**链表实验报告**

姓名：陆明奇 学号：2050283 日期：2021年10月16日

**1. 实验内容**

**1.1 问题描述**

实现链表的相关操作。

**1.2 基本要求**

(1) 链表的基本操作；

(2) 链表的逆置；

(3) 链表的去重；

(4) 合并两个有序链表；

**1.3功能说明（函数、类）**

Node\* ListCreate(**int** n)

{

 Node\* head = **new** Node;

 Node\* p = head;

 //利用尾插法创建一个链表

**for**(**int** i=1;i<=n;i++) {

  Node\* q = **new** Node;

  p->next = q;

  cin >> q->data;

  p = p->next;

 }

 p->next = **NULL**;

**return** head;

}

功能：建立一个长度为n的链表；

算法复杂度：O(n);

**void** ListDisplay(Node\* head)

{

 Node\* p = head;

 p = p->next;

 cout << "链表中的元素为";

**while** (p) {

  cout << p->data << ' ';

  p=p->next;

 }

 cout << endl;

}

功能：输出一个长度为n的链表；

算法复杂度：O(n);

**bool** ListInsert(Node\* head, **int** i, **int** data)

{

 //在带头结点的单链线性表中第i个位置之前插入元素data

 Node\* p = head;

**int** j = 0;

**while** (p && j < i - 1) {//寻找第i-1个结点

  p = p->next;

  j++;

 }

**if** (p == **NULL** || j >= i - 1) {//i小于1或者大于表长加1

  cout << "插入失败" << endl;

**return** **false**;

 }

 Node\* s = **new** Node;//生成新结点

 s->data = data;//插入链表中

 s->next = p->next;

 p->next = s;

**return** **true**;

}

功能：在带头结点的单链线性表中第i个位置之前插入元素data；

算法复杂度：O(n);

**bool** ListDelete(Node\* head, **int** i, **int**& data)

{

 //在带头结点的单链线性表中删除第i个元素,并由data返回其值

 Node\* p = head;

**int** j = 0;

**while** (p->next && j < i - 1) {//寻找第i个结点，并令p指向其前驱

  p = p->next;

  j++;

 }

**if** (p->next == **NULL** || j >= i - 1) {

  cout << "删除位置不合理" << endl;

**return** **false**;

 }

 Node\* q = p->next;//删除并释放结点

 p->next = q->next;

 data = q->data;

 free(q);

**return** **true**;

}

功能：在带头结点的单链线性表中删除第i个元素,并由data返回其值；

算法复杂度：O(n);

**bool** ListGetElem(Node\* head, **int** i, **int**& data)

{

 //当第i个元素存在时，其值赋给data

 Node\* p = head->next;//初始化，p指向第一个结点

**int** j = 1;//计数器

**while** (p && j < i) {//顺指针向后查找，直到p指向第i个元素或p为空

  p = p->next;

  j++;

 }

**if** (p == **NULL** || j >= i) {

  cout << "第" << i << "个元素不存在" << endl;

**return** **false**;

 }

 data = p->data;//取第i个元素

**return** **true**;

}

功能：当第i个元素存在时，其值赋给data；

算法复杂度：O(n);

**void** ListReverse(Node\* head)

{

 Node\* p;

 p = head->next;

 head->next = **NULL**;

**while** (p)

 {//头插法

  Node\* q = p;

  p = p->next;

  q->next = head->next;

  head->next = q;

 }

}

功能：将链表逆置；

算法复杂度：O(n);

**void** ListDeleteDuplication(Node\* head)

{

    Node\* cur = head;

    Node\* pre = **NULL**;

    Node\* next = **NULL**;

**while** (**NULL** != cur) {//针对每一个元素查找所有与其值相等的元素

        pre = cur;

        next = cur->next;

**while** (**NULL** != next) {

**if** (cur->data == next->data) {

                pre->next = next->next;

                free(next);//释放重复元素空间

            }

**else** {

                pre = next;

            }

            next = next->next;

        }

        cur = cur->next;

    }

}

功能：链表的去重

算法复杂度：O(n^2);

**void** ListMerge (Node\* &La,Node\* &Lb,Node\* &Lc)

{

    //单链线性表La和Lb的元素按值非递减排列，归并得到的新单链线性表Lc的元素也按值非递减排列

    Node\* pa=La->next;

    Node\* pb=Lb->next;

    Node\* pc=La;

    Lc=pc;//用La的头结点作为Lc的头结点

**while**(pa&&pb){

**if**(pa->data<=pb->data){

            pc->next=pa;

            pc=pa;

            pa=pa->next;

        }

**else**{

            pc->next=pb;

            pc=pb;

            pb=pb->next;

