PA2 链表实验报告　链表实验报告

姓名：钱子贤 学号：2054170 日期：2021 年 10 月 17 日

实验报告格式按照模板来即可，对字体大小、缩进、颜色等不做要求

实验报告要求在文字简洁的同时将内容表示清楚

1. 涉及数据结构和相关背景

实验目的 ：

1 、掌握线性表的链式表示（单链表、循环链表、双向循环链表)；

2 、掌握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,；

3、 掌握有序线性表的插入、删除、合并操作;

2. 实验内容

```
//线性表的单链表存储结构
typedef struct Node {
    ElemType data;
    struct Node* next;
}Node, * LinkList;
```

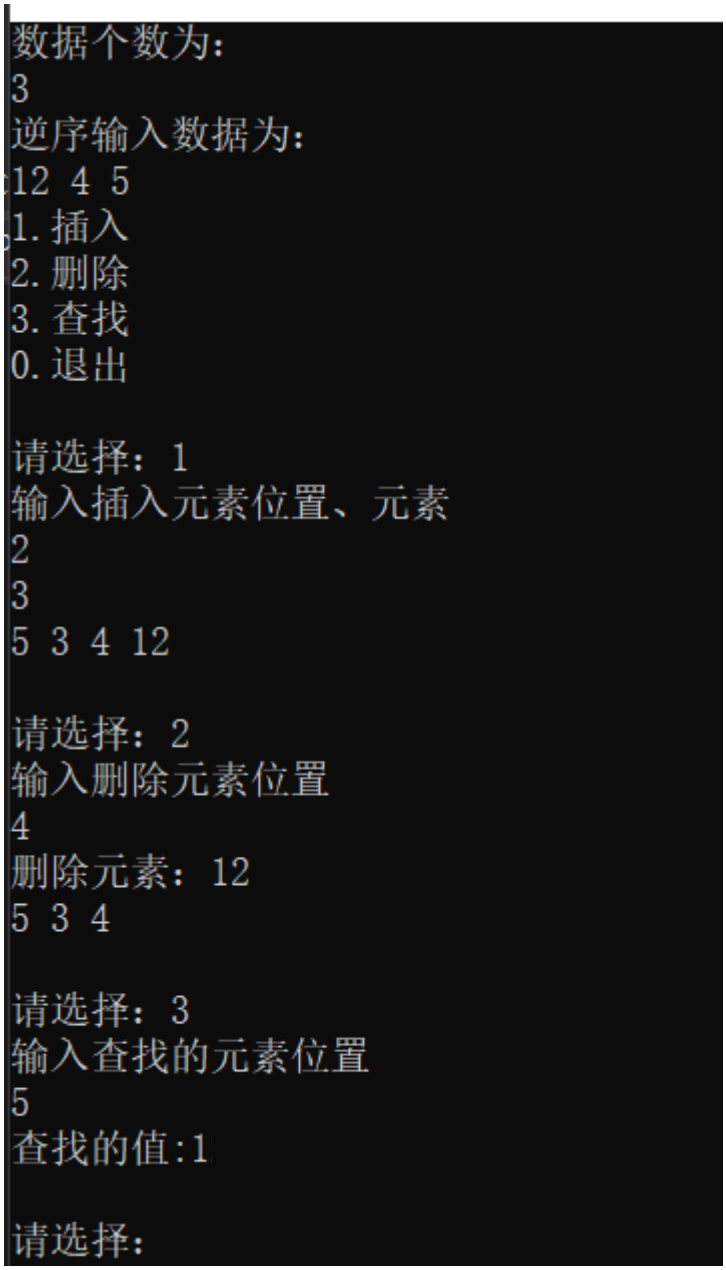
(1) 链表的基本操作;

```
(2) //链表的创建
(3) Status create(LinkList& L, int n)
(4) {
(5)     Node* p;
(6)     L = (LinkList)malloc(sizeof(Node));
(7)     L->next = NULL;
(8)     for (int i = n; i > 0; --i)
(9)     {
(10)        p = (LinkList)malloc(sizeof(Node));
(11)        cin>>p->data;
(12)        p->next = L->next;
(13)        L->next = p;
(14)    }
(15)    return 0;
(16) }
(17)
(18) //链表的插入
(19) Status insert(LinkList& L, int i, ElemType stu)
```

```
(20) {
(21)     Node* p, * s;
(22)     int t=0;
(23)     p = L;
(24)     while (p && t < i - 1)
(25)     {
(26)         p = p->next;
(27)        t++;
(28)     }
(29)     if (!p || t < i - 1)
(30)         return ERROR;
(31)     s = (LinkList)malloc(sizeof(Node));
(32)     s->data = stu;
(33)     s->next = p->next;
(34)     p->next = s;
(35)     return OK;
(36) }
(37)
(38) //链表的删除
(39) Status deletes(LinkList& L, int i, ElemType& stu)
(40) {
(41)     Node* p, * q;
(42)    p = L;
(43)     int t = 0;
(44)     while (p->next && t < i - 1)
(45)     {
(46)         p = p->next;
(47)         t++;
(48)     }
(49)     if (!p || t > i - 1)
(50)         return ERROR;
(51)     q = p->next;
(52)     p->next = q->next;
(53)     stu = q->data;
(54)     free(q);
(55)     return OK;
(56) }
(57)
(58) //链表的查找
(59) Status search(LinkList L, int i, ElemType& stu)
(60) {
(61)     Node* p;
(62)     p = L;
(63)     int t = 0;
```

```
(64)     while (p && t < i)
(65)     {
(66)         p = p->next;
(67)        t++;
(68)     }
(69)     if (!p || t > i)
(70)         return ERROR;
(71)     stu = p->data;
(72)     return OK;
(73) }
(74)
(75) //链表的显示
(76) void print(LinkList L, int n)
(77) {
(78)     Node* p;
(79)     p = L->next;
(80)     for (int i = 0; i <= n; i++)
(81)     {
(82)        cout<<p->data<<" ";
(83)         p = p->next;
(84)     }
(85) }
```

