转

⑧ 更新：修改指定元素的数据

⑨ 数据文件的读写操作

## 四、系统设计

### （1）数据结构的设计

节点类模板

```
template<class T>
class ListNode
{
public:
    T data; //数据域
    ListNode<T>* next; //指针域
public:
    ListNode(); //构造函数
    ListNode(T data); //带参构造函数
    ~ListNode(); //析构函数
};
```

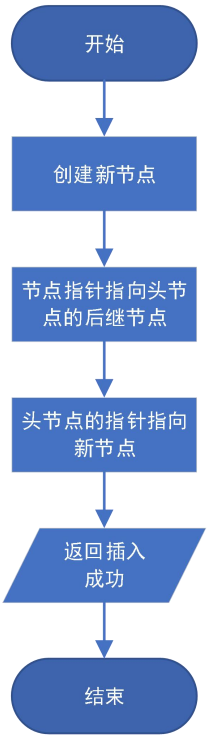
链表类模板（带头节点的单链表）

```
template<class T>
class List
{
private:
    ListNode<T>* head; //头指针
    int numOfNode; //节点个数
public:
    List(); //构造函数
    ~List(); //析构函数
    void insertByHead(T elem); //头部插入
    void insertByTail(T elem); //尾部插入
    bool insertByFrontPos(int pos, T elem); //指定位置前插入
    bool insertByBehindPos(int pos, T elem); //指定位置后插入
    bool insertByFrontPosElem(T posElem, T elem); //元素前插入
    bool insertByBehindPosElem(T posElem, T elem); //元素后插入
    bool deleteByPos(int pos); //删除指定位置元素
    bool deleteByElem(T posElem); //删除指定元素
    int findElemPos(T posElem); //查找元素位置
    ListNode<T>* findElem(T posElem); //查找元素
    int nodeNum(); //获取节点个数
    void sortList(); //升序排序
```

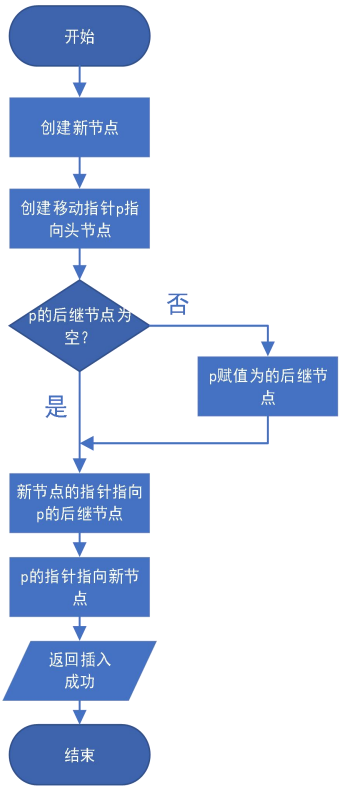
```
    void reverseList(); //反转链表
    void printList(); //打印链表
};
```

