1 附录 B 基于链式存储结构线性表实现的源程序

/* Linear Table On Sequence Structure */

main.cpp

```cpp
#include "def.h"
#include "LinkList.h"
#include "Management.h"
#include "command_op.h"

int main()
{
    cout << "Initializing..." << endl;
    Manager MainList;
    cout << "An empty list has been constructed." << endl;
    cout << "Use the following command to operate the list." << endl;
    int op, state;
    int flag = 1;
    main_menu();
    cout << "enter your command:" << endl;
    cin >> op;
    while (op) {
        switch (op) {
            case 1:
            state = MainList.NewList();
            if (state) {
                cout << "Complete. now the structure is:" << endl;
                structure_disp(MainList);
            }
            else {
                cout << "Memory error!" << endl;
                flag = 0;
            }
            break;
            case 2:
            int index;
            cout << "enter the num of the list:" << endl;
            cin >> index;
            state = MainList.DelList(index);
            if (state == OK) {
                cout << "delete successfully. Now the structure is:" <<
                endl;
                structure_disp(MainList);
            }
            else if (state == INFEASIBLE) {
                cout << "empty table!" << endl;
            }
            else {
                cout << "index out of range." << endl;
                //flag = 0;
            }
            break;
            case 3:
            /*单独使用一个函数来处理单链表操作*/
            state = single_list_op(MainList);
            if (state == ERROR) cout << "index out of range" << endl;
            cout << "now the structure is:" << endl;
            structure_disp(MainList);
            break;
            case 4:
            structure_disp(MainList);
            break;
            default:
            cout << "wrong command!" << endl;
        }
        if (flag == 0) break;
        main_menu();
        cout << "enter your command:" << endl;
        cin >> op;
    }
    cout << "bye." << endl;
    return 0;
}
```

def.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>
#include "stdio.h"
```

```
#include "stdlib.h"
using namespace std;
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
typedef int status;
typedef int ElemType; //数据元素类型定义
#define LIST_INIT_SIZE 100
#define LISTINCREMENT 10
typedef int ElemType;
typedef struct LNode { //单链表（链式结构）结点的定义
    ElemType data;
    struct LNode* next;
}LNode, * LinkList;
```

LinkList.h

```
#pragma once
#include "def.h"

class List {
    private:
    int length;
    LinkList head; //空表头
    LinkList tail; //指向尾部的指针方便操作

    public:
    List* down; //为线性表管理做准备，采用十字链表存储

    List();

    //获取头指针
    LinkList GetHead();

    //初始化表
    status InitList();

    //为表添加数据
    status AddData();
```

```
//销毁线性表
status DestroyList();

//清空线性表
status ClearList();

//线性表判空
status ListEmpty();

//返回线性表长度
int ListLength();

//获取第i个元素，保存在e中
status GetElem(int i, ElemType& e);

//查找元素e的位置序号
status LocateElem(ElemType e);

//获取线性表L中元素e的前驱，保存在pre中
status PriorElem(ElemType e, ElemType& pre);

//获取线性表L元素e的后继，保存在next中
status NextElem(ElemType e, ElemType& next);

//将元素e插入到线性表L的第i个元素之前
status ListInsert(int i, ElemType e);

//删除线性表L的第i个元素，并保存在e中
status ListDelete(int i, ElemType& e);

//遍历单链表
status ListTraverse();

//以文件形式保存表
status SaveList(char FileName[]);

//从文件加载表
status LoadList(char FileName[]);

//反转链表
```

```
    status reverse();

    //删除倒数第n个节点
    status del_n();

    //链表排序
    status SortList();
};

LinkList _reverse(LinkList l, LinkList r);
```

LinkList.cpp

```
#include "LinkList.h"

List::List() {
    this->head = nullptr;
    this->tail = nullptr;
    this->length = 0;
    this->down = nullptr;
}

LinkList List::GetHead() {
    return this->head;
}

status List::AddData() {
    if (!this->head) return INFEASIBLE;
    cout << "enter a series of numbers sperated thit a space and end
      with 0" << endl;
    int num;
    cin >> num;
    while (num) {
        this->tail->next = (LinkList)malloc(sizeof(struct LNode));
        this->tail = this->tail->next;
        this->tail->next = nullptr;
        this->tail->data = num;
        this->length++;
        cin >> num;
    }
    cout << "Add successfully." << endl;
    return OK;
```

```
}

status List::InitList() {
    if (this->head) return INFEASIBLE;
    this->head = (LinkList)malloc(sizeof(struct LNode));
    this->head->next = nullptr;
    this->tail = head;
    return OK;
}

status List::DestroyList() {
    if (!this->head) return INFEASIBLE;
    LinkList p;
    LinkList L = this->head;
    while (L) {
        p = L;
        L = L->next;
        free(p);
    }
    this->head = nullptr;
    this->tail = nullptr;
    this->length = 0;
    return OK;
}

status List::ClearList() {
    if (!this->head) return INFEASIBLE;
    LinkList p;
    LinkList L0 = this->head->next;
    while (L0) {
        p = L0;
        L0 = L0->next;
        free(p);
    }
    this->head->next = nullptr;
    this->length = 0;
    return OK;
}

