2023—2024 学年第二学期

《数据结构与算法导论》实验报告

班级：_____

姓名：_____

学号：_____

班内序号：_____

报告日期：_____

数据结构实验报告

1. 实验要求

实验一：线性表的实现

要求：1、实现头插法尾插法两种方法。2、实现线性表的基本功能：如删除，查找，获取链表长度，销毁，编写 main 函数测试。

实验三：通讯录管理：

利用线性表实现通讯录管理，且数据格式如下：

- **struct DataType{**
  int ID; //编号
  char name[10]; //姓名
  char ch; //性别
  char phone[13]; //电话
  char addr[31]; //地址
  }

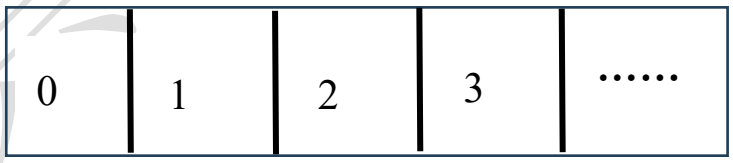
字体：汉字宋体、英文 Times New Roman
字号：五号
颜色：黑色
行距：单倍行距

2. 程序分析

2.1 存储结构

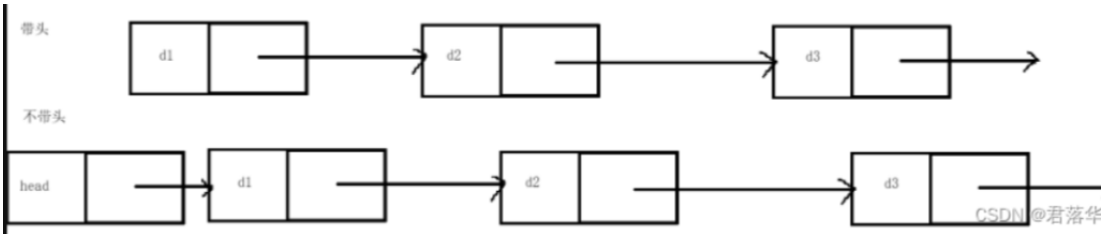
实验一所需要的是顺序表的存储结构：

示意图如下：



实验二所需要链表的存储结构：

示意图如下：

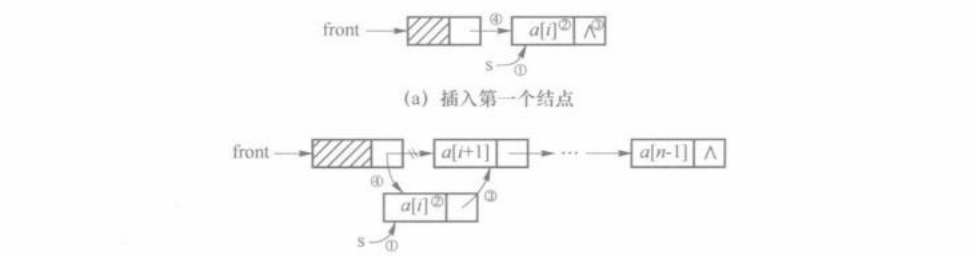


2.2 关键算法分析

实验一代码实现了一个链表数据结构，包括初始化、插入、删除、查找、获取长度和打印链表等操作。下面是关键算法和详细代码分析：

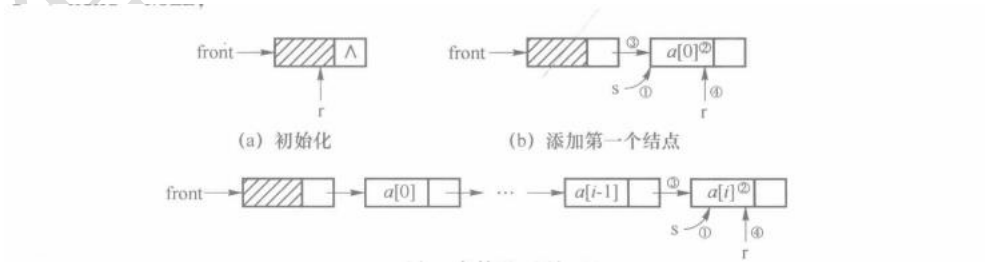
1. 初始化链表：通过传入一个数组和数组长度，创建一个带头结点的链表。

初始化链表(数组 a, 数组长度 n)
    创建头结点 front
    front.next = NULL
    遍历数组 a 的元素
        创建新结点 s
        s.data = a[i]
        s.next = front.next
        front.next = s
    返回链表