（2）基本操作的设计

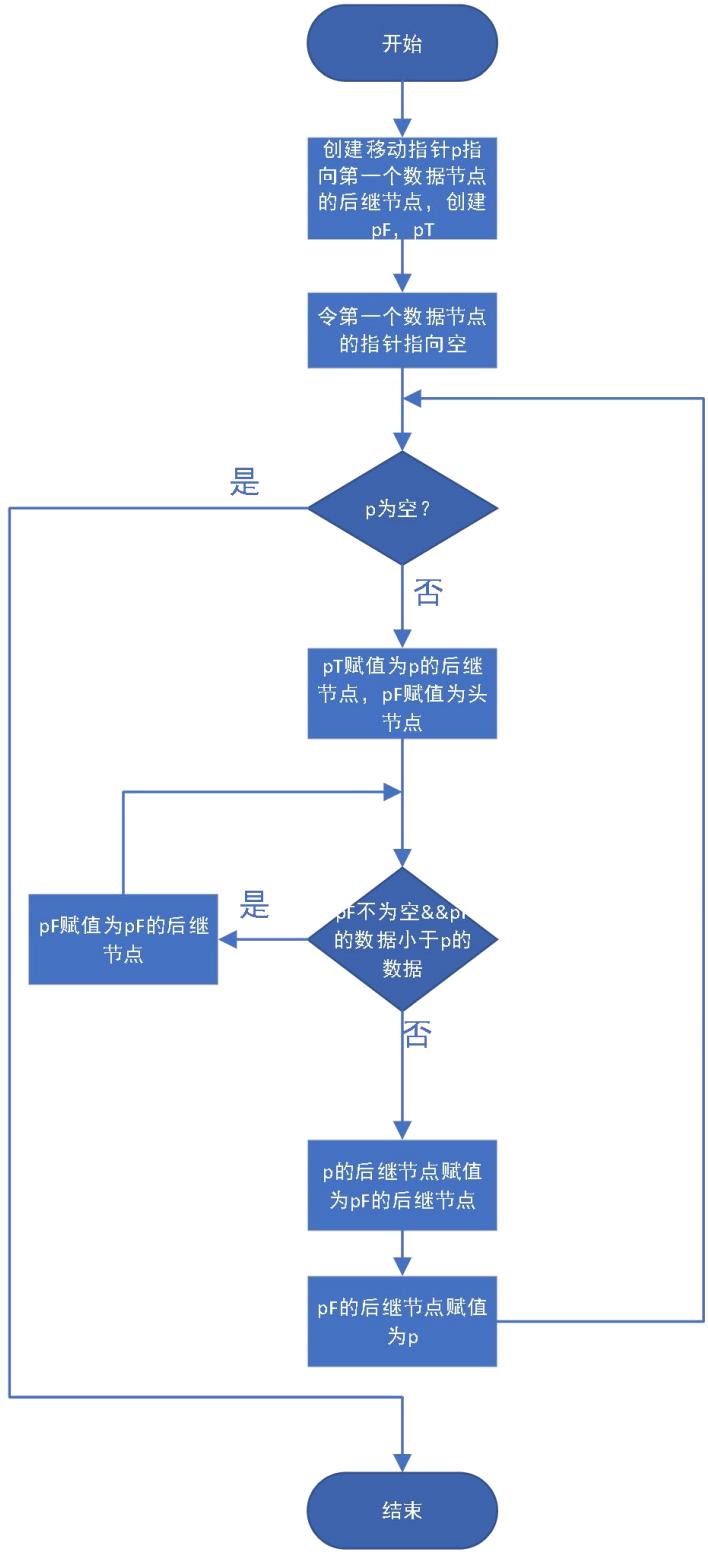
- 头部添加元素



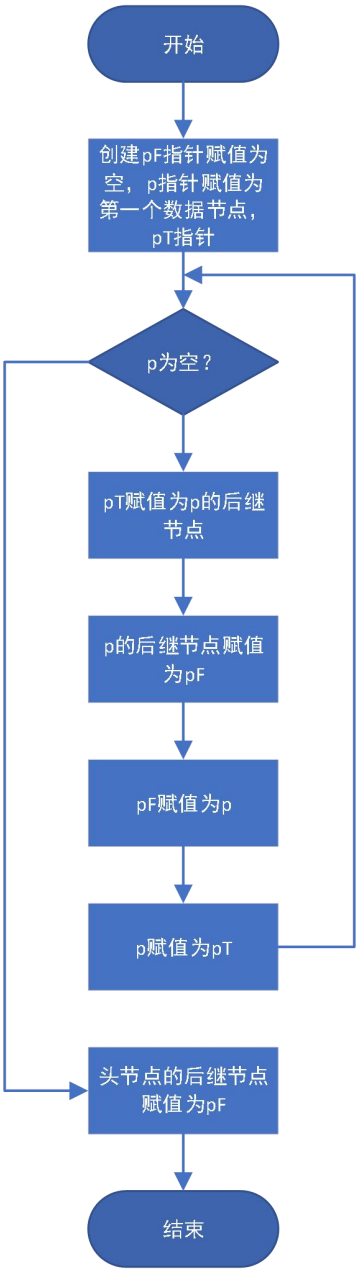

- 尾部添加元素



- 升序排序


开始
创建移动指针p指向第一个数据节点的后继节点，创建pF，pT
令第一个数据节点的指针指向空
p为空？
是
否
pT赋值为p的后继节点，pF赋值为头节点
pF不为空&&pF的数据小于p的数据
是
pF赋值为pF的后继节点
否
p的后继节点赋值为pF的后继节点
pF的后继节点赋值为p
结束

- 逆转链表




## 五、系统实现

- 头部添加元素

```
template<class T>
void List<T>::insertByHead(T elem)
{
    ListNode<T>* newNode = new ListNode<T>(elem);
    newNode->next = head->next;
    head->next = newNode;
    numOfNode++;
}
```

- 尾部添加元素

```
template<class T>
void List<T>::insertByTail(T elem)
{
    ListNode<T>* newNode = new ListNode<T>(elem);
    ListNode<T>* pMove = head;
    while (pMove->next != NULL)
    {
        pMove = pMove->next;
    }
    pMove->next = newNode;
    numOfNode++;
}
```

- 指定位置前插入

```
template<class T>
bool List<T>::insertByFrontPos(int pos, T elem)
{
    if (pos <= 0 || pos > numOfNode) return false;
    ListNode<T>* newNode = new ListNode<T>(elem);
    ListNode<T>* posFrontNode = head;
    ListNode<T>* posNode = head->next;
    for (int i = 1; i < pos; i++)
    {
        posFrontNode = posNode;
        posNode = posNode->next;
    }
    newNode->next = posNode;
    posFrontNode->next = newNode;
    numOfNode++;
    return true;
}
```

- 指定位置后插入

```
template<class T>
bool List<T>::insertByBehindPos(int pos, T elem)
{
    if (pos <= 0 || pos > numOfNode) return false;
    ListNode<T>* newNode = new ListNode<T>(elem);
