

《数据结构》上机报告

2018 年 10 月 8 日

姓名：刘思源 学号：1651390 班级：电子三班 得分：_____

实验题目	链表	
问题描述	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。	
基本要求	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，求链表的长度。 2. (p2)实现对单链表的倒置，输出特定元素的功能 3. (p3)实现对单链表的去重 4. (p4)合并两个无序单链表，得到一个有序的链表 5. (p5)用链表的方式解决约瑟夫问题	
	已完成基本内容（序号）：	1, 2, 3, 4, 5
选做要求		
	已完成选做内容（序号）	
数据结构设计	<pre> typedef struct LNode { ElemType val; struct LNode * next; LNode(int x) :val(x), next(nullptr) {} }LNode, *LinkList; </pre>	

<div>功能(函数)说明</div>	<div> <p>1、链表的初始化</p> <p>已知元素个数的初始化方法。</p> <pre> void initial(LinkList &L, int n) { LNode *p, *q; L = new LNode(0); q = L; int i; for (i = n; i > 0; i--) { p = new LNode(0); cin >> p->val; q->next = p; q = p; } q->next = nullptr; return; } </pre> <p>未知元素个数的初始化方法：</p> <pre> void initial_ordered(LinkList &L) { LNode *p, *q; L = new LNode(0); q = L; int num; while(1) { cin >> num; if (num == 0) { break; } p = new LNode(0); p->val = num;; q->next = p; q = p; } q->next = nullptr; return; } </pre> </div> <div> <p>2、基本的插入操作</p> <p>输入插入元素的位置和运算的值，实现数据的插入。每次分配一次空间。找到元素位置后，将该位置前驱结点连接到新结点，再将新结点连接到后继节点。如果插入的位置不合法，输出-1。</p> <pre> Status insert(LinkList &L, int pos, ElemType target) { if (pos > getLength(L)+1) { </pre> </div>
---------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```

        return INFEASIBLE;
    }
    int count = 0;
    LNode *head, *p=nullptr;
    head = L;
    if (head->next) {
        p = head->next;
    }
    if (pos <= 0) {
        return INFEASIBLE;
    }
    while (head) {
        count++;
        if (count == pos) {
            head->next = (LinkedList)malloc(sizeof(LNode));
            head = head->next;
            head->val = target;
            if (p) {
                head->next = p;
            }
            else {
                head->next = nullptr;
            }
            return OK;
        }
        head = head->next;
        if (count > getLength(L)) {
            head->next = nullptr;
            return INFEASIBLE;
        }
        if (p->next) {
            p = p->next;
        }
        else {
            p->next = nullptr;
            p = p->next;
        }
    }
    return INFEASIBLE;
}

```

3、基本的删除操作

遍历链表，位置不合法，返回-1。如果找到，将该位置前驱结点连接到该元素的后继结点。并且释放该元素的空间。

```

Status delete_LinkList(LinkedList &L, int pos)

```

```

{
    if (pos > getLength(L)) {
        return INFEASIBLE;
    }

    LNode *head;
    head = L;
    LNode *p;
    p = L->next;
    int count = 0;
    while (head) {
        count++;
        if (count == pos) {
            LNode *q;
            q = p;
            if (p) {
                p = p->next;
                head->next = p;
            }
            else {
                head->next = nullptr;
                free (p);
            }
            free(q);
            return OK;
        }
        head = head->next;
        if (!head) {
            return INFEASIBLE;
        }
        if (p->next) {
            p = p->next;
        }
        else {
            p->next = nullptr;
        }
    }
    return INFEASIBLE;
}

```

4、查找指定元素位置

遍历链表，如果找到该元素，返回位置，如果未找到，返回-1。

```

Status locate(LinkList L, int target)
{
    LNode *head;

```

```

head = L;
int count = 0;
while (head) {
    if (head->val == target) {
        return count;
    }
    head = head->next;
    count++;
}
return INFEASIBLE;
}

```

5、求链表的长度

遍历链表，使用一个计数器，当链表遍历结束后返回计数器的值。

```

Status getLength(LinkList L)
{
    LNode *head;
    head = L->next;
    int count = 0;
    while (head) {
        head = head->next;
        count++;
    }
    return count;
}

```

6、链表的倒置

断开头节点和链表，依次将每个后继结点连接到前驱结点上，当最后一个结点完成连接后，将头节点连接到该节点上。

```

LNode *reverse(LNode *head)
{
    if (head->next == nullptr && head->next->next == nullptr) {
        return head;
    }
    LNode *tail = head->next;
    LNode *p = tail->next;
    while (p != nullptr) {
        tail->next = p->next;
        p->next = head->next;
        head->next = p;
        p = tail->next;
    }
    return head;
}

```

7、链表的去重

使用三个指针，分别指向头结点，结点 p 和结点 p 的前驱结点。遍历链表，

比较当前结点的值是否等于任意一个该结点前方的结点的值，如果等于，将该结点的前驱结点连接到该结点的后继节点，释放该结点的空间。

```
void delete_rep(LinkList L)
{
    LNode *head, *p,*heada;
    heada = L;
    head = L->next;
    p = L->next;
    bool flag = false;
    while (head) {
        flag = false;
        while (p != head) {
            if (p->val == head->val) {
                if (head->next != nullptr) {
                    heada->next = head->next;
                }
                else {
                    heada->next = nullptr;
                    free(head);
                    return;
                }
                flag = true;
                break;
            }
            p = p->next;
        }
        LNode *q=nullptr;
        if (flag) {
            q = head;
        }
        if (head->next != nullptr) {
            head = head->next;
            if (q) {
                free(q);
            }
        }
        else {
            return;
        }
        if (!flag) {
            heada = heada->next;
        }
        p = L->next;
    }
}
```

