# 计算机学院 高级语言程序设计 课程实验报告

| | | |
|---|---|---|
| 实验题目：线性群体类模板编程 | | 学号：202200400053 |
| 日期：2024-05-23 | 班级： 2202 | 姓名： 王宇涵 |
| Email：1941497679@qq.com | | |

**实验目的：**

1. 练习线性群体类模板：结点、链表类模板；
2. 练习操作受限的线性群体：栈、队列

**实验软件和硬件环境：**

软件环境 ：VSCODE + DEV-C++

硬件环境 ：Legion Y7000P

**实验步骤与内容：**

**1. 实现第 9 章实验 9，实验任务（1），实现 PPT 例 9-5 结点类。lab9_1.cpp**

（1）编写程序 Node.h 实现例 9-5 的结点类，并编写测试程序 lab9_1.cpp 实现链表的基本操作。

测试:

```
// 创建一些节点
Node<int> node1(1); // 第一个节点，数据为1
Node<int> node2(2); // 第二个节点，数据为2
Node<int> node3(3); // 第三个节点，数据为3

// 将节点插入到链表中
node1.insertAfter(&node2); // 在node1之后插入node2
node2.insertAfter(&node3); // 在node2之后插入node3

// 遍历链表并打印每个节点的数据
Node<int> *current = &node1;
while (current != nullptr) {
    std::cout << current->data << " ";
    current = current->nextNode();
}
std::cout << std::endl;

// 删除node1之后的节点（即删除node2）
Node<int> *deletedNode = node1.deleteAfter();

// 再次遍历链表并打印每个节点的数据
current = &node1;
while (current != nullptr) {
    std::cout << current->data << " ";
    current = current->nextNode();
}
std::cout << std::endl;
```

输出结果：

```
1 2 3
1 3
```

**2. 实现第 9 章实验 9，实验任务（2），利用“实验 10 附件”中的 link.h，实现 PPT 例 9-6 的链表类。lab9_2.cpp**

**问 link.h 的实现代码中有 bug 吗？参加附件中的 testbug.cpp,如何修改错误？**

（2）编写程序 link. h 实现例 9-6 的链表类。在测试程序 lab9_2. cpp 中声明两个整型链表 A 和 B，分别插入 5 个元素，然后把 B 中的元素加入 A 的尾部。

Lab9_2 输入

```
A.insertRear(1);
A.insertRear(2);
A.insertRear(3);
A.insertRear(4);
A.insertRear(5);
```

```
B.insertRear(6);
B.insertRear(7);
B.insertRear(8);
B.insertRear(9);
B.insertRear(10);
```

输出

```
PS D:\BaiduSyncdisk\CLASSES\C++\exp\exp10\实验十代码\output> & .\'lab9_2.exe'
Elements of linked list A after appending B: 1 2 3 4 5 6 7 8 9 10
```

答 : link.h 修改后输出如下:

```
insertFront Bug
List: 6  7
insertAfter Bug
List: 6  7  8
deleteCurrent Bug
没有节点可以删除
List: 6  7  8
```

修改错误方法

```
template <class T>
void LinkedList<T>::insertFront(const T& item) {
    //    Node<T>* n = new Node<T>(item, front);
    Node<T>* n = newNode(item, front->next);
//    front = n;
    front -> next = n;
    if(prevPtr == front)  //如果当前节点的前节点是头结点front
        prevPtr = n;      //当前节点的前节点应该是新插入的头结点
    size ++;
    position ++;
}
```

因为头结点是不存储数值的，则我们将值插入头结点之后，且可能更新 prevPtr 指针

```
//在当前节点之后插入节点
template <class T>
void LinkedList<T>::insertAfter(const T& item) {
    if (currPtr == rear) {
        insertAt(item);
    }
    Node<T>* temp = new Node<T>(item, currPtr->next);
    currPtr->next = temp;
    size ++;
    Node<T>* temp = new Node<T>(item, NULL);
    temp->next = currPtr->next;
    currPtr->next = temp;
    size ++;
}
```

如果 currPtr 指向尾结点，则我们在尾结点之前插入值

```
//删除当前节点
template <class T>
void LinkedList<T>::deleteCurrent() {
    if (currPtr == rear || currPtr == front) {
        cout << "没有节点可以删除" << endl;
        return;
    }
    Node<T>* temp = currPtr;
    currPtr = currPtr->next;
    delete temp;
    size --;
}
```

如果 currPtr 指向头结点或尾结点，则我们返回报错信息，无法删除节点

**3. 实现第 9 章实验 9，实验任务（3），用链表实现队列，完成 lab9_3.cpp**

设计为

```
// 入队
void enqueue(const T &item) {
    list.insertRear(item);
}

// 出队
T dequeue() {
    if (isEmpty()) {
        cerr << "Queue is empty, cannot dequeue!" << endl;
        exit(1);
    }
    return list.deleteFront();
}

// 获取队头元素
T& front() {
    if (isEmpty()) {
        cerr << "Queue is empty, cannot access front!" << endl;
        exit(1);
    }
    list.reset(0);
    return list.data();
}
```