2. 尾插法创建链表：通过传入一个数组和数组长度，创建一个不带头结点的链表。

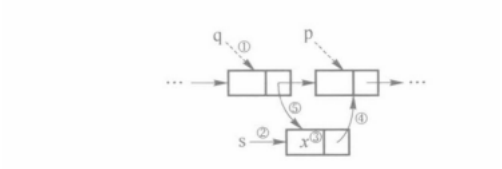
尾插法创建链表(数组 a, 数组长度 n)
    创建头结点 front
    遍历数组 a 的元素
        创建新结点 s
        s.data = a[i]
        s.next = NULL
        找到链表的尾结点 p
        p.next = s
    返回链表



3. 插入元素：在链表的第 n 个位置插入元素 x。

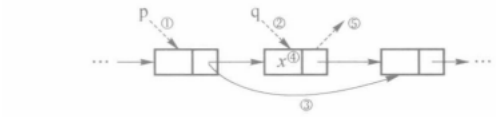
插入元素(位置 n, 元素 x)
    如果 n 小于等于 0 或大于链表长度 +1
        抛出异常 "插入错误"
    否则
        找到第 n-1 个结点 p
        创建新结点 s

s.data = x
s.next = p.next
p.next = s



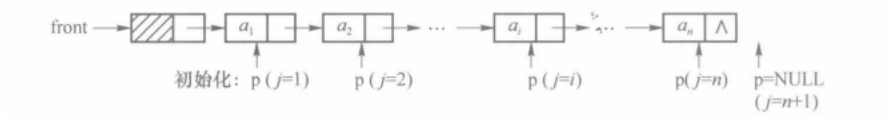
4. 删除元素：删除链表的第 n 个位置的元素。

删除元素(位置 n)
    如果 n 小于等于 0 或大于链表长度
        抛出异常 "位置错误"
    否则
        找到第 n-1 个结点 p
        找到第 n 个结点 q
        p.next = q.next
        删除结点 q



5. 查找元素：查找链表中值为 x 的元素的位置。

查找元素(值 x)
    创建新结点 p
    p = front
    初始化位置 i 为 0
    当 p 不为空时
        如果 p.data == x
            返回 i
        否则
            p = p.next
            i++
    如果遍历完链表仍未找到值为 x 的元素
        返回 -1



6. 获取链表长度：获取链表的长度。

获取链表长度()
    初始化长度 i 为 -1
    创建新结点 p
    p = front
    当 p 不为空时

```
        p = p.next
        i++
    返回 i
```

7. 打印链表：打印链表的所有元素。

```
打印链表()
    获取链表长度 n
    如果 n > 0
        输出 "front->"
        遍历链表的每个结点
            输出 p.data + "->"
        输出最后一个结点的数据
    否则 如果 n == -1
        输出 "链表已经析构"
    否则
        输出 "NULL"
```

时间空间复杂度如下：

1. 初始化链表：时间复杂度为 O(n)，空间复杂度为 O(n)。
2. 尾插法创建链表：时间复杂度为 O(n)，空间复杂度为 O(1)。
3. 插入元素：时间复杂度为 O(1)，空间复杂度为 O(1)。
4. 删除元素：时间复杂度为 O(1)，空间复杂度为 O(1)。
5. 查找元素：时间复杂度为 O(n)，空间复杂度为 O(1)。
6. 获取链表长度：时间复杂度为 O(n)，空间复杂度为 O(1)。
7. 打印链表：时间复杂度为 O(n)，空间复杂度为 O(1)

实验二的代码实现了一个电话簿管理系统，包括添加、删除、查询和定位功能。主要使用了线性表（SeqList）来存储和管理数据。

1. 定义了一个结构体 DataType，用于存储联系人的基本信息，包括 ID、姓名、性别、电话和地址。
2. 定义了一个类 PhoneBook，用于创建联系人对象，并提供了打印联系人信息的方法。
3. 定义了一个模板类 SeqList，用于实现线性表的基本操作，包括插入、删除、获取和定位元素。
4. 在 main 函数中，创建了两个联系人对象 book1 和 book2，并将它们添加到线性表 list 中。然后根据用户的输入执行相应的操作，如查询所有联系人、查询单个联系人、添加联系人、删除联系人和定位联系人。

伪代码如下：

```
定义结构体 DataType:
    ID: int
    姓名： string
    性别： char
    电话： string
    地址： string

定义类 PhoneBook:
```

构造函数(DataType X):
self.ID = X.ID
self.姓名 = X.姓名
self.性别 = X.性别
self.电话 = X.电话
self.地址 = X.地址
打印信息():
输出 self.ID, self.姓名， self.性别， self.电话， self.地址
判断相等(PhoneBook p):
如果 self.ID == p.ID:
返回 True
否则：
返回 False

