实验二 线性表的链式存储系统维护

一、实验目的

1.掌握线性表的链式存储的定义和基本使用方法。

2.掌握线性表的链式存储存储单元的排列特点。

3.掌握线性表的链式存储系统的建立、遍历、插入、查找 、删除操作，学会相关的函数定义和调用。

二、实验内容

1.建立一个链表。

2.能够对建立的链表进行查找、修改、插入、删除等操作。当输入指令错误时，能够提示错误信息。主函数中可以选择由 switch\case 语句构成主菜单，再根据提示进行相应操作。

三、实验指导

1.以循环的方式建立一个有表头的链表；

2.遍历链表，并计算链表结点个数；

3.在查找功能中要实现：当能找到时打印该值的前驱结点，找不到时输出“没找到”；

4.在插入功能中要实现：在某个特定的结点之后插入一个结点；

5.在删除功能主要实现：删除特定值的结点，注意区分该节点是否为链表结尾。

四、代码实现

```
//Experiment 2: Linked List by:Yang Yujie using C++
#include <iostream>
using namespace std;

typedef struct LNode{
    int value;  //头结点L->value 的值表示单链表中的元素个数；其他结点 LNode->value 的值表示单链表结点中存储的值。
    struct LNode* next;
}LNode, *LinkedList;

bool InitializeList (LinkedList &L);//初始化链表，以固定格式输入建立链表。
bool ShowLinkedList (LinkedList L);//遍历链表并打印。
bool InsertElement (LinkedList &L, int InsertedElement, int GoalElem);//插入操作。
bool LocateElement (LinkedList L, int GoalElem, int &Location);//定位操作，通过引用返回目标元素的标号。
bool ModifyElement (LinkedList &L, int FormerElem, int NewElem);//修改操作。
bool DeleteElement (LinkedList &L, int &Location, int ElementDelete);//删除操作，通过引用返回被删除元素的标号。

//元素的位置标号从 1 开始。
int main() {
    int command;
    LinkedList test;
    cout << "Experiment 2: Linked List." << endl
         << "<Instruction> Please initialize the linked list." << endl;
```

```
InitializeList(test);
ShowLinkedList(test);

cout << endl
     << "<Instruction> Please type in the command number to operate:"
     << endl << endl;
cout << "/* The command corresponds to operations.\n"
       " * command -1:Terminate the program.\n"
       " * command 1 :LocateElement.\n"
       " * command 2 :ModifyElement.\n"
       " * command 3 :InsertElement.\n"
       " * command 4 :DeleteElement.\n"
       " */" << endl;

while (cin >> command) {
    switch (command) {
       case 1: {
          int GoalElem, Location;
          bool flag1 = true;
          while (flag1) {
             cout << "/* LocateElement */" << endl;
             ShowLinkedList(test);
             cout << "\n<Instruction> Please type in the located element:";
             cin >> GoalElem;
             if (LocateElement(test, GoalElem, Location)) {
                flag1 = false;
             } else {
                flag1 = true;
             }
          }
       }  break;

       case 2: {
          int FormerElem, NewElem;
          bool flag2 = true;
          while (flag2) {
             cout << "/* ModifyElement */" << endl;
             ShowLinkedList(test);
             cout << "\n<Instruction> Please type in the former element:";
             cin >> FormerElem;
             cout << "<Instruction> Please type in the new element:";
             cin >> NewElem;
             if (ModifyElement(test, FormerElem, NewElem)) {
                ShowLinkedList(test);
                flag2 = false;
             } else {
                flag2 = true;
             }
          }
```

```
    }   break;

    case 3: {
        int InsertedElement, GoalElem;
        bool flag3 = true;
        while (flag3) {
            cout << "/* InsertElement */" << endl;
            ShowLinkedList(test);
            cout << "\n<Instruction> Please type in the inserted element:";
            cin >> InsertedElement;
            cout << "<Instruction> Please type in the located element:";
            cin >> GoalElem;
            if (InsertElement(test, InsertedElement, GoalElem)) {
                ShowLinkedList(test);
                flag3 = false;
            } else {
                flag3 = true;
            }
        }
    }   break;

    case 4: {
        int LocationDelete = 1;
        bool flag4 = true;
        int ElementDelete;
        while (flag4) {
            cout << "/* DeleteElement */" << endl;
            ShowLinkedList(test);
            cout << "\n<Instruction> Please type in the Element to delete:";
            cin >> ElementDelete;
            if (DeleteElement(test, LocationDelete, ElementDelete)) {
                ShowLinkedList(test);
                flag4 = false;
            } else {
                flag4 = true;
            }
        }
    }   break;

    case -1: {
        cout << " <Instruction> The program terminated! " << endl;
    }   break;

    default: {
        cout << "The command is invalid!" << endl;
    }   break;

}
if (command == -1) {
```

```
            break;
        }
        cout << endl
             << "<Instruction> Please type in the command number to operate."
             << endl << endl;
        cout << "/* The command corresponds to operations.\n"
                " * command -1:Terminate the program.\n"
