

“数据结构”

课程设计报告

**设计题目**  静态链表

**姓 名**  王竞翔

**学 号**  2023217273

**专 业**  计算机科学与技术

**班 级**  计算机科学与技术一班

**完成日期**  2024.6.13

课程设计成绩评定表

|  |  |  |  |
| --- | --- | --- | --- |
| **设计题目** | 静态链表 | **成绩** |  |
| **内容简介** | 实现了三个不同类型的静态链表测试：普通静态链表，单循环静态链表和双向循环链表。每个静态链表使用数组来模拟链表结构，并提供了各种操作函数，如初始化链表、获取长度、按位置获取元素、按值查找元素、插入元素、删除元素、尾插法创建链表、头插法创建链表、打印链表数据、打印空单元下标等。TestManager类用于测试普通静态链表，而C\_TestManager类用于测试单循环静态链表，D\_TestManager用于测试双循环链表。可以通过显示的菜单选择不同的功能进行测试，并通过输入选择相应的操作来对链表进行操作 | | |
| **评语** | **教师签名：** | | |

**（一） 需求和规格说明**

静态链表（Static Linked List）指利用数组实现链表的功能，免去了顺序表插入和删除操作时耗时的移动元素操作。是一个空间换时间的实例。在像JAVA、C#等没有指针的程序设计语言中，要实现链表就必须采用这种方式。下面以静态单链表为例来解释静态链表的实现：

申请一个较大的一维数组SL[n]。数组的元素由2个成员构成：一个成员data保存链表的数据元素；另一个成员next是指向链表中下一个元素的指针，事实上这里的next是存储下一个元素的数组下标。通过next，形成链式结构。当链表结束，最后一个结点的next域赋给一个特殊的值，比如：-1，表示链表结束。

与动态链表不同的是，插入新结点时，首先要判断表空间（数组）是否已满。如果表空间未满，则要获取一个空的单元，即取得空单元的指针（数组下标），那么如何取得空单元的指针呢？一般我们把所有空单元也组成一个链表，根据空单元链表的头指针很容易就能获取一个空单元。删除结点时，则需要把删除元素的单元回收到空单元链表中。所以，静态链表中事实上存储有2条链表，一条是保存数据的链表，另一条是空单元的链表。

静态链表也可以带有头结点，一般用数组SL[]头尾2个单元分别作为两条链表的头结点。比如，我们可以SL[0]为单链表头结点，头指针为0；以SL[n-1]为空单元链表的头结点，头指针为n-1，反之亦可。

下图所示为一个静态单链表，数据链表头结点为SL[0]，保存数据值a、b、c、d、e。空单元头结点为SL[11]。

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数组下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| data |  |  | **e** | **b** |  | **a** |  | **d** |  | **c** |  |  |
| next | **5** | **4** | **-1** | **9** | **6** | **3** | **-1** | **2** | **10** | **7** | **1** | **8** |

数据链表：

头指针

L

头结点

**a**

**3**

**b**

**9**

**0**

**d**

**2**

**e**

**-1**

**c**

**7**

**5**

空单元链表：

头指针

S

头结点

**10**

**1**

**11**

**6**

**-1**

**4**

**8**

**【功能要求—基本】**（65分）

1. 单链表初始化
2. 求链表长度
3. 按序号定位元素
4. 按给定值查找元素
5. 插入元素
6. 删除元素
7. 尾插法创建链表
8. 头插法创建链表
9. 打印数据链表
10. 打印空单元下标