时间复杂度 O(n)



(2) 链表的逆置;

```
LNode *Inverse(LNode *L)
{
    LNode *p, *q;
    p = L->next;
    L->next = NULL;
    while (p != NULL)
```

```
    {
        q = p;
        p = p->next;

        q->next = L->next;
        L->next = q;
    }
    return L;
}
```

```
数据个数为:
3
逆序输入数据为:
1 2 3
1. 插入
2. 删除
3. 查找
0. 退出

请选择: 1
输入插入元素位置、元素
0 0
0 3 2 1
```

相当于逆序输出，时间复杂度 O(n)

(3) 链表的去重;

```
void deleteSame(Linklist str1)
{
    Linklist p = str1->next;
    if (p == NULL) {
        return;
    }
    while (p->next != NULL) {
        Linklist u = p->next;
        if (p->data == u->data) {
            p->next = u->next;
            free(u);//删除结点 删除链表
        }
        else
            p = p->next;
    }
```

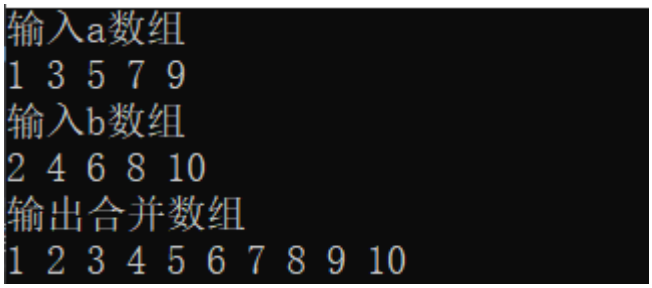
```
}
```

```
输入个数
10
输入元素
1 2 3 3 4 44 44 5 6 6
去重后链表为
1 2 3 4 44 5 6
```

时间复杂度 O(n)

(4) 合并两个有序链表;

```
void function(Linklist str1, Linklist str2) {
    Linklist pa = str1->next, pb = str2->next;
    Linklist ra = str1;
    while (pa && pb) {
        if (pa->data <= pb->data) {
            ra->next = pa;
            ra = pa;
            pa = pa->next;
        }
        else {
            ra->next = pb;
            ra = pb;
            pb = pb->next;
        }
    }
    if (pa)
        pb = pa;
    while (pb) {
        ra->next = pb;
        ra = pb;
        pb = pb->next;
    }
    ra->next = NULL;
    free(str2);
}
```