```
return;
}
```

8、无序链表合并为有序链表

本题目首先使用遍历比较，同时遍历两个链表，将两个链表内的较小值合并入新的 L3 表，剩下的进行插入排序。但是这样做时间复杂度过高。所以直接将 L2 的表头接到 L1 的表尾，然后对新链表进行一次快速排序。可以在 $O(\log n)$ 时间内完成。

```
LNode *quickSortList(LNode *head)
{
    if (head == NULL || head->next == NULL) {
        return head;
    }
    LNode tmpHead(0);
    tmpHead.next = head;
    qsortList(&tmpHead, head, NULL);
    return tmpHead.next;
}

void qsortList(LNode *headPre, LNode*head, LNode*tail)
{
    if (head != tail && head->next != tail) {
        LNode* mid = partitionList(headPre, head, tail);
        qsortList(headPre, headPre->next, mid);
        qsortList(mid, mid->next, tail);
    }
}

LNode* partitionList(LNode* lowPre, LNode* low, LNode* high)
{
    int key = low->val;
    LNode node1(0), node2(0);
    LNode* little = &node1, *big = &node2;
    for (LNode*i = low->next; i != high; i = i->next) {
        if (i->val < key) {
            little->next = i;
            little = i;
        }
        else {
            big->next = i;
            big = i;
        }
    }
    big->next = high;
    little->next = low;
    low->next = node2.next;
    lowPre->next = node1.next;
}
```

```

        return low;
    }

void Merge(LinkList L1, LinkList L2, LinkList L3)
{
    LNode *p, *q, *t;
    p = L1->next;
    q = L2->next;
    t = L3;
    t->next = L1->next;
    while (t->next) {
        t = t->next;
    }
    t->next = L2->next;
}

```

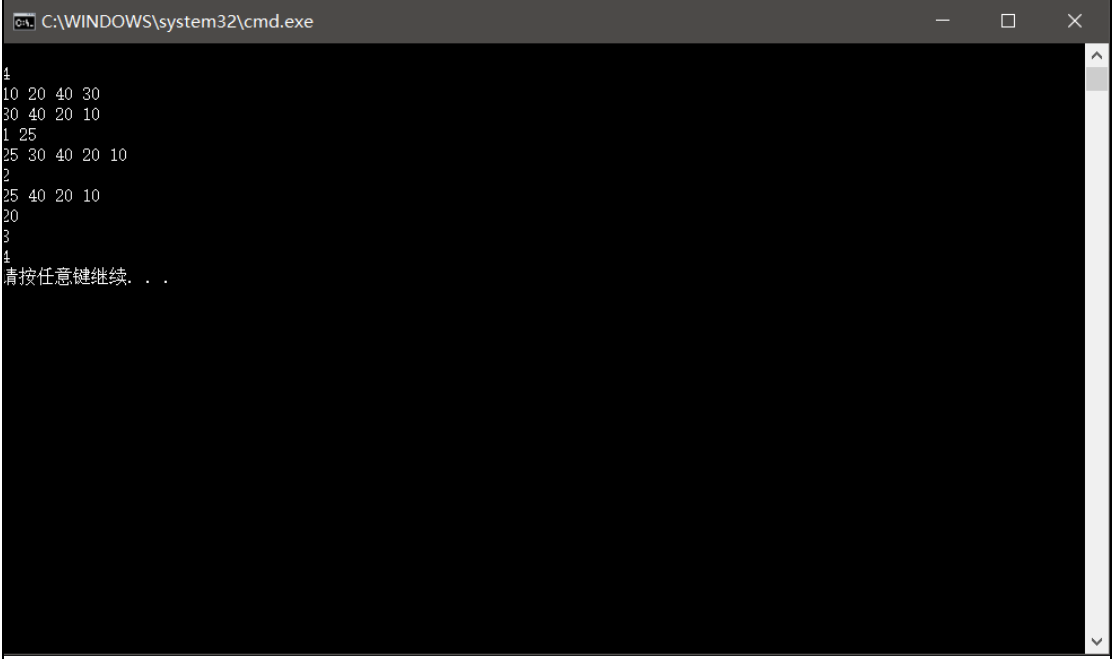
9、约瑟夫问题

首先将 n 个人存储在一个循环链表中，每次搜索到第 m 个人的时候，输出该位置的数字，将该位置的前驱结点连接到后继节点，释放该位置空间。当某时刻，前驱结点就是后继节点本身的时候，说明链表中只有一个元素，输出该元素，释放空间。

```

void Josephus(LinkList L, int n, int s, int m)
{
    initial(L, n);
    LNode *p, *q;
    int i;
    q = L;
    p = L->next;
    bool flag = true;
    while (L->next) {
        if (flag) {
            for (i = 1; i < s; i++) {
                p = p->next;
                q = q->next;
            }
            flag = false;
        }
        for (i = 1; i < m; i++) {
            p = p->next;
            q = q->next;
        }
        if (p != q) {
            cout << p->val << " ";
            LNode *t;
            t = p;

```


	<pre> q->next = p->next; p = p->next; free(t); } if (p == q) { cout << p->val << endl; free(p); return; } }</pre>
开发环境	Visual studio 2017
调试分析	



	
心得体会	<ol style="list-style-type: none"> 1、 链表使用头结点将能很方便的对后续的问题进行处理，否则在进行插入等操作的时候都不是很方便。 2、 链表的删除、插入、替换实际上采用的思路和顺序表是不同的。很多时候直接将前结点连接到后结点，或者是将值调换，都是更为简单的操作方法。链表本身的物理结构就很灵活，在进行处理时也要灵活处理。 3、 在进行无序链表合并为有序链表时，本文采用了快速排序的方法。个人认为这不是最好的方法，该题还需思考。