测试输入

```
q.enqueue(1);
q.enqueue(2);
q.enqueue(3);
q.enqueue(4);
q.enqueue(5);
```

测试输出

```
Elements dequeued from the queue: 1 2 3 4 5
```

**4. 实践第 9 章 PPT，例 9-8、9-9，利用栈实现计算器。(算式采用后缀输入法，每个操作数、操作符之间以空白符分隔。)**

**你能否课本图 9-6、图 9-7 所示的表达式算法？（正常序）**

实现一个简单的整数计算器，能够进行加、减、乘、除和乘方运算。使用时算式采用后缀输入法，每个操作数、操作符之间都以空白符分隔。例如，若要计算"3+5"则输入"3 5 +"。乘方运算符用"^"表示。每次运算在前次结果基础上进行，若要将前次运算结果清除，可键入"c"。当键入"q"时程序结束。

后缀输入测试输入和输出

```
4 5 +
= 9
4 2 ^
= 16
5 19 + 2 -
= 24 = 22
6 3 /
= 2
```

**完成正常序计算器**

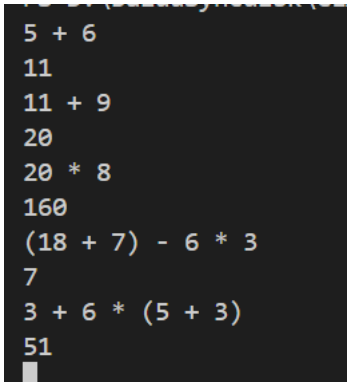
实现

```
stack<int>num;
stack<char>op;
map<char, int>pr = {
        {'+', 1},
        {'-', 1},
        {'*', 2},
        {'/', 2}};

void eval() {
    int b = num.top(); num.pop();
    int a = num.top(); num.pop();
    char c = op.top(); op.pop();
    int x;
    if (c == '+') x = a + b;
    else if (c == '-') x = a - b;
    else if (c == '*') x = a * b;
    else x = a / b;
    num.push(x);
}
```

```cpp
for (int i = 0; i < s.size(); i++) {
    auto c = s[i];
    if (c == '(') {
        op.push(c);
    }
    else if (isdigit(c)) {
        int j = i, x = 0;
        while (j < s.size() && isdigit(s[j])) {
            x = x * 10 + s[j++] - '0';
        }
        i = j - 1;
        num. push(x);
    }
    else if(c == ')') {
        while (op.top() != '(') eval();
        op.pop();
    }
    else if (c == '+' || c == '-' || c == '*' || c == '/'){
        // 如果当前操作符的优先级小于等于栈顶操作符的优先级，则先计算栈顶操作符
        while (op.size() && pr[c] <= pr[op.top()] ) eval();
        op.push(c);
    }
}
```

测试输入输出




**结论分析与体会：**

在完成第 9 章实验后，我收获颇多。

在实验任务（1）中，我实现了结点类，深刻理解了面向对象编程的基本概念和如何在 C++中定义和使用类。

任务（2）中，通过实现链表类，我学会了如何使用指针操作动态内存以及管理链表结构。特别是在分析 link.h 代码中的 bug 时，我进一步掌握了调试技巧和代码审查能力。

任务（3）中，用链表实现队列，增强了我对数据结构的理解，尤其是队列的基本操作和链表的结合使用。

最后，在实践利用栈实现计算器的过程中，我不仅复习了栈的基本操作，还学会了如何将中缀表达式转换为后缀表达式并进行计算，熟练掌握了图 9-6、图 9-7 所示的表达式算法。

总的来说，通过这些实验，我不仅练习了线性群体类模板：结点、链表类模板，而且练习了操作受限的线性群体：栈、队列，还提升了编程和问题解决的能力。

就实验过程中遇到的问题及解决处理方法，自拟 1－3 道问答题：

**1.链表有无头结点有什么区别**

答 ：链表有头结点和没有头结点的主要区别在于管理链表结构的简便性和操作的实现复杂度。有头结点的链表在链表的开头有一个特殊的节点，该节点不存储有效数据，仅用于简化链表的操作，比如插入和删除操作。这样，在进行插入或删除时，无需特殊处理空链表或修改头指针，代码实现更为简洁和一致。

**2.栈和队列的作用？**

答 :栈是一种后进先出（LIFO, Last In First Out）的结构，常用于需要逆序处理数据的场景，如函数调用管理、表达式求值、语法解析和深度优先搜索等。

队列则是一种先进先出（FIFO, First In First Out）的结构，适用于按顺序处理数据的场景，如任务调度、广度优先搜索（BFS）、消息队列和缓冲区管理等。