实现链表的各种基本运算

姓名：凌强 班级：21计算机二班 学号：2021413579

实验目的：

1. 掌握链表的基本运算。
2. 掌握顺序存储的概念，学会对顺序存储数据结构进行操作。
3. 加深对顺序存储数据结构的理解，逐步培养解决实际问题的编程能力。

实验内容：

1. 初始化链表，接收输入的数据，将以链表方式存储。
2. 通过输入的数字序号选择对应的功能以完成链表的插入、删除、查找、销毁等操作。
3. 采用头插法和尾插法两种创建链表的方式进行实验。

# **需求分析**

1. 完成对链表的创建、销毁、插入、删除、定位、输出等基本操作。
2. 演示程序中，规定 ElemType 为 int 型。
3. 演示程序以用户和计算机的对话方式执行，在运行框中出现提示信息之后，由用户在键盘上输入完成相关命令的执行；相应结果显示在其后。
4. 程序执行的命令包括：
   1. 输出用户界面。
   2. 读取用户输入的数据，通过头插法和尾插法分别创建两个链表。
   3. 输入数字选择对应的功能。
   4. 求线性表中第index个元素的值。
   5. 求L中第一个数据为item的位序。
   6. 在链表的第n个位置插入数据 m。
   7. 删除链表中第n个数据。
   8. 销毁链表。
   9. 退出程序。
5. 测试数据：见（[六、测试结果](#_测试结果)）。

# **概要设计**

1. 链表表的抽象数据定义：

typedef struct LList

{

    int data;

    struct LList \*next;

} LinkList;

1. 基本操作：

*//头插法输入数据*

LinkList \*HeadCreate(int *n*)

*//尾插法输入数据*

LinkList \*EndCreate(int *n*)

*//定位元素*

void Locate(LinkList \**p*)

*//插入元素*

void Insert(LinkList \**p*)

*//删除元素*

void Delete(LinkList \**p*)

*//查找元素*

void Find(LinkList \**p*)

*//输出数据*

void OutData(LinkList \**p*)

*//打印主界面*

void welcome(int \**n*)

1. 程序包含两个模块：
2. 主函数模块：

用于与测试用户交换数据。

1. 操作函数模块：

用于指引测试用户下一步操作，操作链表。

# **详细设计**

## 链表的定义

*#include* <iostream>

using namespace std;

typedef struct LList

{

    int data;

    struct LList \*next;

} LinkList;

## 操作函数

LinkList \*HeadCreate(int *n*){

    LinkList \*head, \*p;

    int i;

    head = NULL;

    cout<<"请输入"<<*n*<<"个数据："<<endl;

*for* (i = 0; i < *n*; i++)

    {

        p = (LinkList \*)malloc(sizeof(LinkList));

        cin >> p->data;

        p->next = head;

        head = p;

*// p->next = NULL;*

    }

    cout << "创建成功！" << endl;

*return* head;

}

LinkList \*EndCreate(int *n*){

    LinkList \*end, \*p,\*head = NULL;

    int i;

    cout<<"请输入"<<*n*<<"个数据："<<endl;

*for* (i = 0; i < *n*; i++)

    {

        p = (LinkList \*)malloc(sizeof(LinkList));

        cin >> p->data;

*if* (i == 0)

        {

            end = p;

            end->next = NULL;

            head = end;

        }

*else*

        {

            end->next = p;

            p->next = NULL;

            end = p;

        }

    }

    cout << "创建成功！" << endl;

*return* head;

}

void OutData(LinkList \**p*)

{

*if* (!*p*)

    {

        cout << "list is empty!";

    }

*while* (*p*)

    {

        cout << *p*->data << " ";

*p* = *p*->next;

    }

}

*//定位*

void Locate(LinkList \**p*)

{

    int n,i;

    LinkList \*h;

    h = *p*;

    cout<<"请输入要定位的元素索引:"<<endl;

    cin>>n;