    ListNode<T>* posNode = head->next;
    ListNode<T>* posBehindNode = posNode->next;
    for (int i = 1; i < pos; i++)
    {
        posNode = posBehindNode;
        posBehindNode = posBehindNode->next;
    }
```

```
    newNode->next = posBehindNode;
    posNode->next = newNode;
    numOfNode++;
    return true;
}
```

- 指定元素前插入

```
template<class T>
bool List<T>::insertByFrontPosElem(T posElem, T elem)
{
    ListNode<T>* newNode = new ListNode<T>(elem);
    ListNode<T>* posFrontNode = head;
    ListNode<T>* posNode = head->next;
    while (!(posNode->data == posElem) && posNode != NULL)
    {
        posFrontNode = posNode;
        posNode = posNode->next;
    }
    if (posNode == NULL) return false;
    else
    {
        newNode->next = posNode;
        posFrontNode->next = newNode;
    }
    numOfNode++;
    return true;
}
```

- 指定元素后插入

```
template<class T>
bool List<T>::insertByBehindPosElem(T posElem, T elem)
{
    ListNode<T>* newNode = new ListNode<T>(elem);
    ListNode<T>* posNode = head->next;
    ListNode<T>* posBehindNode = posNode->next;
    while (!(posNode->data == posElem) && posBehindNode != NULL)
    {
        posNode = posBehindNode;
        posBehindNode = posBehindNode->next;
    }
    if (posBehindNode == NULL && !(posNode->data == posElem))
        return false;
    else
    {
        newNode->next = posBehindNode;
```

```
        posNode->next = newNode;
    }
    numOfNode++;
    return true;
}
```

