《数据结构》上机报告

2018 年 9 月 29 日

姓名： 赵得泽 学号： 1753642 班级： 电子 2 班 得分：

<table>
<tr><td>实验题目</td><td colspan="2">链表</td></tr>
<tr><td>问题描述</td><td colspan="2">链表是指采用链式存储结构的线性表，它用一组任意的存储单元存储线性表的数据元素（这组存储单元可以是连续的，也可以是不连续的）。因此对任一数据元素，除了存储其本身的信息外，还需存储其直接后继的存储位置，这两部分组成数据元素的存储映像，称为结点。若是双向链表，还需存储直接前驱的存储位置。</td></tr>
<tr><td rowspan="2">基本要求</td><td colspan="2">1. (p1)实现链表的基本操作，包括链表的初始化、第 i 个元素前插入一个新的元素、删除第 i 个元素、查找某元素、链表的销毁。
2. (p2)实现对单链表的创建（头插法，尾插法），逆置，查找和销毁。
3. (p3)实现单链表的去重，即只保留一个相同元素。
4. (p4) 实现两个有序（非递减）表合并生成新的有序表的功能。
5. (p5)无头结点单项循环链表实例应用：Josephus 问题。</td></tr>
<tr><td>已完成基本内容（序号）：</td><td>1，2，3，4，5</td></tr>
<tr><td rowspan="2">选做要求</td><td colspan="2"></td></tr>
<tr><td>已完成选做内容（序号）</td><td></td></tr>
<tr><td>数据结构设计</td><td colspan="2">

```
struct LNode
{
    int data;
    LNode *next;
};
```

本实验的数据结构主要运用的是线性表中的链表，通过链表的基本操作去实现问题。在该结构体中包含数据域和指针域，在内存中用一组任意的存储单元来存储线性表的数据元素，用每个数据元素所带的指针来确定其后继元素的存储位置。这两部分信息组成数据元素的存储映像，称作结点。正是利用结点的这种存储特点，元素的删除、插入，链表的逆置、去重等操作就显得更为方便。</td></tr>
</table>

功能(函数)说明	/*** 功能：创建单向有头结点链表 方法特点：头插法，逆序输出 输入参数：x 说明：元素每次插入的时候从头结点插入，头结点始终指向新插入的元素 ***/ void CreatList(LNode *head, int n) { LNode *p; int x; head->next = NULL; for (int i = n; i > 0; i--) { p = new LNode; cin >> x; p->data = x; p->next = head->next; head->next = p; } } /*** 功能：创建单向有头结点链表 方法特点：尾插法，正序输出 输入参数：x 说明：建立头节点，元素插入时每次都从尾部插入，新插入的元素总指向最后，即NULL; ***/ void CreatList(LNode *head) { LNode *p, *q; int x; q = head; while (1) { cin >> x; if (x == 0) break; p = new LNode; p->data = x; q->next = p; q = p; } q->next = NULL; } /***

```
功能：在指定位置前插入元素
输入参数：位置i,元素x
说明：插入的关键就是将插入的元素与i个元素链接，再找到第i个元素的前一个元素，
让它指向要插入的元素即可.
*************************************************************************/
void InsertList(LNode *head, int n, int i, int x)
{
    LNode *p, *q;
    q = new LNode;
    int j = 0;
    p = head;
    while (j < i - 1 && p)
    {
        p = p->next;
        j++;
    }
    if (!p)
        cout << -1 << endl;
    else
    {
        q->data = x;
        q->next = p->next;
        p->next = q;
        PrintList(head);
        cout << endl;
    }
}
/*************************************************************************
功能：在指定位置删除元素
输入参数：位置j
说明：关键是找到要删除元素的前一个元素，即第j-1个元素，然后还要保证
(p->next)!=NULL，即第i个元素的下一个元素不能为空，这样才能顺利将第j-1个元素于
第j+1个元素相链接，从而达到删除的目的.
*************************************************************************/
void DelList(LNode *head, int n, int j)
{
    LNode *p, *q;
    p = head;
    int i = 0;
    while (p->next&&i < j - 1)
    {
        p = p->next;
        i++;
    }
```

```
    if (!(p->next)
        cout << -1 << endl;
    else
    {
        q = p->next;
            p->next = q->next;
            PrintList(head);
            cout << endl;
    }
}
/************************************************************************
功能：查找指定元素
输入参数：y
说明：利用循环遍历链表，直到找到跳出循环，从而得到指定位置.
************************************************************************/
void SearchList(LNode *head, int y)
{
    LNode *p;
    p = head->next;
    int i = 1;
    while (p&&p->data != y)
    {
        p = p->next;
        i++;
    }
    if (p == NULL)
        cout << -1 << endl;
    else
        cout << i << endl;
}
/************************************************************************
功能：求链表长度
说明:利用循环直到表尾，再用i计数即可得到表长.
************************************************************************/
int Length(LNode *head)
{
    LNode *p;
    p = head->next;
    int i = 0;
    while (p)
    {
        p = p->next;
        i++;
    }
```

```
    return i;
}
/*************************************************************************
功能：打印表
说明：遍历输出即可
*************************************************************************/
void PrintList(LNode *head)
{
    LNode *p;
    p = head;
    p = p->next;
    while (p)
    {
        cout << p->data << " ";
        p = p->next;
    }
}
/*************************************************************************
功能：销毁表
说明：while循环遍历，删除每个结点.
*************************************************************************/
void destroy(LNode *head)
{
    LNode *p;
    p = head->next;
    while (p)
    {
        delete p;
        p = p->next;
    }
}
/*************************************************************************
功能：逆置表
说明：表的逆置是在同一张表中操作的，根据已经创建好的表，首先从表头进行遍历并保存第一个结点，然后从第二个结点开始，顺序移动指针，每次都将指针指向的元素和表头链接，直到表尾，从而改变元素的顺序.
*************************************************************************/
void Reverse(LNode *head)
{
    LNode *p, *q;
    p = head->next;
    head->next = NULL;
    while (p)
    {
```

```
            q = p;
            p = p->next;
            q->next = head->next;
            head->next = q;
        }
}
/*************************************************************************
功能：表去重
说明：该方法是在同一张表中进行操作的，主要操作就是以头结点及第一个结点为引导存储去重表（记为p），在以第二个结点为首（记为q)顺序遍历，并且p中的元素同时和q比较，若相等则跳过，该元素不与p进行链接;反之进行链接.
*************************************************************************/
void Rev_same(LNode *head)
{
    LNode *p, *q, *r, *s;
    p = head->next;
    r = head;
    int flag = 0;
    while (p)
    {
        if (flag == 0)
        {
            q = p;
            p = p->next;
            q->next = NULL;
            r->next = q;
            r = q;
        }
        s = head->next;
        while (s && p)
        {
            if (s->data == p->data)
            {
                p = p->next;
                flag = 1;
                break;
            }
            flag = 0;
            s = s->next;
        }
    }
}
```

```
/*****************************************************************************
功能：合并链表
输入参数：链表ha,hb
输出值：链表hc
说明：分别建立指针pa,pa遍历表ha,hb，并用pc初始化链表hc,若pa->data <
pb->data，pc链接pa,pa移动；反之，pc链接pb,pb移动；若是某张表到达表尾，则将pc
链接到另一张表，从而完成合并.
*****************************************************************************/
void Cmb_List(LNode *ha, LNode *hb, LNode *hc)
{
    LNode *pa, *pb, *pc;
    pa = ha->next;
    pb = hb->next;
    pc = hc;
    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;
}
/*****************************************************************************
功能：创建单向无头结点循环链表
说明：头结点置空，在建立新结点时，将通过判断头结点是否为空来初始化头结点，
每次插入的节点都作为尾结点，并将其指针域置空，直到最后将为结点和头结点相连，
构成无头结点循环链表.
*****************************************************************************/
LNode* CreatList(int n)
{
    LNode *h;
    LNode *r, *p;
    h = new LNode;
    h = NULL;
    r = NULL;
```