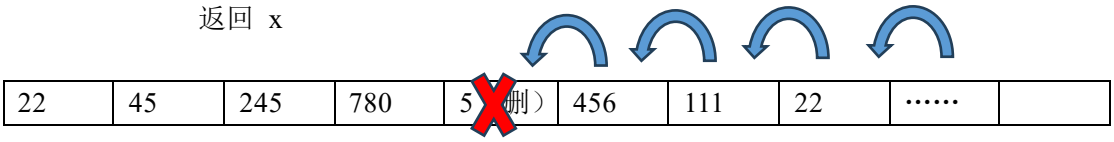
定义模板类 SeqList:
构造函数(T a[], int n):
如果 n > N:
抛出异常 "超出最大长度"
将 a 中的元素复制到 data 中，并设置 length 为 n
打印列表():
遍历 data，调用每个元素的 print() 方法
插入(int i, T x):
如果 length >= N:
抛出异常 "数组上溢"
如果 i < 1 或 i > length + 1:
抛出异常 "访问位置出错"
将 data[i-1] 之后的元素向后移动一位
将 x 赋值给 data[i-1]
length++

获取(int i):
如果 i < 1 或 i > length:
抛出异常 "数据上溢"
返回 data[i-1]

22	45	245	780	New	456	111	22	……	

删除(int i):
如果 i < 1 或 i > length:
抛出异常 "位置异常"
将 data[i-1] 的值保存到变量 x 中
将 data[i-1] 之后的元素向前移动一位
length--

返回 x



定位(T x):

遍历 data，如果找到与 x 相等的元素，返回其位置加一

返回 0

代码的时间空间复杂度如下：

1. 时间复杂度：
   - 插入操作：O(n)，因为在最坏的情况下，需要将 n 个元素向后移动一位。
   - 删除操作：O(n)，因为在最坏的情况下，需要将 n 个元素向前移动一位。
   - 获取操作：O(1)，因为直接通过索引访问数组元素。
   - 定位操作：O(n)，因为在最坏的情况下，需要遍历整个数组。
2. 空间复杂度：O(N)，其中 N 为线性表的最大长度。因为需要使用一个固定大小的数组来存储数据

2.3 其他

在写实验二的代码时，如何将 struct 的数组直接赋值给 class 类的数组，这一点我的代码中写的比较简单粗暴。还有在 locate 函数中，写的代码仍然有些啰嗦，先将值传入了 pp 然后再赋值给 struct 类，再转化成 PhoneBook 的类才进行的传入函数。

3. 程序运行结果

实验一主函数流程分析：

1. **初始化测试：**
   - 创建一个整型数组 **a**，包含了 10 个整数。
   - 使用 **LinkList<int>** 类型的对象 **test**，将数组 **a** 初始化为一个链表。
   - 打印初始化后的链表内容。
2. **空链表测试：**
   - 创建一个空的 **LinkList<int>** 类型的对象 **TestNull**。
   - 打印空链表的内容。
3. **GetLength 测试：**
   - 分别获取 **test** 和 **TestNull** 的长度，并输出结果。
4. **插入测试：**
   - 在 **test** 中的第 3 个位置插入值为 100 的节点，并打印链表内容。
   - 在 **TestNull** 中的第 1 个位置插入值为 10 的节点，并打印链表内容。
5. **删除测试：**

- 删除 **test** 中的第 3 个节点，并打印链表内容。

6. **查找测试：**
   - 查找 **test** 中值为 64 的节点的位置，并输出结果。
7. **尾插法测试：**
   - 使用尾插法构建一个新的链表 **test01**，并打印链表内容。
8. **析构测试：**
   - 测试 **test** 的析构函数，打印析构后的链表内容。

测试条件和输出：

1. 边界情况测试：
   - 在空链表中插入节点。
   - 在链表的首部插入节点。
   - 在链表的尾部插入节点。
   - 删除链表的头节点。
   - 删除链表的尾节点。
   - 查找链表中第一个节点和最后一个节点。
2. 异常情况测试：
   - 尝试在不存在的位置插入节点。
   - 尝试删除不存在的位置的节点。
   - 在空链表中进行节点的删除、查找等操作。
   - 在链表中查找一个不存在的元素。