- 删除指定位置元素

```
template<class T>
bool List<T>::deleteByPos(int pos)
{
    if (head->next == NULL || pos > numOfNode) return false;
    ListNode<T>* posFrontNode = head;
    ListNode<T>* posNode = head->next;
    for (int i = 1; i < pos; i++)
    {
        posFrontNode = posNode;
        posNode = posNode->next;
    }
    posFrontNode->next = posNode->next;
    delete posNode;
    posNode = NULL;
    numOfNode--;
    return true;
}
```

- 删除指定元素

```
template<class T>
bool List<T>::deleteByElem(T posElem)
{
    if (head->next == NULL) return false;
    ListNode<T>* posFrontNode = head;
    ListNode<T>* posNode = head->next;
    while (posNode != NULL && !(posNode->data == posElem))
    {
        posFrontNode = posNode;
        posNode = posNode->next;
    }
    if (posNode == NULL) return false;
    else
    {
        posFrontNode->next = posNode->next;
        delete posNode;
        posNode = NULL;
    }
    numOfNode--;
```

```
    return true;
}
```

- 查找元素位置

```
template<class T>
int List<T>::findElemPos(T posElem)
{
    if (head->next == NULL) return -1;
    int pos = 1;
    ListNode<T>* pMove = head->next;
    while (pMove != NULL)
    {
        if (pMove->data == posElem) return pos;
        pos++;
    }
    return -1;
}
```

- 获取元素

```
template<class T>
ListNode<T>* List<T>::findElem(T posElem)
{
    if (head->next == NULL) return NULL;
    ListNode<T>* pMove = head->next;
    while (pMove != NULL)
    {
        if (pMove->data == posElem) break;
        pMove = pMove->next;
    }
    return pMove;
}
```

- 排序

```
template<class T>
void List<T>::sortList()
{
    if (head->next == NULL) return;
    ListNode<T>* pMove = head->next->next;
    ListNode<T>* pFront;
    ListNode<T>* pTemp;
    head->next->next = NULL;
    while (pMove != NULL)
    {
        pTemp = pMove->next;
        pFront = head;
        while (pFront->next != NULL &&
```

```
            pFront->next->data <pMove->data)
        {
            pFront = pFront->next;
        }
        pMove->next = pFront->next;
        pFront->next = pMove;
        pMove = pTemp;
    }
}
```

- 反转链表

```
template<class T>
void List<T>::reverseList()
{
    if (head->next == NULL) return;
    ListNode<T>* pFront = NULL;
    ListNode<T>* pMove = head->next;
    while (pMove != NULL)
    {
        ListNode<T>* temp = pMove->next;
        pMove->next = pFront;
        pFront = pMove;
        pMove = temp;
    }
    head->next = pFront;
}
```

- 获取链表节点个数

```
template<class T>
int List<T>::nodeNum()
{
    return numOfNode;
}
```

- 打印输出链表

```
template<class T>
void List<T>::printList()
{
    if (head->next == NULL) return;
    ListNode<T>* pMove = head->next;
    while (pMove != NULL)
    {
        cout << pMove->data << ' ';
        pMove = pMove->next;
    }
    cout << endl;
```

```
}
```

## 六、系统测试与结果

(1) 使用标准数据类型测试

测试函数

```
void testByInt()
{
    List<int> list1;
    cout << "尾部插入 1,2,3,4: " << endl;
    list1.insertByTail(1);
    list1.insertByTail(2);
    list1.insertByTail(3);
    list1.insertByTail(4);
    cout << "头部插入 9: " << endl;
    list1.insertByHead(9);
    cout << "第一次打印: " << endl;
    list1.printList();
    cout << "1 号位置前插入 8: " << endl;
    list1.insertByFrontPos(1, 8);
    cout << "4 号位置后插入 0: " << endl;
    list1.insertByBehindPos(4, 0);
    cout << "元素 4 前插入 8: " << endl;
    list1.insertByFrontPosElem(4, 8);
    cout << "元素 2 后插入 0: " << endl;
    list1.insertByBehindPosElem(2, 0);
    cout << "第二次打印: " << endl;
    list1.printList();
    cout << "反转链表: " << endl;
    list1.reverseList();
    cout << "第四次打印: " << endl;
    list1.printList();
    cout << "排序: " << endl;
    list1.sortList();
    cout << "第五次打印: " << endl;
    list1.printList();
    cout << "删除元素 2: " << endl;
    list1.deleteByElem(2);
    cout << "第六次打印: " << endl;
    list1.printList();
}
```