        }

    }

    pc->next=pa?pa:pb;//插入剩余段

    free(Lb);//释放Lb的头结点

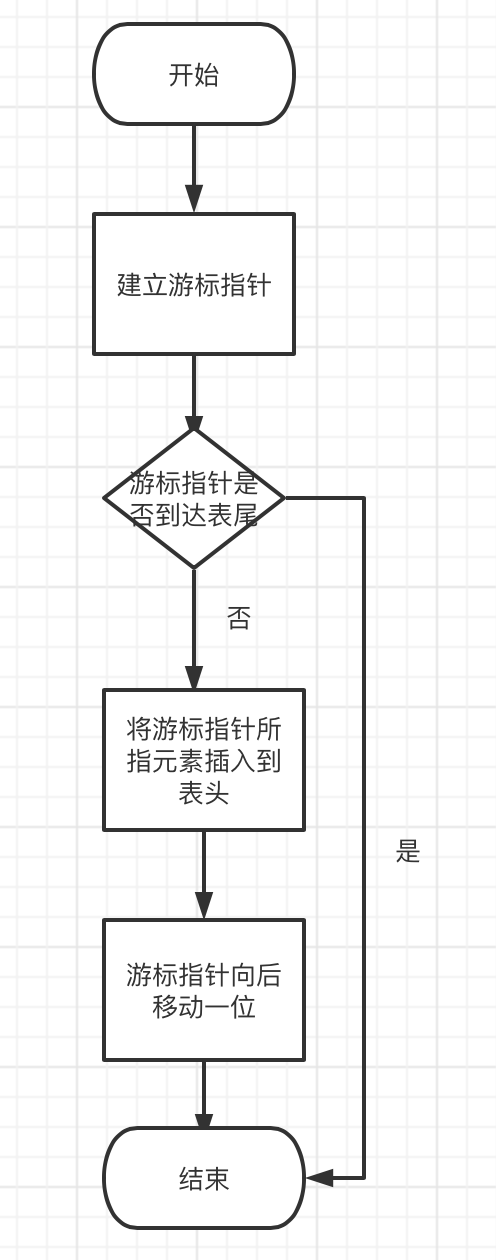
}

功能：合并两个有序链表；

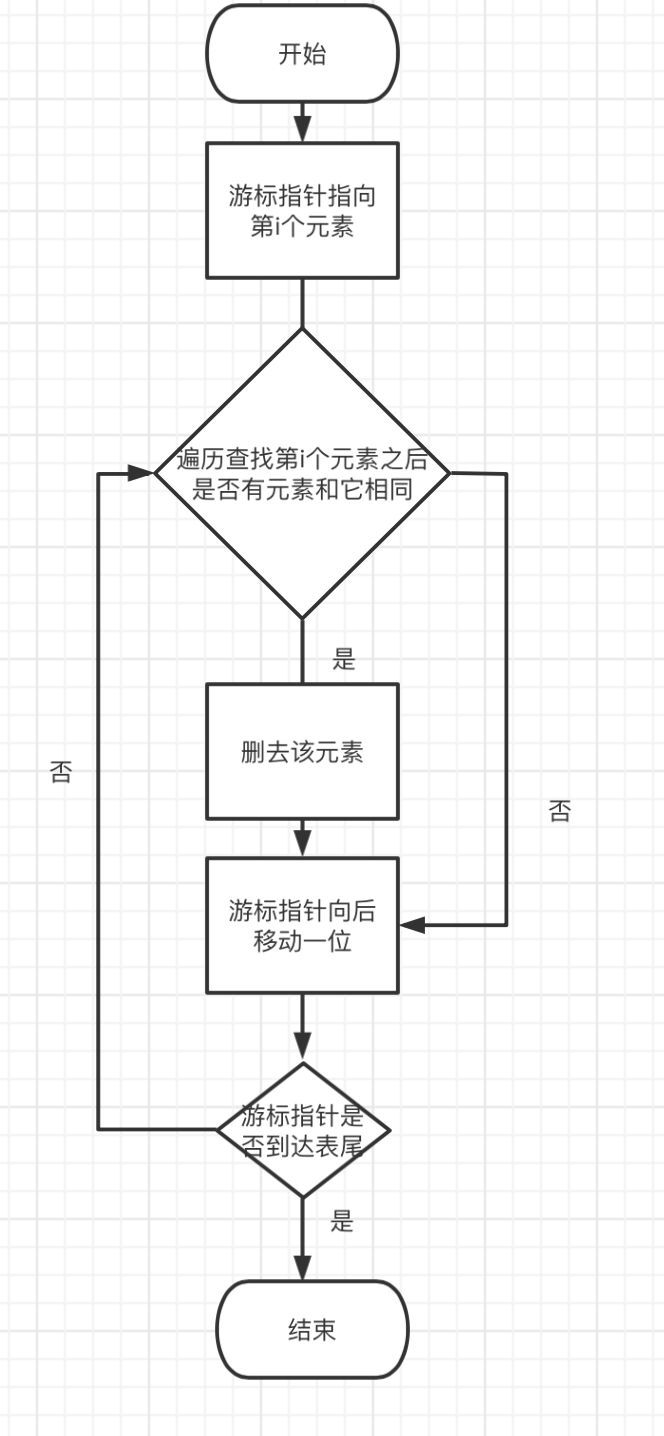
算法复杂度：O(n);

**1.4流程图及算法复杂度分析**

链表的逆置：采用头插法进行逆置，从头结点开始遍历整个链表，每次将元素插入到表头，完成链表的逆置，因为需要从表头遍历一次链表，算法复杂度为O(n);



链表的去重：针对本问题，我采用两层循环，即针对每一个元素查找所有与其值相等的元素，然后删除即可，因为存在嵌套循环，故算法复杂度为O(N^2)。



**2. 实验总结**

通过这次实验，我掌握了线性表的链式表示（包括单链表、循环链表、双向循环链表）；熟悉了链表实现线性表的基本操作，如建立、查找、插入、删除以及去重等；对于有序线性表的插入、删除、合并操作有了更深入的了解。