时间复杂度 O(n)

2.1.5 调试分析（遇到的问题和解决方法）

调试图片如上，问题在于链表要注意释放内存，不然会编译报错。有些问题直接循环解决不了，会出现越界的情况，递归解决，终止条件为!head || !head->next，返回值为处理完后的子链表，传递给两个交换节点的前一个，后一个节点再指向前一个，再返回。递归比遍历方法简化在遍历是从后往前的，所以不需要记录前一个节点，因为可以直接作为返回值传递给上一个函数，因此也不需要考虑头节点的情况。

2.1.6 总结和体会

去重一定是原链，去重一定是要破坏原链的，如果一定不破坏原链 就将原链所有元素复制到新链两个尾拆分为两个，所以一个 while 循环中有两步，合并为一个，所以一个 while 循环中只有一步，但是是有序的 所以要 if else， 其实大部分链表问题都可以分为两种，遍历和递归。遍历比较简单，因为没有重复的数字，所以需要一个节点，涉及至少三个节点，操作比较繁琐，但算法时间复杂度为 O(n)。而每次递归需要分两种情况，当存在重复元素时，函数返回值应赋值给最后一个元素的下一个元素，并返回下一个元素。当不存在重复元素时，函数返回值赋值给下一个元素，此时是返回当前节点。

3.实验总结

创建链表需要将既定数据按照链表的结构进行存储，本文以一种最简单的方式来演示：使用数组对链表赋值。将原来在连续空间存放的数组数据，放置在不连续的链表空间中，使用指针进行链接。迭代是从前往后依次处理，直到循环到链尾；而递归恰恰相反，首先一直迭代到链尾也就是递归基判断的准则，然后再逐层返回处理到开头。总结来说，链表翻转操作的顺序对于迭代来说是从链头往链尾，而对于递归是从链尾往链头。因为查找数据时效率低，因为不具有随机访问性，所以访问某个位置的数据都要从第一个数据开始访问，然后根据第一个数据保存的下一个数据的地址找到第二个数据，以此类推。 要找到第三个人，必须从第一个人开始问起，不指定大小，扩展方便。链表大小不用定义，数据随意增删。