status List::ListEmpty() {
    if (!this->head) return INFEASIBLE;
    if (this->length == 0) return TRUE;
    return FALSE;
}

int List::ListLength() {
    if (!this->head) return INFEASIBLE;
    return this->length;
}

status List::GetElem(int i, ElemType& e) {
    if (this->head == nullptr) return INFEASIBLE;
    int cnt = 1;
    LinkList L = this->head->next;
    while (L) {
        if (cnt == i) {
            e = L->data;
            return OK;
        }
        L = L->next;
        cnt++;
    }
    return ERROR;
}

status List::LocateElem(ElemType e) {
    if (this->head == nullptr) return INFEASIBLE;
    int cnt = 1;
    LinkList L = this->head->next;
    while (L) {
        if (L->data == e) return cnt;
        L = L->next;
        cnt++;
    }
    return ERROR;
}

status List::PriorElem(ElemType e, ElemType& pre) {
    if (this->head == nullptr) return INFEASIBLE;
    LinkList L = this->head;
    if (L->next == nullptr || L->next->data == e) return ERROR;
    L = L->next;
    while (L) {
        if (L->next == nullptr) return ERROR;
        if (L->next->data == e) {
            pre = L->data;
            return OK;
        }
        L = L->next;
    }
}

status List::NextElem(ElemType e, ElemType& next) {
    if (!this->head) return INFEASIBLE;
    LinkList L = this->head;
    if (!L->next || !L->next->next) return ERROR;
    L = L->next;
    while (L) {
        if (L->data == e) {
            if (L->next) {
                next = L->next->data;
                return OK;
            }
            else return ERROR;
        }
        L = L->next;
    }
    return ERROR;
}

status List::ListInsert(int i, ElemType e) {
    LinkList L = this->head;
    if (!this->head) return INFEASIBLE;
    int cnt = 1;
    while (L->next) {
        if (cnt == i) {
            if (L->next == nullptr) {
                L->next = new (struct LNode);
                L->next->data = e;
                L->next->next = nullptr;
                this->length++;
                //L = L0;
                return OK;
            }
            LinkList new_node = new (struct LNode);
            new_node->data = e;
            new_node->next = L->next;
            L->next = new_node;
            //L = L0;
            this->length++;
            return OK;
        }
        L = L->next;
        cnt++;
    }
    return ERROR;
}

status List::ListDelete(int i, ElemType& e) {
    if (!this->head) return INFEASIBLE;
    LinkList L = this->head;
    int cnt = 1;
    while (L->next) {
        if (cnt == i) {
            e = L->next->data;
            LinkList cell = L->next;
            L->next = L->next->next;
            free(cell);
            //L = L0;
            this->length--;
            return OK;
        }
        cnt++;
        L = L->next;
    }
    //L = L0;
    return ERROR;
}

status List::ListTraverse() {
    if (!this->head) return INFEASIBLE;
    LinkList L = this->head;
    if (L->next == nullptr) return OK;
    L = L->next;
    while (L) {
        printf("%d ", L->data);
        L = L->next;
    }
    return OK;
}

status List::SaveList(char FileName[]) {
    if (!this->head) return INFEASIBLE;
    FILE* fp = fopen(FileName, "w");
    LinkList L = this->head->next;
    while (L) {
        fprintf(fp, "%d ", L->data);
        L = L->next;
    }
    fclose(fp);
    return OK;
}

status List::LoadList(char FileName[]) {
    if (this->head) return INFEASIBLE;
    this->head = new (struct LNode);
    LinkList L0 = this->head;
    FILE* fp = fopen(FileName, "r");
    int num;
    while (fscanf(fp, "%d", &num) != EOF) {
        L0->next = new (struct LNode);
        L0 = L0->next;
        L0->data = num;
        this->length++;
    }
    L0->next = nullptr;
    fclose(fp);
    return OK;
}

status List::reverse() {
    if (!this->head) return INFEASIBLE;
    if (this->length <= 1) return OK;
    LinkList pre = this->head->next, cur = pre->next, next0 = cur->next;
    while (cur) {
        cur->next = pre;
        pre = cur;
        cur = next0;
        if (next0) next0 = next0->next;
    }
    this->head->next->next = nullptr;
    this->head->next = pre;
    return OK;
}

status List::del_n() {
    cout << "enter the n:" << endl;
    int n, e;
    cin >> n;
    n = this->length - n + 1;
    return this->ListDelete(n, e);
}

status List::SortList() {
    if (!this->head) return INFEASIBLE;
    if (this->length <= 1) return OK;

    vector<int> arr;
    LinkList cur = this->head->next;
    while (cur) {
        arr.push_back(cur->data);
        cur = cur->next;
    }
    cur = this->head->next;
    sort(arr.begin(), arr.end());
    for (int i = 0; i < arr.size(); i++) {
        cur->data = arr[i];
        cur = cur->next;
    }
    return OK;
}

LinkList _reverse(LinkList l, LinkList r) {
    if (l->next == r) {
        r->next = l;
        return l;
    }
    l = _reverse(l->next, r)->next;
    return l;
}
```