**【功能要求—扩展1】**（+10分）

实现静态单循环链表，功能参照基本功能要求。

**【功能要求—扩展2】**（+10分）

实现静态双循环链表，功能参照基本功能要求。

**（二） 设计**

【静态链表的设计思路】

数据结构设计：

1.Node 类：

\_data: 用来存储数据。

\_cur: 用来存储下一个元素的索引，类似于指针的作用，在静态链表中扮演链表链接的角色。

2.StaticLinkedList 类：

SL: 一个固定大小的数组，用来存储 Node 对象，模拟链表的节点。

\_size: 当前链表中数据节点的数量。

功能实现：

1.单链表的初始化 (init\_list)：

初始化静态链表的空闲节点链，将除了最后一个外的所有节点通过 \_cur 链接起来，形成一个空闲链表。

最后一个节点的 \_cur 指向 MAX\_SIZE\_1（链表尾部），表示结束。

2.链表的基本操作：

插入 (insert)：在指定位置插入一个新节点，调整 \_cur 以保持链表结构。

删除 (delete\_elem)：删除指定位置的节点，并将其回收到空闲链表中。

按序号定位 (get\_elem)：根据序号（位置）找到链表中的元素。

按值查找 (search)：根据给定值在链表中查找元素的位置。

3.链表的创建：

尾插法 (push\_back)：用户输入一系列元素，使用尾插法将这些元素一一加入到链表中。

头插法 (push\_front)：用户输入一系列元素，使用头插法将这些元素一一加入到链表中。

4.辅助功能：

打印链表 (print\_list)：遍历链表，打印所有数据节点的值。

打印空闲节点 (print\_empty\_pos)：打印当前所有空闲节点的位置。

5.测试管理类（TestManager）

设计为单例模式，只能有一个 TestManager 实例。

包含一个静态方法 test，用于显示操作菜单，并接收用户输入执行相应的链表操作。

菜单设计包含了初始化、增删查等操作，用户通过输入选择对应的操作。

【静态循环链表的设计思路】

静态单循环链表的设计思路和静态链表几乎一致，同样的类设计，只不过在插入，删除等一些操作时需要加额外的边界判断。

【静态双循环链表的设计思路】

数据结构设计：

1.Node 类：

\_data: 用来存储数据。

\_cur: 用来存储下一个元素的索引，类似于指针的作用，在静态链表中扮演链表链接的角色。

\_pre: 用来存储前一个元素的索引，类似于指针的引用，在静态链表中扮演链表链接的角色。

2.StaticLinkedList 类：

SL: 一个固定大小的数组，用来存储 Node 对象，模拟链表的节点。

\_size: 当前链表中数据节点的数量。

功能实现：

1.单链表的初始化 (init\_list)：

初始化静态链表的空闲节点链，将除了最后一个外的所有节点通过 \_cur 链接起来，形成一个空闲链表。

最后一个节点的 \_cur 指向 MAX\_SIZE\_1（链表尾部），表示结束。

2.链表的基本操作：

插入 (insert)：在指定位置插入一个新节点，调整 \_cur 以保持链表结构。

删除 (delete\_elem)：删除指定位置的节点，并将其回收到空闲链表中。

按序号定位 (get\_elem)：根据序号（位置）找到链表中的元素。

按值查找 (search)：根据给定值在链表中查找元素的位置。

3.链表的创建：

尾插法 (push\_back)：用户输入一系列元素，使用尾插法将这些元素一一加入到链表中。

头插法 (push\_front)：用户输入一系列元素，使用头插法将这些元素一一加入到链表中。

4.辅助功能：

打印链表 (print\_list)：遍历链表，打印所有数据节点的值。

打印空闲节点 (print\_empty\_pos)：打印当前所有空闲节点的位置。

5.测试管理类（TestManager）

设计为单例模式，只能有一个 TestManager 实例。

包含一个静态方法 test，用于显示操作菜单，并接收用户输入执行相应的链表操作。

菜单设计包含了初始化、增删查等操作，用户通过输入选择对应的操作。

**系统类图:**

**单循环链表 双循环链表**

Node

\_data

\_cur

\_pre

Node

\_data

\_cur

StaticLinkedList

Node SL[MAX\_SIZE]

size\_t \_size

TestManager

test();

show\_menu();

init\_list();

get\_len();

get\_elem();

search();

insert();

delete\_elem();

push\_back();

push\_front();

print\_list();

print\_empty\_pos();