                " * command 1 :LocateElement.\n"
                " * command 2 :ModifyElement.\n"
                " * command 3 :InsertElement.\n"
                " * command 4 :DeleteElement.\n"
                " */" << endl;
    }
    return 0;
}

bool InitializeList (LinkedList &L) {
    L = new LNode;
    if(!L){
        cout << "<InitializeList>\nThe Initialization goes WRONG!" << endl;
        return false;
    }else {
        L->next = nullptr;
        L->value = 0;
        int tempch;
        LNode* WorkPtr = L;
        cout << "<InitializeList>\nThe input format: a1 a2 a3 ... an\\n" << endl;
        while (true) {
            LNode* NewNode = new LNode;
            NewNode->next = nullptr;
            WorkPtr->next = NewNode;
            cin >> NewNode->value;
            L->value++;
            WorkPtr = WorkPtr->next;
            tempch = getchar();
            if (tempch == '\n') {
                break;
            }
        }
        cout << "\n<InitializeList>\nThe LinkedList has been initialized!" << endl;
        return true;
    }
}

bool ShowLinkedList (LinkedList L) {
    if(L == nullptr) {
        return false;
    }
    LNode* WorkPtr = L->next;
```

```
    cout << "<ShowLinkedList>\n";
    cout << "*** Numbers of elements: " << L->value << endl;
    cout << "*** LinkedList: " << endl;
    while (WorkPtr) {
        cout << WorkPtr->value;
        if (WorkPtr->next) {
            cout << " -> ";
        }
        WorkPtr = WorkPtr->next;
    }
    cout << '\n';
    return true;
}

bool InsertElement (LinkedList &L, int InsertedElement, int GoalElem) {
    LNode* WorkPtr = L->next;
    while ((WorkPtr) && (WorkPtr->value != GoalElem)) {
        WorkPtr = WorkPtr->next;
    }
    if (!WorkPtr) {
        cout << "<InsertElement>\nGoal Element Not Found." << endl;
        return false;
    }
    LNode* NewNode = new LNode;
    NewNode->value = InsertedElement;
    NewNode->next = WorkPtr->next;
    WorkPtr->next = NewNode;
    L->value++;
    cout << "<InsertElement>\nOperating Success!" << endl;
    return true;
}

bool LocateElement (LinkedList L, int GoalElem, int &Location) {
    LNode* WorkPtr = L->next, *FormerNode = L; int i = 1;
    while ((WorkPtr) && (WorkPtr->value != GoalElem)) {
        i++;
        WorkPtr = WorkPtr->next;
        FormerNode = FormerNode->next;
    }
    if (i > L->value) {
        cout << "<LocateElement>\nGoal Element Not Found." << endl;
        return false;
    } else {
        Location = i;
        cout << "<LocateElement>\nOperating Success!" << endl;
        cout << "*** GoalElem at the Location: "
        << Location << endl;
        cout << "*** Former Node: "
        << FormerNode->value << endl;
```

```
        return true;
    }
}

bool ModifyElement (LinkedList &L, int FormerElem, int NewElem) {
    LNode* WorkPtr = L->next;
    while ((WorkPtr) && (WorkPtr->value != FormerElem)) {
        WorkPtr = WorkPtr->next;
        }
    if (!WorkPtr) {
        cout << "<ModifyElement>\nOperating Failed." << endl;
        return false;
    }
    if (WorkPtr->value == FormerElem) {
        WorkPtr->value = NewElem;
        cout << "<ModifyElement>\nOperating Success!" << endl;
        cout << "The Former Node <"
            << FormerElem << "> has been modified to <"
            << NewElem << ">." << endl;
        return true;
    }
    return false;
}

bool DeleteElement (LinkedList &L, int &Location, int ElementDelete) {
    LNode* WorkPtr = L->next, *FormerNode = L;  Location = 1;
    bool IsTailNode = false;
    while ((WorkPtr) && (WorkPtr->value != ElementDelete)) {
        WorkPtr = WorkPtr->next;
        FormerNode = FormerNode->next;
        Location++;
    }
    if (!WorkPtr) {
        cout << "<DeleteElement>\nOperating Failed." << endl << endl;
        return false;
    }
    if (WorkPtr->value == ElementDelete) {
        if (!WorkPtr->next) {
            IsTailNode = true;
        }
        LNode *DeletedNode = WorkPtr;
        FormerNode->next = WorkPtr->next;
        delete DeletedNode;
        L->value--;
        cout << "<DeleteElement>\nOperating Success!" << endl;
        cout << "The node <"
        << ElementDelete << "> at the location <"
        << Location << "> has been deleted." << endl;
        if (IsTailNode) {
```

```
            cout << "The deleted node is at the tail." << endl;
        } else {
            cout << "The deleted node is not at the tail." << endl;
        }
            return true;
    }
    return false;
}
```

