程序设计基础课程设计报告

——动态链表

高宇轩 23009200132

2024 年 4 月 23 日

1 原始题目及要求

链表是一种重要的数据结构，需要动态的进行存储分配，要求通过函数分别实现动态链表的建立、结点的插入、结点的删除以及链表的输出。

2 题目分析

2.1 题目功能

通过动态的存储分配实现链表。

链表能够被建立，能够在尾部插入元素，能够删除某一个元素，能够打印链表的内容，能够查找某个元素是否在链表中。

2.2 题目知识点

内存管理、结构体定义、指针运用、函数

3 题目总体方案设计

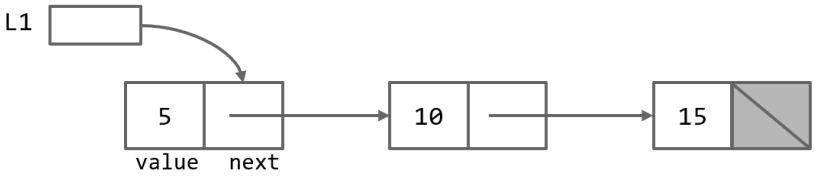

图 1: 链表示意图

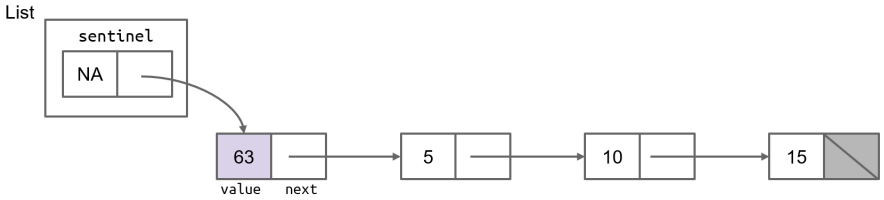

图 2: 增加哨兵节点后的链表示意图

4 各功能模块的设计说明

4.1 链表建立

基于我们的哨兵节点，建立链表的操作相当于初始化一个哨兵节点。

4.2 插入元素

插入元素，要先把插入节点的 next 成员设置为前一个结点原来的 next 成员，然后将前一个节点的 next 成员设置为新插入的节点。

4.3 删除元素

先找到要删除元素对应的节点，然后用一个游标跟踪前一个结点，将前一个节点的 next 成员直接设置为删除节点的 next，此时需要删除的节点就不在链表中了，同时要释放删除节点的内存，防止内存泄漏。

4.4 打印链表

遍历一遍链表，同时将每一个节点的 value 成员输出。

4.5 清除内存

遍历所有的节点，然后将每一个节点分配的内存释放，防止出现内存泄漏。

5 程序的集成测试

图 3 中的操作调用了链表的所有功能，通过图 4 中链表输出的结果，我们可以看出链表的操作满足了我们的要求。

```cpp
int main() {
    LinkedList lst;
    lst.append( val: 1);
    lst.append( val: 2);
    lst.append( val: 3);
    lst.append( val: 4);
    lst.print();

    std::cout << "The last one of LinkedList is" << lst.pop() << std::endl;
    lst.print();

    std::cout << "Remove 2 from the LinkedList: " << lst.remove( val: 2) << std::endl;
    lst.print();
    std::cout << "Remove 4 from the LinkedList: " << lst.remove( val: 4) << std::endl;
    lst.print();

    std::cout << "The LinkedList contains 1: " << lst.contains( val: 1) << std::endl;
    std::cout << "The LinkedList contains 2: " << lst.contains( val: 2) << std::endl;

    lst.clear();
    std::cout << "After clear:";
    lst.print();

    return 0;
}
```

图 3: 程序的集成测试内容

```
Ubuntu: '/mnt/e/202402-202406/C++Programs/23009200132/2LinkedList/cmake-build-debug/LinkedList'
[1, 2, 3, 4]
The last one of LinkedList is 4
[1, 2, 3]
Remove 2 from the LinkedList: 1
[1, 3]
Remove 4 from the LinkedList: 0
[1, 3]
The LinkedList contains 1: 1
The LinkedList contains 2: 0
After clear:[]
```

图 4: 程序的集成测试结果

6 总结

根据程序的集成测试，可以看出我们的链表操作符合程序的需求。

另外，为了提高链表的效率，我们还可以设置尾指针作为标识，提高查询，插入或删除尾部元素的操作。此时我们可以类比头部的哨兵节点，设置尾部的哨兵节点来减少空链表情况下的讨论，或者采用循环链表的方式进行简化。