运行结果

```
尾部插入1,2,3,4:
头部插入9:
第一次打印:
9 1 2 3 4
1号位置前插入8:
4号位置后插入0:
元素4前插入8:
元素2后插入0:
第二次打印:
8 9 1 2 0 0 3 8 4
反转链表:
第四次打印:
4 8 3 0 0 2 1 9 8
排序:
第五次打印:
0 0 1 2 3 4 8 8 9
删除元素2:
第六次打印:
0 0 1 3 4 8 8 9
```

(2) 使用自定义数据类型测试

自定义 Student 类

```
class Student
{
private:
    string name;
    string stuNumber;
    int score[3];
    int sum;
public:
    Student() {}
    Student(string name, string number, int math,
            int english, int program)
    {
        this->name = name;
        this->stuNumber = number;
        score[0] = math;
        score[1] = english;
        score[2] = program;
        sum = math + english + program;
    }
    bool operator<(const Student& s) const
```

```
    {
        if (sum == s.sum)
        {
            if (score[0] != s.score[0])
                return score[0] < s.score[0];
            else if (score[1] != s.score[1])
                return score[1] < s.score[1];
            else return score[2] < s.score[2];
        }
        return sum < s.sum;
    }
    bool operator>(const Student& s) const
    {
        if (sum == s.sum)
        {
            if (score[0] != s.score[0])
                return score[0] > s.score[0];
            else if (score[1] != s.score[1])
                return score[1] > s.score[1];
            else return score[2] > s.score[2];
        }
        return sum > s.sum;
    }
    bool operator==(const Student& s)
    {
        // 只通过学号比较
        return stuNumber == s.stuNumber;
    }
    friend ostream& operator<<(ostream&, Student s);
};

ostream& operator<<(ostream& os, Student s)
{
    cout << "姓名：" << s.name << ' ';
    cout << "学号：" << s.stuNumber << ' ';
    cout << "数学：" << s.score[0] << ' ';
    cout << "英语：" << s.score[1] << ' ';
    cout << "程序：" << s.score[2] << ' ';
    cout << "总分：" << s.sum << endl;
    return cout;
}
```

测试函数

```
void testByStudent()
{
```

```
    List<Student> list2;
    list2.insertByTail(Student("李四", "63210506", 97, 87, 60));
    list2.insertByTail(Student("张三", "63210501", 93, 65, 86));
    list2.insertByTail(Student("王五", "63210509", 90, 20, 53));
    list2.insertByHead(Student("赵四", "63210505", 91, 66, 81));
    Student s1("测试", "12120003", 65, 98, 64);
    Student s2("李四", "63210506", 97, 87, 60);
    Student s3("赵六", "63210505", 91, 66, 81);
    cout << "添加 Student 测试" << endl;
    list2.printList();
    cout << "李四后插入 Student 测试" << endl;
    list2.insertByBehindPosElem(s2, s1);
    list2.printList();
    cout << "赵四前插入 Student 测试" << endl;
    list2.insertByFrontPosElem(s3, s1);
    list2.printList();
    cout << "排序测试" << endl;
    list2.sortList();
    list2.printList();
    cout << "反转测试" << endl;
    list2.reverseList();
    list2.printList();
    cout << "删除测试" << endl;
    list2.deleteByElem(s3);
    list2.printList();
}
```