**（箭头表示包含，没有涉及到继承与多态）**

**属性和方法定义**

|  |  |  |  |
| --- | --- | --- | --- |
| Node | **成员** | **\_data** | **存放数据** |
| **\_cur** | **存放游标** |
| **StaticLinkedList** | **成员** | **Node[MAX\_SIZE]** | **存放节点** |
| **\_size** | **当前链表长度** |
| **TestManager** | **方法** | **void test();** | **程序的入口** |
| **void show\_menu();** | **展示菜单，让用户做出选择** |
| **int get\_elem()** | **给出下标，返回元素** |
| **size\_t search()** | **给出元素值，返回对应下表** |
| **void insert()** | **特定位置插入元素** |
| **void delete()** | **特定位置删除元素** |
| **void push\_back()** | **尾插法创建链表** |
| **void push\_front()** | **头插法创建链表** |
| **void print\_list0** | **打印数据元素** |
| **void print\_empty\_pos()** | **打印空下标单元** |

**（三） 用户手册**

程序运行时，用户输入对应的下标即可进入相关的操作，对于这三个静态链表，想测试哪个就调用哪个的Testmanager::test()函数即可。链表一定要初始化才可以使用，如果链表已经有元素了，则初始化链表会清空链表中的所有元素。

**（四） 调试及测试**

用户对静态链表进行操作时，都是对同一个链表操作，所以把链表的定义放在外面，保证用户调用的是同一个链表。测试了边界，但插入元素的位置或者删除元素的位置不合法时，会通过assert给出提示并终止程序，防止进一步变坏；还进行了集成测试，对于同一个链表，调用不同的方法，保证链表的通用性，不会导致只能使用其中一个功能的情况；也进行了回归测试，每加入一个新的代码，就会对之前所有的代码进行一次测试，防止出现任何异常；最后，进行了性能测试，比如求长度的算法，本来打算从头到尾依次遍历，但是时间复杂度为O(n)，如果在定义一个长度变量，不会增加太多的空间复杂度，反而降低了许多的时间复杂度。

**（五） 运行实例（只是运行部分，因为绝大多数需要与用户交互）：**

\* \* \* \* \* \* \* \* \* \* \* \*

\* 2023217273 王竞翔 \*

\* 普通静态链表测试 \*

\* \* \* \* \* \* \* \* \* \* \* \*

\* 1. 单链表初始化 \*

\* 2. 求链表长度 \*

\* 3. 按序号定位元素 \*

\* 4. 按给定值查找元素 \*

\* 5. 插入元素 \*

\* 6. 删除元素 \*

\* 7. 尾插法创建链表 \*

\* 8. 头插法创建链表 \*

\* 9. 打印数据链表 \*

\* 10.打印空单元下标 \*

\* 0. 退出 \*

\* \* \* \* \* \* \* \* \* \* \* \*

选择要测试的功能:9

[ 7 8 9 ]

**（六）进一步改进**

（1）这个静态链表只是实现了这10个功能，实际的链表不只有这些功能，还有反转链表，重新指定长度，拷贝构造函数，赋值构造函数，重载一些运算符，用模板实现泛型，封装迭代器类，实现迭代器功能，replace代替函数，链表的比较等等，都没有实现，以后可以加上。

（2）由于不会UI设计方面，所以只能能用控制台做出简易的用户交互界面，学了GUI设计，比如Qt后，可以适当的完善下界面，显得美观些。

（3）有些算法没有考虑到最好的时间复杂度和空间复杂度，可以借助STL中的容器，算法来完善算法的效率。

**（七）心得体会**

出现的问题有无法正确的认识到游标和指针的关联，以及如何初始化，更新游标等。也认识到了在其他没有指针的语言（Java，C#等），此种静态链表的优点（增删改查快捷），缺点（需要多出int[]数组长度存放游标，增大了空间复杂度）不过此种设计思路值得学习。

**（八）对课程设计的建议**

适当的修改下结构，不一定每个课程设计都需要用类的三大特性进行继承多态。比如静态结构，只是用到了封装，而这样的话，画类图就显得没太大的必要。可以有多种形式的展示方式，比如动画。

**（九）附录⎯⎯源程序**