*if*(!*p*)

    {

        cout << "list is empty!";

*return* ;

    }

*for*(i = 0;i<n-1;i++) h = h->next;

    cout<<"第"<<n<<"个元素为:"<<endl<<h->data<<endl;

    OutData(*p*);

}

*//查找元素*

void Find(LinkList \**p*)

{

    int n,i = 1;

    LinkList \*h;

    h = *p*;

    cout<<"请输入要查找的元素:"<<endl;

    cin>>n;

*if*(!*p*){

        cout << "list is empty!";

*return*;

    }

*while*(h->data != n)

    {

        h = h->next;

        i++;

    }

    cout<<"元素在第"<<i<<"个位置"<<endl;

    OutData(*p*);

}

*//插入元素*

void Insert(LinkList \**p*)

{

    int m,n,i = 1;

    LinkList \*h,\*L;

    h = *p*;

    L = (LinkList \*)malloc(sizeof(LinkList));

    cout<<"请输入要插入的元素和插入位置:"<<endl;

    cin>>L->data>>n;

*if*(!*p*){

        cout << "list is empty!";

*return*;

    }

*while*(i != n-1)

    {

        h = h->next;

    }

    L->next = h->next;

    h->next = L;

    cout<<"插入后的数据为:"<<endl;

    OutData(*p*);

}

*//删除元素*

void Delete(LinkList \**p*)

{

    int n;

    LinkList \*p1,\*p2;

    cout<<"请输入要删除的元素:"<<endl;

    cin>>n;

*if*(!*p*){

        cout << "list is empty!";

*return*;

    }

*if*(*p*->data == n)

    {

        p2=*p*;

*p*=*p*->next;

        free(p2);

    }

    p1=*p*;

    p2=*p*->next;

*while*(p2!=NULL)

    {

*if*(p2->data == n)

        {

            p1->next=p2->next;

            free(p2);

            p2=p1->next;

        }

*else*

        {

            p1=p2;

            p2=p2->next;

        }

    }

    OutData(*p*);

}

*//打印主界面*

void welcome(int \**n*)

{

    system("cls");

    printf("〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓〓  ☆     链 表 测 试 系 统      ☆  〓〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓★★★★★         ★★★★★★★         ★★★★★〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓★★★★★         ★★★★★★★         ★★★★★〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          1.定位元素        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          2.插入元素        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          3.删除元素        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          4.查找元素        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          5.输出元素        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆          0.退出系统        ☆  ★〓〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓★  ☆                              ☆  ★〓〓〓〓〓〓〓〓\n");

    printf("〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓〓\n");

    cout << "初始化链表成功！" << endl;

    cout<<"请输入要输入元素的个数:"<<endl;

    cin>>\**n*;

}

## 主函数完善

int main()

{

    LinkList \*L1;

    LinkList \*L2;

    int n;

    welcome(&n);

*// L1 = HeadCreate(n);*

    L2 = EndCreate(n);

    int op;

*while* (true)

    {

*//指引用户操作*

        printf("\n请输入对应的功能键(数字): ");

        cin >> op;

*switch* (op)

        {

*case* 1:

            Locate(L2);

*break*;

*case* 2:

            Insert(L2);

*break*;

*case* 3:

            Delete(L2);

*break*;

*case* 4:

            Find(L2);

*break*;

*case* 5:

            cout<<"当前数据为:"<<endl;

            OutData(L2);

*break*;

*default*:

*break*;

        }

    }

    system("pause");

*return* 0;