运行结果

```
添加Student测试
姓名：赵四 学号：63210505 数学：91 英语：66 程序：81 总分：238
姓名：李四 学号：63210506 数学：97 英语：87 程序：60 总分：244
姓名：张三 学号：63210501 数学：93 英语：65 程序：86 总分：244
姓名：王五 学号：63210509 数学：90 英语：20 程序：53 总分：163

李四后插入Student测试
姓名：赵四 学号：63210505 数学：91 英语：66 程序：81 总分：238
姓名：李四 学号：63210506 数学：97 英语：87 程序：60 总分：244
姓名：测试 学号：12120003 数学：65 英语：98 程序：64 总分：227
姓名：张三 学号：63210501 数学：93 英语：65 程序：86 总分：244
姓名：王五 学号：63210509 数学：90 英语：20 程序：53 总分：163

赵四前插入Student测试
姓名：测试 学号：12120003 数学：65 英语：98 程序：64 总分：227
姓名：赵四 学号：63210505 数学：91 英语：66 程序：81 总分：238
姓名：李四 学号：63210506 数学：97 英语：87 程序：60 总分：244
姓名：测试 学号：12120003 数学：65 英语：98 程序：64 总分：227
姓名：张三 学号：63210501 数学：93 英语：65 程序：86 总分：244
姓名：王五 学号：63210509 数学：90 英语：20 程序：53 总分：163

排序测试
姓名：王五 学号：63210509 数学：90 英语：20 程序：53 总分：163
姓名：测试 学号：12120003 数学：65 英语：98 程序：64 总分：227
姓名：测试 学号：12120003 数学：65 英语：98 程序：64 总分：227
姓名：赵四 学号：63210505 数学：91 英语：66 程序：81 总分：238
姓名：张三 学号：63210501 数学：93 英语：65 程序：86 总分：244
姓名：李四 学号：63210506 数学：97 英语：87 程序：60 总分：244

反转测试
姓名：李四 学号：63210506 数学：97 英语：87 程序：60 总分：244
姓名：张三 学号：63210501 数学：93 英语：65 程序：86 总分：244
姓名：赵四 学号：63210505 数学：91 英语：66 程序：81 总分：238
姓名：测试 学号：12120003 数学：65 英语：98 程序：64 总分：227
姓名：测试 学号：12120003 数学：65 英语：98 程序：64 总分：227
姓名：王五 学号：63210509 数学：90 英语：20 程序：53 总分：163

删除测试
姓名：李四 学号：63210506 数学：97 英语：87 程序：60 总分：244
姓名：张三 学号：63210501 数学：93 英语：65 程序：86 总分：244
姓名：测试 学号：12120003 数学：65 英语：98 程序：64 总分：227
姓名：测试 学号：12120003 数学：65 英语：98 程序：64 总分：227
```

(3) 使用文件测试

测试函数

```
void testByFile()
{
    ifstream cin("test.in");
    ofstream cout("test.out",ios::app);
    List<int> list1;
    int a, b, c, d, e, f, g, h, i, j, k;
    cin >> a >> b >> c >> d >> e;
    cout << "尾部插入 1,2,3,4: " << endl;
    list1.insertByTail(a);
    list1.insertByTail(b);
    list1.insertByTail(c);
    list1.insertByTail(d);
    cout << "头部插入 9: " << endl;
    list1.insertByHead(e);
    cout << "第一次打印: " << endl;
    list1.printList();
    cout << "1 号位置前插入 8: " << endl;
    list1.insertByFrontPos(1, 8);
    cout << "4 号位置后插入 0: " << endl;
    list1.insertByBehindPos(4, 0);
    cout << "元素 4 前插入 8: " << endl;
    list1.insertByFrontPosElem(4, 8);
    cout << "元素 2 后插入 0: " << endl;
    list1.insertByBehindPosElem(2, 0);
    cout << "第二次打印: " << endl;
    list1.printList();
    cout << "反转链表: " << endl;
    list1.reverseList();
    cout << "第四次打印: " << endl;
    list1.printList();
    cout << "排序: " << endl;
    list1.sortList();
    cout << "第五次打印: " << endl;
    list1.printList();
    cout << "删除元素 2: " << endl;
    list1.deleteByElem(2);
    cout << "第六次打印: " << endl;
    list1.printList();
}
```