预期输出：：

- 在空链表中插入节点：预期输出为插入后的链表。
- 删除链表的头节点：预期输出为删除头节点后的链表。
- 尝试删除不存在的位置的节点：预期输出为错误提示信息："位置错误"。
- 在链表中查找一个不存在的元素：预期输出为查找结果为-1。
-

```
初始化测试：front->10->45->33->22->49->4->1->38->64->91
边界情况测试：
在空链表中插入节点：front->100
在链表的首部插入节点：front->99->10->45->33->22->49->4->1->38->64->91
在链表的尾部插入节点：front->99->10->45->33->22->49->4->1->38->64->91->88
删除链表的头节点：front->10->45->33->22->49->4->1->38->64->91->88
删除链表的尾节点：front->10->45->33->22->49->4->1->38->64->91
查找链表中第一个节点和最后一个节点：第一个节点值为 10，最后一个节点值为 91
异常情况测试：
尝试在不存在的位置插入节点：错误提示：插入错误
尝试删除不存在的位置的节点：错误提示：越位！
在空链表中进行节点的删除、查找等操作：查找结果为：-1
在链表中查找一个不存在的元素：查找结果为：-1
Press any key to continue . . .
```

符合预期

实验二 main 函数流程分析：

1. **初始化通讯录列表和联系人信息：**
   - 创建一个包含两个联系人信息的数组 **init_book**，其中每个元素包括联系人

的 ID、姓名、性别、电话和地址。
- 使用 **init_book** 中的信息初始化两个 **PhoneBook** 对象 **book1** 和 **book2**。
- 将 **book1** 和 **book2** 放入一个 **PhoneBook** 数组 **Book** 中。
- 使用 **Book** 数组初始化一个 **SeqList<PhoneBook,100>** 对象 **list**，作为通讯录列表。

2. **打印通讯录列表：**
   - 调用 **list.PrintList()** 方法，打印初始化后的通讯录列表中的所有联系人信息。
3. **循环执行用户操作：**
   - 进入一个无限循环，直到用户选择退出程序。
   - 在每次循环中，用户可以选择执行查询、增加、删除或定位操作。
4. **查询功能：**
   - 如果用户选择查询，可以进一步选择查询所有联系人信息或查询单个联系人信息。
   - 如果选择查询单个联系人信息，则需要输入要查询的联系人的编号。
5. **增加功能：**
   - 如果用户选择增加，需要依次输入新联系人的 ID、姓名、性别、电话和地址信息。
   - 将新联系人信息封装为 **DataType** 类型，然后调用 **list.Insert()** 方法将其添加到通讯录列表中。
6. **删除功能：**
   - 如果用户选择删除，需要输入要删除的联系人在列表中的位置。
   - 调用 **list.Delete()** 方法删除指定位置的联系人信息，并将被删除的联系人信息打印出来。
7. **定位功能：**
   - 如果用户选择定位，需要输入要定位的联系人的 ID。
   - 创建一个临时的 **DataType** 对象，并将其 ID 设置为用户输入的 ID。
   - 调用 **list.Locate()** 方法定位指定联系人信息的位置，并将位置打印出来。
8. **异常处理：**
   - 在每个操作中，程序会进行一些异常处理，例如超出最大长度、数组上溢或访问位置出错等情况。
9. **循环结束：**
   - 用户选择退出程序后，跳出循环，程序结束执行。

测试条件和结果分析：

1. 初始化测试条件：
   - 创建包含两个联系人信息的数组 **init_book**，并用其初始化两个 **PhoneBook** 对象 **book1** 和 **book2**。
   - 将 **book1** 和 **book2** 放入一个 **PhoneBook** 数组 **Book** 中。
   - 使用 **Book** 数组初始化一个 **SeqList<PhoneBook,100>** 对象 **list**。
2. 打印列表测试：
   - 调用 **list.PrintList()** 打印通讯录列表。
3. 查询功能测试：
   - 查询所有联系人信息。
   - 查询单个联系人信息，包括输入编号查询单个联系人信息。
4. 增加功能测试：

- 添加新的联系人信息，包括输入新的联系人信息，然后调用 **list.Insert()** 添加到通讯录列表，并打印新增的联系人信息。

5. 删除功能测试：
   - 删除指定位置的联系人信息，包括输入要删除的位置，然后调用 **list.Delete()** 删除该位置的联系人信息，并打印被删除的联系人信息。
6. 定位功能测试：
   - 定位指定联系人信息的位置，包括输入要定位的联系人的 ID，然后调用 **list.Locate()** 定位该联系人信息的位置，并输出位置。

