

实 验 报 告

课程名称： 数据结构 (C)

实验项目： 线性表的生成与操作

实验仪器： 计算机

| 项目 | 报告格式 | 写作质量 | 逻辑、注释质量、思想描述 | 复杂度分析 | 合计 |
|---------|----------|-----------|--------------|-----------|------------|
| 百分比 (%) | 15 | 25 | 40 | 20 | 100 |
| 得分 | 9 | 25 | 25 | 10 | 100 |

系 别： 计算机学院

专 业： 数据科学与大数据技术

班级姓名： 大数据 1701 张丹颖

日 期： 2018.10.07

成 绩：

同组成员： 无

指导教师： 丁濛

一、实验目的

1. 掌握线性表的顺序存储和链式存储结构；
2. 验证顺序表及链表的基本操作的实现；(验证)
3. 理解算法与程序的关系，能够将算法转换为对应程序；
4. 体会线性表在实际应用中能够解决的问题。(设计、综合)

二、实验内容

2.1 项目一

实现顺序存储的线性表并分析每个功能的复杂度。要求具有如下功能：

1. 构造，析构；
2. 随机访问；
3. 在表后添加一个新数据；
4. 向表中特定位置插入一个新数据；
5. 将另外一张表内的内容添加到一个表的后面；
6. 查找表中是否含有特定元素；
7. 删除表中的某个特定元素；
8. 打印所有元素。

2.2 项目二

实现链式存储的线性表（单向链表）并分析每个功能的复杂度。要求具有如下功能：

1. 构造，析构；
2. 在第一个元素之前插入一个新数据；在最后一个元素之后插入一个新数据；
3. 删除第一个元素；删除最后一个元素；
4. 查找表中是否含有特定元素；
5. 向表中特定元素之前/之后插入一个新数据；
6. 删除表中的某个特定元素；
7. 遍历整个链表；
8. 反转一个链表，即 $a \rightarrow b \rightarrow c \rightarrow d$, 反转后为 $d \rightarrow c \rightarrow b \rightarrow a$.