	`` for (int i = 1; i <= n; i++) { p = new LNode; p->data = i; p->next = NULL; if (h == NULL) { h = p; r = h; } else r->next = p; r = p; } r->next = h; return h; ```
开发环境	Win10,VS2017,C++高级程序语言设计
调试分析	Problem-1: ``` 6 12 23 34 45 65 67 6 50 7 65 67 65 45 34 23 12 67 65 45 34 23 50 12 67 65 45 34 23 50 2 6 ``` Problem-2: ``` 6 15 20 30 68 76 99 99 76 68 30 20 15 15 30 99 ``` Problem-3: ``` 9 12 23 432 3 12 23 2 12 2 12 23 432 3 2 ``` Problem-4: ``` 12 23 34 3 4 324 234 23 4313 0 23 2 233 98 100 0 2 3 4 12 23 23 23 34 98 100 233 234 324 4313 ``` Problem-5: ``` 10 2 3 4 7 10 3 8 2 9 6 1 5 ```

心得体会	本次实验主要是单向链表的创建（包括头插法和尾插法），删除，插入，逆置，打印，销毁以及其他的一些实际运用（比如合并有序链表，链表去重），在 problem-1 的算法重，这些操作得到了具体运用，其中还可以用本题测试循环链表和双向循环链表，从而掌握其用法。对于合并链表，它和顺序表的操作方法并无太大区别，但是链表去重使用链表比使用顺序表要方便很多，因为顺序表必须在连续的存储空间进行存储，而链表则可以通过指针的移动进行任意链接，从而大大提升了选择的范围，所以在链表去重这点，我只利用了原来的一条链表进行操作（顺序表则还要增加一条来存储去重表），主要操作就是以头结点及第一个结点为引导存储去重表（记为 p)，在以第二个结点为首（记为 q)顺序遍历，并且 p 中的元素同时和 q 比较，若相等则跳过，该元素不与 p 进行链接;反之进行链接。（具体源代码见“函数功能说明”）这样结束之后直接利用头结点直接输出去重表即可。在约瑟夫问题中用到了单向无头结点循环链表,主要就是它的创建及打印,通过判断 p->next 与 head 是否相等为终止条件，再实现问题要求的结果。最值得注意的是，链表的使用要分外小心，因为一不小心就可能会使程序运行到中途崩溃，出现未知 bug，也就是要防止在一条链表的表尾之前出现 NULL，尤其是在约瑟夫环和去重问题中进行插入、删除等操作时。总之，链表的使用必须严谨、认真地分析自己的每一步操作，以防程序崩溃。