特殊情况测试条件：

- 超出最大长度的情况：尝试创建超过最大长度的 **SeqList** 对象。
- 数组上溢的情况：尝试向满员的 **SeqList** 对象插入新的元素。
- 访问位置出错的情况：尝试在不存在的位置访问 **SeqList** 对象中的元素。

```
联系人id: 22222 联系人姓名:Lily 联系人性别: M 联系人电话:  13001 联系人地址:  beiyou

**********通讯录**********
************************
*********输入1查询********
*********输入2增加********
*********输入3删除********
*********输入4定位********
1
*********输入1查询所有********
*********输入2查询单个********
1
按顺序遍历各个元素
联系人id: 11111 联系人姓名:Mary 联系人性别: F 联系人电话:  13000 联系人地址:  beiyou
联系人id: 22222 联系人姓名:Lily 联系人性别: M 联系人电话:  13001 联系人地址:  beiyou

**********通讯录**********
************************
*********输入1查询********
*********输入2增加********
*********输入3删除********
*********输入4定位********
1
*********输入1查询所有********
*********输入2查询单个********
3
**********通讯录**********
************************
*********输入1查询********
*********输入2增加********
*********输入3删除********
*********输入4定位********
1
*********输入1查询所有********
*********输入2查询单个********
2
输入编号:3
terminate called after throwing an instance of 'char const*'
PS D:\C++working>
```

上图为查询的测试图

```
2
输入id: 33333
输入姓名:lll
输入性别:F
输入电话: 130006
输入地址: beiyy
联系人id: 33333 联系人姓名:lll 联系人性别: F 联系人电话:  130006 联系人地址:  beiyy
**********通讯录**********
************************
*********输入1查询********
*********输入2增加********
*********输入3删除********
*********输入4定位********
1
*********输入1查询所有********
*********输入2查询单个********
1
按顺序遍历各个元素
联系人id: 11111 联系人姓名:Mary 联系人性别: F 联系人电话:  13000 联系人地址:  beiyou
联系人id: 22222 联系人姓名:Lily 联系人性别: M 联系人电话:  13001 联系人地址:  beiyou
联系人id: 33333 联系人姓名:lll 联系人性别: F 联系人电话:  130006 联系人地址:  beiyy

**********通讯录**********
************************
*********输入1查询********
*********输入2增加********
*********输入3删除********
*********输入4定位********
```

上图为增加的测试图

```
*********输入4定位********
3
删除的数字: 3
联系人id: 33333 联系人姓名:lll 联系人性别: F 联系人电话:  130006 联系人地址:  beiyy
**********通讯录**********
************************
*********输入1查询********
*********输入2增加********
*********输入3删除********
*********输入4定位********
3
删除的数字: 5
terminate called after throwing an instance of 'char const*'
PS D:\C++working>
```

删除功能示意图

```
*********输入4定位********
4
11111
位置是：1
**********通讯录**********
************************
*********输入1查询********
*********输入2增加********
*********输入3删除********
*********输入4定位********
4
55555
位置是：0
**********通讯录**********
************************
*********输入1查询********
*********输入2增加********
*********输入3删除********
*********输入4定位********
```

上图为测试定位功能

4. 总结

遇到的问题和解决方案：

1. 实验一中遇到最大的问题就是存在头结点，然后再 GetLength 时候，需要控制好计数器的起始位置，并且和 p 工作指针指向 front 还是 front->next 要搭配好。解决方案就是一步步进行代码调试，跟踪变量的变化，最后得到解决
2. 实验一中还有一个问题就是 GetLength 函数在遇到空链表时会返回-1，所以在打印链表那里要着重注意特殊情况，这里是加了一个 if 判断
3. 实验二中最大的问题是，在初始化时，想将一个 DataType 的结构体数组直接赋值给 PhoneBook 类里面的构造函数，从而构造出一个 PhoneBook 的数组，然后直接调用顺序表的初始化函数进行初始化。但是发现好像并没有办法实现，这里是采取了一种比较笨的方法，就是一步步自己初始化。
4. 实验二中 Locate 函数要求传入的是一个 PhoneBook 类，这里直接 cin 是没有办法解决的，只能是一步步进行封装，先 cin 一个 id 然后封装成结构体，再调用类的构造函数

心得体会：

1. 在编写链表类时，需要考虑各种边界情况和异常情况，确保程序的稳定性和健壮性。
2. 对于链表的操作，包括插入、删除、查找等，需要仔细考虑算法的实现和效率，尽量减少不必要的遍历和操作。
3. 在链表的析构函数中，要确保正确释放每个节点的内存，以避免内存泄漏问题。
4. 编写测试代码时，需要覆盖各种可能的情况，包括正常情况和异常情况，以保证程序的正确性和稳定性。