}

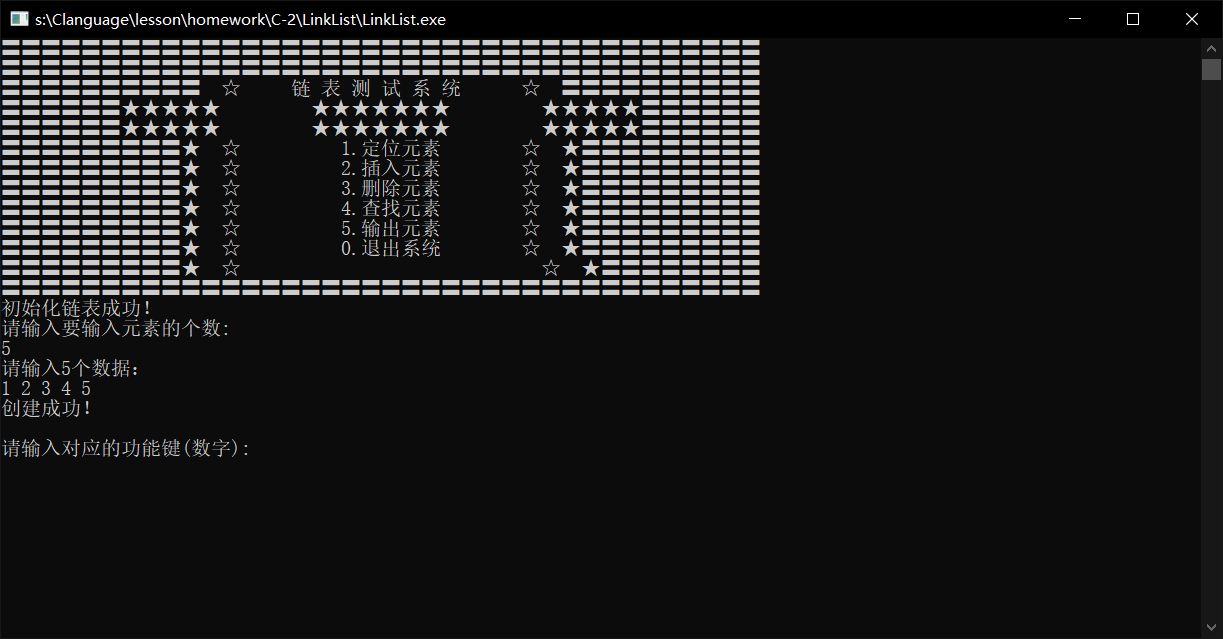
# **调试分析**

1. 模块划分较合理。
2. 本次测试全程使用无头节点的链表，无论是头插法还是尾插法，都没有定义头节点。
3. 顺序表示在删除，插入操作中的时间复杂度高于链式。O（n）
4. 定位、求前驱、求后继由于都要寻找e的位置，时间复杂度相当。
5. 遍历的时间复杂度均为O(n)。
6. 本次实验报告完成了对链表的各种基本运算表示，是一次良好的程序设计训练。