五、程序调试

```
Experiment 2: Linked List.
<Instruction> Please initialize the linked list.
<InitializeList>
The input format: a1 a2 a3 ... an\n
1 2 3 4 5

<InitializeList>
The LinkedList has been initialized!
<ShowLinkedList>
*** Numbers of elements: 5
*** LinkedList:
1 -> 2 -> 3 -> 4 -> 5
```

图 1 单链表初始化

```
<Instruction> Please type in the command number to operate:

/* The command corresponds to operations.
 * command -1:Terminate the program.
 * command 1 :LocateElement.
 * command 2 :ModifyElement.
 * command 3 :InsertElement.
 * command 4 :DeleteElement.
 */
-1
 <Instruction> The program terminated!

进程已结束，退出代码为 0
```

图 2 程序退出

```
/* LocateElement */
<ShowLinkedList>
*** Numbers of elements: 5
*** LinkedList:
1 -> 2 -> 3 -> 4 -> 5

<Instruction> Please type in the located element:3
<LocateElement>
Operating Success!
*** GoalElem at the Location: 3
*** Former Node: 2
```

图 3 单链表查找成功

```
/* LocateElement */
<ShowLinkedList>
*** Numbers of elements: 5
*** LinkedList:
1 -> 2 -> 3 -> 4 -> 5

<Instruction> Please type in the located element:6
<LocateElement>
Goal Element Not Found.
```

图 4 单链表查找失败

```
/* ModifyElement */
<ShowLinkedList>
*** Numbers of elements: 5
*** LinkedList:
1 -> 2 -> 3 -> 4 -> 5

<Instruction> Please type in the former element:4
<Instruction> Please type in the new element:100
<ModifyElement>
Operating Success!
The Former Node <4> has been modified to <100>.
<ShowLinkedList>
*** Numbers of elements: 5
*** LinkedList:
1 -> 2 -> 3 -> 100 -> 5
```

图 5 单链表修改成功

```
/* ModifyElement */
<ShowLinkedList>
*** Numbers of elements: 5
*** LinkedList:
1 -> 2 -> 3 -> 4 -> 5

<Instruction> Please type in the former element:6
<Instruction> Please type in the new element:100
<ModifyElement>
Operating Failed.
```

图 6 单链表修改失败

```
/* InsertElement */
<ShowLinkedList>
*** Numbers of elements: 5
*** LinkedList:
1 -> 2 -> 3 -> 4 -> 5

<Instruction> Please type in the inserted element:100
<Instruction> Please type in the located element:4
<InsertElement>
Operating Success!
<ShowLinkedList>
*** Numbers of elements: 6
*** LinkedList:
1 -> 2 -> 3 -> 4 -> 100 -> 5
```

图 7 单链表插入成功

```
/* InsertElement */
<ShowLinkedList>
*** Numbers of elements: 5
*** LinkedList:
1 -> 2 -> 3 -> 4 -> 5

<Instruction> Please type in the inserted element:100
<Instruction> Please type in the located element:6
<InsertElement>
Goal Element Not Found.
```

图 8 单链表插入失败

```
/* DeleteElement */
<ShowLinkedList>
*** Numbers of elements: 5
*** LinkedList:
1 -> 2 -> 3 -> 4 -> 5

<Instruction> Please type in the Element to delete:5
<DeleteElement>
Operating Success!
The node <5> at the location <5> has been deleted.
The deleted node is at the tail.
<ShowLinkedList>
*** Numbers of elements: 4
*** LinkedList:
1 -> 2 -> 3 -> 4
```

图 9 单链表删除成功（附带删除尾结点提示）

```
/* DeleteElement */
<ShowLinkedList>
*** Numbers of elements: 5
*** LinkedList:
1 -> 2 -> 3 -> 4 -> 5

<Instruction> Please type in the Element to delete:6
<DeleteElement>
Operating Failed.
```

图 10 单链表删除失败