2.3 项目三

以双向链表的形式重新实现第 2 题，并分析两种实现中各个功能的复杂度区别。

2.4 项目四

完成上机平台上实验一的题目。

三、实验过程

3.1 项目 1

大致思想：封装 MyArray 类，在主函数中实例化，实现其各种方法；

1. 构造和析构函数，令 size=0 和头指针置空（复杂度 $O(1)$ ）；
 2. 随机访问，直接利用数组下标访问（复杂度 $O(1)$ ）；
 3. 追加数字，由于数组容量有限，需要新开辟原 size+1 的空间，再把数字逐个复制过去（复杂度 $O(n)$ ）；
 4. 插入数字，开辟新空间，逐个将前 index-1 和后 index+1 复制，中间 index 位插入（复杂度 $O(n)$ ）；
 5. 追加表，开辟新空间，逐个将旧表和新表的值 copy 过去（复杂度 $O(n)$ ）；
 6. 查找特定元素，需要遍历（复杂度 $O(n)$ ）；
 7. 删除元素，从 index 位开始，后一位数覆盖前一位（复杂度 $O(n)$ ）；
 8. 打印，需要遍历（复杂度 $O(n)$ ）
- (1) 实验步骤
 - (2) 必要代码

```
/*
    subject: Data Structure
    To: experiment1_problem1
    Creator: 张丹颖2017011760
    date:2018.10.08
*/

#include<iostream>
// #include<cstring>
using namespace std;

struct MyArray{
    int *data;
    int size;
```

```
MyArray(){//构造函数
    data = NULL;
    size = 0;
}

~MyArray(){//析构
    delete data;
    data = NULL;
    size = 0;
}

void Create_Array(){
    cout<<"please input the size:";
    cin>>size;
    data = new int[size];//存多少数，new多大空间
    cout<<"please input number:";
    for(int i=0;i<size;++i){
        cin>>data[i];
    }
    cout<<"create successfully!" <<endl;
}

//利用数组下标随机访问
int &GetItem(int index){
    int m = -1, n = -2;
    if(size==0){//返回-1表示空表
        return m;
    }

    if(size>index&&index>=0){
        return (data)[index];
    }
    else{
        return n;//返回-2表示下标不合法
    }
}

//有引用&就可以实行右赋值操作getItem(s,4) = 6

//表后添加
void Append(int d){
    size++;
    int *temp = new int[size];//开辟新空间
    temp[size-1]=d;
    for(int i=0;i<size-1;++i){//数据复制
        temp[i] = data[i];
    }
    delete [] data;//释放原来的data
    data = temp; //给data赋新值
```

```
        cout<<"append successfully!"<<endl;
    }

    //指定下标前面插入新数据
    void InsertBefore(int d,int index){
        if(index<0){           //检查输入的下标
            cout<<"input illegal!"<<endl;
            return;
        }

        size++;
        int *temp = new int[size];    //开辟新空间

        for(int i=0;i<index;++i){    //index前面的数据复制
            temp[i] = data[i];
        }

        temp[index]=d;                //插入该数据

        for(int i=index+1;i<size;++i){ //index 后面的数据复制
            temp[i]=data[i-1];
        }

        delete [] data;
        data= temp;
        cout<<"insert successfully!"<<endl;
    }

    //将另外一张表的内容添加到这个表的后面
    int AddTableBack(int table[] ,int l){
        //int NewSize =
            size+strlen(table);在c++的函数里面，如果把一个数组作为参数传进去，那么这个数组就会退化为一个指针，因而就不知
            此法不可行

        int NewSize = size+l;
        int *temp = new int [NewSize]; //开辟能容纳两个表大小的新空间
        //数据赋值
        for(int i=0;i<size;++i){
            temp[i] = data[i];
        }

        int j = 0;
        for(int i=size;i<NewSize;++i){ //此处注意下标
            temp[i] = table[j++];
        }

        delete [] data;
        data = temp;

        size = NewSize;
```

```
    return size;//返回新线性表的长度
}

//查找表的特定元素
int LocateItem(int d){
    int i=0;
    for( i=0;i<size;++i){
        if(data[i] == d){
            return i;//return直接结束该函数调用，与break有区别;
        }
    }
    return -1;
}

//删除表中某特定元素
int DeleteItem(int index){
    if(index<0){//检查下标合不合理
        return -1;
    }
    size--;
    int ans = data[index];
    for(int i=index;i<size;++i){//从index位开始，后一位数覆盖前一位
        data[i]=data[i+1];
    }
    return ans;//返回删除的数
}

//打印所有元素
void printAll() {
    if(size==0){
        cout<<"Linear table is empty!, please create first!"<<endl;
        return;
    }

    for(int i=0;i<size;++i){
        cout<<data[i]<<' ';
    }
    cout<<endl;
}

};

int main(){
    char ch;
    MyArray a;

    bool flag = 1;//退出界面标志
    while(flag){
```

```
cout<<"a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)"<<endl;
cin>>ch;
switch(ch){
    case 'a':{
        a.Create_Array();
        break;
    }

    case 'b':{
        int inb ;
        cout<<"input index that you want to visit:";
        cin>>inb;
        int outb = a.Getltem(inb);

        if(outb==--1){
            cout<<"Linear table is empty!, please create first!"<<endl;
        }
        else if(inb==--2){
            cout<<"input illegal!"<<endl;
        }
        else{
            cout<<"the number is "<<outb<<endl;
        }
        break;
    }

    case 'c':{
        int inc;
        cout<<"input the num you want to append:";
        cin>>inc;
        a.Append(inc);
        break;
    }

    case 'd':{
        int dd,ind;
        cout<<"input the num and place(index)";
        cin>>dd>>ind;
        a.InsertBefore(dd,ind);
        break;
    }

    case 'e':{
        int scale;

        cout<<"input the size of your inserted array:";
        cin>>scale;
        int *array = new int[scale];
        cout<<"input its number:";
        for(int i=0;i<scale;++i){
```

```
        cin>>array[i];
    }

    cout<<"the length of combined table is "<<a.AddTableBack(array,scale)<<endl;
    break;
}

case 'f':{
    if(a.size==0){
        cout<<"Linear table is empty!, please create first!"<<endl;
        break;;
    }
    int inf;
    cout<<"input the number you wanna find:";
    cin>>inf;

    int outf = a.LocateItem(inf);
    if(outf==-1){
        cout<<"can' t find!"<<endl;
    }
    else{
        cout<<"the index is "<<outf<<endl;
    }
    break;
}

case 'g':{
    if(a.size==0){
        cout<<"Linear table is empty!, please create first!"<<endl;
        break;
    }

    int ing;
    cout<<"input the index you wanna delete:";
    cin>>ing;

    int outg = a.DeleteItem(ing);
    if(outg==-1){
        cout<<"input illegal!"<<endl;
    }
    else{
        cout<<"the num you delete is "<<outg<<endl;
    }
    break;
}

case 'h':{
    a.printAll();
    break;
}
```



```

        case 'i':{
            flag =0;
            break;
        }

        default:{
            cout<<"input illegal&please input again!"<<endl;
            break;
        }

    }
}

return 0;
}