# **用户使用说明**

程序运行环境为vscode,执行文件为LinkList.cpp

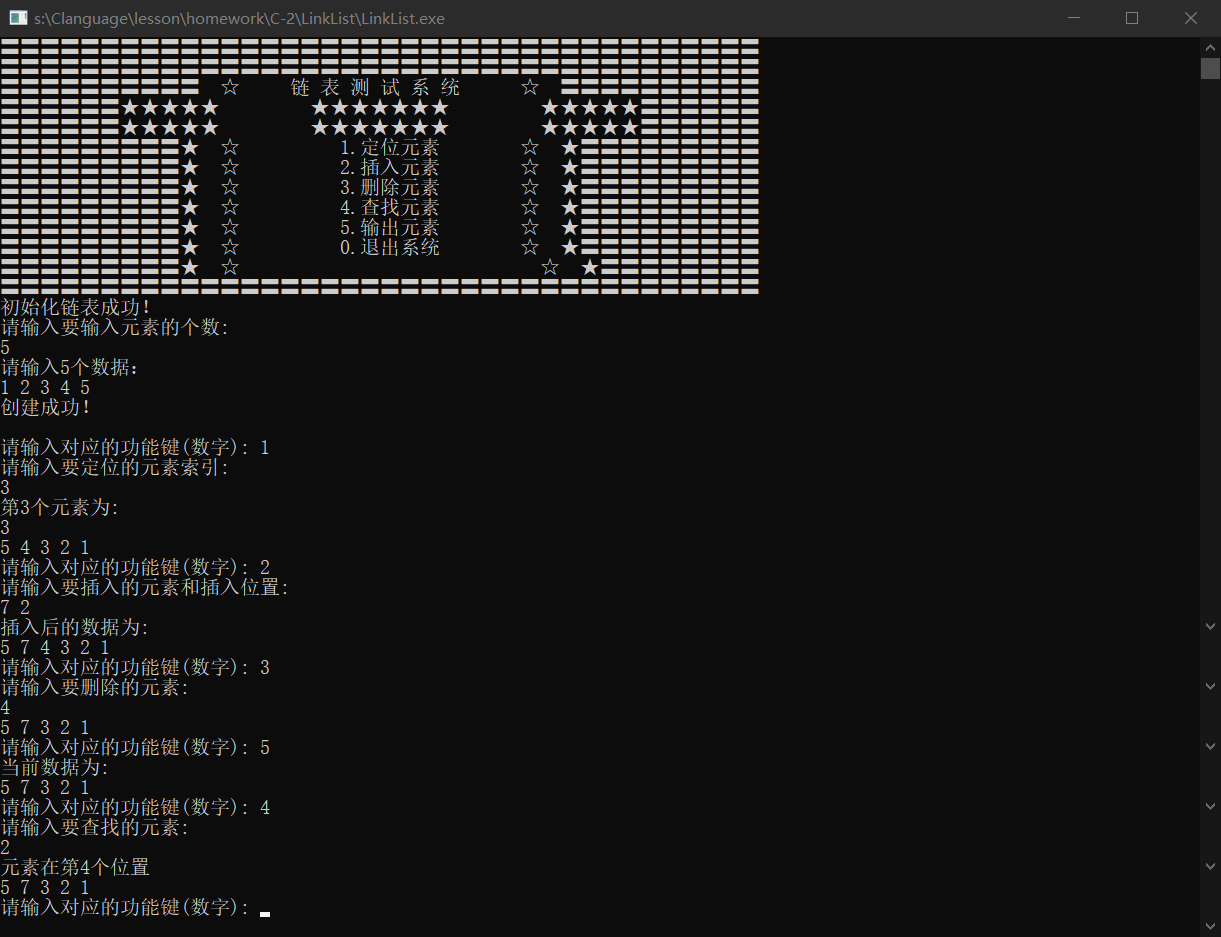
用户界面：



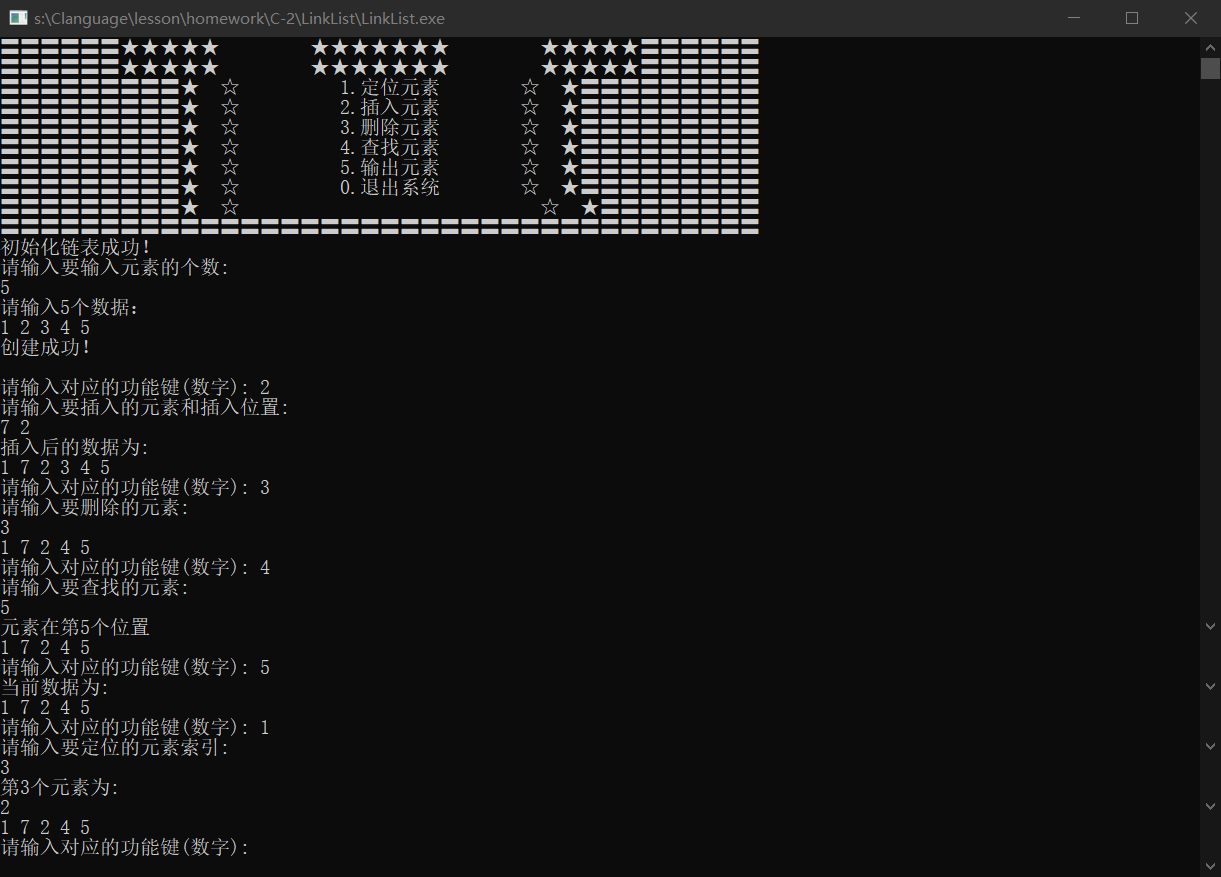
先输入元素个数，然后依次输入数据，最后创建成功，选择对应的功能。

# **测试结果**

## 链表（头插法）



## 链表（尾插法）



# **实验体会**

通过两次实验的测试，学习了顺序表和链表的定义和各种基本操作，并且了解到了各自的优缺点和其对应的应用领域，大致总结一下吧：

1. 顺序表：

顺序表的特点是逻辑上相邻的数据元素，物理存储位置也相邻，并且顺序表的存储空间需要预先分配。

* 1. 优点：

1. 方法简单，各种高级语言中都有数组，容易实现。
2. 不用为表示节点间的逻辑关系而增加额外的存储开销。
3. 顺序表具有按元素序号随机访问的特点。
   1. 缺点：
4. 在顺序表中做插入、删除操作时，平均移动表中的一半元素，因此n较大的顺序表效率低。
5. 需要预先分配足够大的空间，预先分配过大，可能会导致顺序表后部大量闲置；预先分配过小，又会造成溢出。
6. 链表：

在链表中逻辑上相邻的数据元素，物理存储位置不一定相邻，它使用指针实现元素之间的逻辑关系。并且，链表的存储空间是动态分配的。

1. 优点：
2. 插入、删除运算方便。
3. 缺点：
4. 要占用额外的存储空间存储元素之间的关系，存储密度降低。存储密度是指一个节点中数据元素所占的存储单元和整个节点所占的存储单元之比。
5. 链表不是一种随机存储结构，不能随机存取元素。
6. 通过索引查找元素效率低。

# **附录**

源程序：

LinkList.cpp