源代码：

```
#include <stdio.h>
#include <stdlib.h>
#include<iostream>
using namespace std;
#define OK 1
#define ERROR 0

typedef int Status;
typedef int ElemType;

//线性表的单链表存储结构
typedef struct Node {
    ElemType data;
    struct Node* next;
}Node, * LinkList;

//链表的创建
Status create(LinkList& L, int n)
{
    Node* p;
    L = (LinkList)malloc(sizeof(Node));
    L->next = NULL;
    for (int i = n; i > 0; --i)
    {
        p = (LinkList)malloc(sizeof(Node));
        cin>>p->data;
        p->next = L->next;
        L->next = p;
    }
    return 0;
}

//链表的插入
Status insert(LinkList& L, int i, ElemType stu)
{
    Node* p, * s;
    int t=0;
    p = L;
    while (p && t < i - 1)
    {
        p = p->next;
       t++;
```

```
    }
    if (!p || t < i - 1)
        return ERROR;
    s = (LinkList)malloc(sizeof(Node));
    s->data = stu;
    s->next = p->next;
    p->next = s;
    return OK;
}

//链表的删除
Status deletes(LinkList& L, int i, ElemType& stu)
{
    Node* p, * q;
   p = L;
    int t = 0;
    while (p->next && t < i - 1)
    {
        p = p->next;
        t++;
    }
    if (!p || t > i - 1)
        return ERROR;
    q = p->next;
    p->next = q->next;
    stu = q->data;
    free(q);
    return OK;
}

//链表的查找
Status search(LinkList L, int i, ElemType& stu)
{
    Node* p;
    p = L;
    int t = 0;
    while (p && t < i)
    {
        p = p->next;
       t++;
    }
    if (!p || t > i)
        return ERROR;
    stu = p->data;
```

```
    return OK;
}

//链表的显示
void print(LinkList L, int n)
{
    Node* p;
    p = L->next;
    for (int i = 0; i <= n; i++)
    {
       cout<<p->data<<" ";
        p = p->next;
    }
}

//链表的显示
void print(LinkList L)
{
    LinkList p;
    p = L->next;
    while (p != NULL)
    {
       cout<<p->data<<" ";
        p = p->next;
    }
   cout<<"\n";
}

int main()
{
    LinkList L;
    Status i, n, select, k;
    ElemType stu;
    cout << "数据个数为：" << endl;
    cin >> n;
   cout<<"逆序输入数据为：\n";
    create(L, n);
   cout<<"1.插入\n2.删除\n3.查找\n0.退出\n";
    for (k = 0;; k++)
    {
       cout<<"\n请选择：";
        cin >> select;
        if (select == 1)
        {
```

```
            cout<<"输入插入元素位置、元素\n";
             cin >> i >>  stu;
             insert(L, i, stu);
             print(L);
        }
        if (select == 2)
        {
            cout<<"输入删除元素位置\n";
             cin >> i;
             deletes(L, i, stu);
            cout<<"删除元素："<<stu<<endl;
             print(L);
        }
        if (select == 3)
        {
            cout<<"输入查找的元素位置\n";
             cin >> i;
             search(L, i, stu);
            cout<<"查找的值:"<<stu<<endl;
        }
        if (select == 0)
            break;
    }
    return OK;
}
```

```
#include <iostream>
#include<stdio.h>
#include<stdlib.h>
using namespace std;
typedef struct node {
    int data;
    struct node* next;
}Node, * Linklist;

void deleteSame(Linklist str1)
{
    Linklist p = str1->next;
    if (p == NULL) {
        return;
    }
```

```
    while (p->next != NULL) {
        Linklist u = p->next;
        if (p->data == u->data) {
            p->next = u->next;
            free(u);//删除结点 删除链表
        }
        else
            p = p->next;
    }
}
int main()
{
    int n;
    cout << "输入个数" << endl;
    cin >> n;
    int a[100];
    cout << "输入元素" << endl;
    for (int i = 0; i < n; i++)
        cin >> a[i];
    cout << "去重后链表为" << endl;
    Linklist first = new Node;
    Linklist stu = first;
    for (int i = 0; i < n; i++) {
        Linklist s = new Node;
        s->data = a[i];
        stu->next = s;
        stu = s;
    }
    stu->next = NULL;


    Linklist p = first->next;

    deleteSame(first);
    p = first->next;
    while (p) {
        cout << p->data;
        cout << " ";
        p = p->next;
    }
    cout << endl;


    return 0;
```

```
}
#include <iostream>
#include<stdio.h>
#include<stdlib.h>
using namespace std;
typedef struct node {
    int data;
    struct node* next;
}Node, * Linklist;

void function1(Linklist str1, Linklist str2) {
    Linklist pa = str1->next, pb = str2->next;
    Linklist ra = str1;
    while (pa && pb) {
        if (pa->data <= pb->data) {
            ra->next = pa;
            ra = pa;
            pa = pa->next;
        }
        else {
            ra->next = pb;
            ra = pb;
            pb = pb->next;
        }
    }
    if (pa)
        pb = pa;
    while (pb) {
        ra->next = pb;
        ra = pb;
        pb = pb->next;
    }
    ra->next = NULL;
    free(str2);
}
int main()
{
    int a[5];
    cout << "输入a数组" << endl;
    for (int i = 0; i < 5; i++)
        cin >> a[i];
    Linklist first = new Node;
    Linklist r = first;
    for (int i = 0; i < 5; i++) {
```

```
        Linklist s = new Node;
        s->data = a[i];
        r->next = s;
        r = s;
    }
    r->next = NULL;
    Linklist p = first->next;
    cout << "输入b数组" << endl;

    int b[5];
    for (int i = 0; i < 5; i++)
        cin >> b[i];
    Linklist first2 = new Node;
    Linklist r2 = first2;
    for (int i = 0; i < 5; i++) {
        Linklist s = new Node;
        s->data = b[i];
        r2->next = s;
        r2 = s;
    }
    r2->next = NULL;
    p = first2->next;
    function1(first, first2);
    cout << "输出合并数组" << endl;

    p = first->next;
    while (p) {
        cout << p->data<<" ";
        p = p->next;
    }
    return 0;
}
```