运行结果

```
尾部插入1,2,3,4:
头部插入9:
第一次打印:
9 1 2 3 4
1号位置前插入8:
4号位置后插入0:
元素4前插入8:
元素2后插入0:
第二次打印:
8 9 1 2 0 0 3 8 4
反转链表:
第四次打印:
4 8 3 0 0 2 1 9 8
排序:
第五次打印:
0 0 1 2 3 4 8 8 9
删除元素2:
第六次打印:
0 0 1 3 4 8 8 9
```

## 七、实验分析

（1）算法的性能分析

- 头部添加元素

  头部添加元素只需在头节点处进行操作，平均时间复杂度为$O(1)$

- 尾部添加元素

  尾部添加元素，需要先遍历到链表尾部，然后在尾部完成插入操作，平均时间复杂度为$O(n)$

- 指定位置前/后插入元素

  该操作需要先找到指定位置，再完成插入操作，时间复杂度最好为$O(1)$，最坏为$O(n)$，平均为$O(n)$

- 指定元素前/后插入元素

  该操作需要先找到指定元素，在找到的前提下，再完成插入操作，时间复杂度最好为$O(1)$，最坏为$O(n)$，平均为$O(n)$

- 指定位置删除元素

  该操作需要先找到指定位置，再完成删除操作，时间复杂度最好为$O(1)$，最坏为$O(n)$，平均为$O(n)$

- 删除指定元素

该操作需要先找到指定元素，在找到的前提下，再完成删除操作，时间复杂度最好为$O(1)$，最坏为$O(n)$，平均为$O(n)$

- 根据位置取元素

该操作需要遍历到找到指定位置，时间复杂度最好为$O(1)$，最坏为$O(n)$，平均为$O(n)$

- 查找元素位置

操作需要遍历查找，时间复杂度最好为$O(1)$，最坏为$O(n)$，平均为$O(n)$

- 排序

该排序使用链式结构的插入排序方法，时间复杂度为$O(n)$

- 反转

反转操作通过一次遍历实现链表的反转，时间复杂度为$O(n)$

（2）数据结构的分析

- 优点

根据上述分析可知，顺序表的存储特性，一块连续的存储区域意味着，通过一次计算便能访问到需要访问的元素，这种特性可以让存取元素变得快捷。同样地，只需要下标便可以轻松的维护整块存储区域。

- 缺点

每次给顺序表开辟一块连续的存储区域如果不能充分利用，就意味着有部分的空间浪费。另外，由于每个元素都是连续的，每次删除、插入等需要移动元素时，需要将部分元素一起移动，需要多次操作，在需要频繁插入，删除的场景中，顺序表操作较慢。

## 八、实验总结

本次实验通过使用 C++面向对象的特性实现出了链表的模板，对链表的基本操作进行了实现，体会线性表的运用和操作，其中对链表的排序和反转更加灵活的应用了线性表的链式存储结构的特征，理解数据结构的基础逻辑，看似繁琐的操作实际上是理解数据结构的必经步骤。

此外，本次实验依旧存在一定的不足，实现的单链表的操作受限，由一定的局限性，还有很大的改善空间，这些问题也将在后续实验中不断完善，不断优化，更加熟练掌握和运用数据结构。

## 参考文献：

[1]李春葆.《数据结构教程（第 6 版）》.北京：清华大学出版社.

[2]陈越,何钦铭.《数据结构（第 2 版）》.北京：高等教育出版社.

[3]殷人昆.《数据结构（用面向对象与 C++语言描述）（第 3 版）》.北京：清华大学出版社