```

(3) 实验结果

```

a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)
a
please input the size:5
please input number:1 2 3 4 5
create successfully!
a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)
a
input the size of your inserted array:5
input its number:6 7 8 9 10
the length of combined table is 10
a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)
a
input index that you want to visit:6
the number is 7
a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)
a
input the num you want to append:11
append successfully!
a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)
a
1 2 3 4 5 6 7 8 9 10 11
a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)
a
input the num and place(index)0
0
insert successfully!
a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)
a
0 1 2 3 4 5 6 7 8 9 10 11
a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)
a
input the number you wanna find:44
can't find!
a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)
a
input the number you wanna find:6
the index is 6
a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)
a
input the index you wanna delete:11
the num you delete is 11
a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)
a
0 1 2 3 4 5 6 7 8 9 10
a(create),b(visit),c(append),d(insert),e(addTable),f(find),g(delete),h(print),i(quit)

```

图 1: 问题一结果

3.2 项目 2

大致思想：封装 LinkList 类，实现其各种方法

1. 构造和析构函数，为 head 结点开辟空间或释放空间（复杂度 $O(1)$ ）；
2. 第一个元素之前插入（复杂度 $O(1)$ ）；
3. 最后一个元素之后插入（复杂度 $O(1)$ ）；
4. 删除第一个数（复杂度 $O(1)$ ）；

5. 删除最后一个数（复杂度 $O(1)$ ）；
6. 查找特定元素，需要遍历（复杂度 $O(n)$ ）；
7. 元素之前插入，需要遍历找到该元素（复杂度 $O(n)$ ）；
8. 元素之后插入，需要遍历（复杂度 $O(n)$ ）；
9. 删除任意一个数，需要遍历（复杂度 $O(n)$ ）；
10. 打印，需要遍历（复杂度 $O(n)$ ）
11. 递归反转，需要遍历找到倒数第二个结点（复杂度 $O(n)$ ）
12. 常规反转，需要遍历走一遍（复杂度 $O(n)$ ）

(1) 实验步骤

(2) 必要代码

```
/*
subject: Data Structure
To: experiment1_problem2
Creator: 张丹颖2017011760
date:2018.10.08
*/

#include<iostream>
using namespace std;

struct Node{
    int data;
    Node * next = NULL;
};

class LinkList{
private:
    int len ;
    Node *phead = NULL;
    Node *ptail = NULL;

public:
    LinkList (){//构造
        phead = new Node;
        ptail = phead;
    }

    ~LinkList (){//析构
        delete phead ;
        phead = NULL;
    }
};
```

```
}

Node* getHead(){
    return phead;
}

//初始化链表
void Create_List(){//头结点不存放有效数据，尾结点的指针域指向空
    len = 0;
    Node *p = phead;
    cout<<"create a LinkList(5)"<<endl;
    for(int i=0;i<5;++i){
        Node *pnew = new Node;
        cin>>pnew->data;
        pnew->next = NULL;

        p->next = pnew;
        p = p->next;
        len++;
    }

    ptail = p;//尾指针
}

//第一个元素之前插入
void InsertHead(int d){//即在phead后插入一个结点
    Node* newphead = new Node;
    newphead->next = phead->next;//新结点指向原来的phead的下一个结点
    phead->next = newphead;//更新phead的下一个结点为这个新结点
    newphead->data = d;
    len++;
}

//最后一个元素之后插入新数据
void InsertTail(int d){//定义了ptail不需要再遍历
    Node * newPtail = new Node;
    newPtail->next = NULL;
    newPtail->data = d;
    ptail->next = newPtail;//直接在结尾ptail后追加新结点
    ptail = newPtail;//更新ptail
    len++;
}

//删除头结点，即删除phead紧接的那个结点
int DeleteHead(){
    int datatemp;
    Node* p;
    datatemp = phead->next->data ;//存储删除那个结点的数据域
    p = phead->next;
    phead->next = phead->next->next;//断开phead和它的直接后继
```

```
    delete p;
    p = NULL;

    len--;
    return datatemp;
}

//删除尾结点
int DeleteTail(){
    Node* p = phead->next;
    for(int i = 0; i < len - 2; ++i){ //遍历到尾结点的前一个结点
        p = p->next;
    }
    int datatemp = p->next->data;

    ptail = p; //更新ptail
    p = p->next;
    ptail->next = NULL; //注意！ 更新尾结点的指针域
    delete p;
    p = NULL;
    len--;
    return datatemp;
}

//查找表中是否有特定元素
int LocateItem(int num){
    Node* p = phead->next;
    int i = 0;
    for( i = 0; i < len; ++i){ //遍历寻找
        if(p->data == num){

            return i;
        }
        p = p->next;
    }

    return -1;
}

//表中特定元素之前插入 （包含在第一个元素之前插入）
void InsertBefore(int idx, int num){
    if(idx == 0){ //判断是不是再第一位插入（即替换phead的后继）
        Node* newphead = new Node;
        newphead->next = phead->next; //newphead指向原phead的后继
        phead->next = newphead; //phead指向newphead
        newphead->data = num;
    }
}
```

```
len++;
cout<<"insert Before successfully!"<<endl;
}
else if(idx>=0&&idx<len)//除了插入第一位后的任意位置插入
{
    Node* p = phead->next;
    for(int i=0;i<idx-1;++i){//遍历到指定结点的上一个结点
        p = p->next;
    }

    Node *newnode = new Node;
    newnode->next = NULL;
    newnode->data = num;

    newnode->next = p->next ;//新结点的指针域指向指定结点上一个结点的指针域
    p->next = newnode; //更新指定结点上一个结点的指针域

    len++;
    cout<<"insert Before successfully!"<<endl;
}
else{
    cout<<"input illegal!"<<endl;
}
}

// 表中特定元素之后插入（包含最后一个元素之后插入）
void InsertBack(int idx,int num){
    if(idx==len-1){ //判断是不是在尾结点之后加入
        Node * newPtail = new Node;
        newPtail->next = NULL;

        newPtail->data = num;
        ptail->next = newPtail;//尾结点后面追加
        ptail = newPtail;
        len++;
        cout<<"insert Before successfully!"<<endl;
    }
    else if(idx>=0&&idx<len){
        Node* p = phead->next;
        for(int i=0;i<idx;++i){//遍历到指定结点，在下一个结点插入
            p = p->next;
        }

        Node *newnode = new Node;
        newnode->next = NULL;
        newnode->data = num;

        newnode->next = p->next;
        p->next = newnode;
```

```
        len++;
        cout<<"insert Before successfully!"<<endl;
    }
    else{
        cout<<"input illegal!"<<endl;
    }
}

//删除特定元素 （包含删除第一个元素和最后一个元素）
int DeleteItem(int k){
    int datatemp;
    Node* p;

    if(k<0||k>=len){ //输入不合法
        return -1;
    }
    else if(k==0){ //删除第一个元素
        datatemp = phead->next->data ;
        p = phead->next;
        phead->next = phead->next->next;

        delete p;
        p = NULL;
    }
    else if(k==len-1){ //删除最后一个元素
        p = phead->next;
        for(int i =0;i<len-2;++i){
            p = p->next;
        }
        datatemp = p->next->data;
        ptail = p;
        p = p->next;
        ptail->next = NULL;
        delete p;
        p = NULL;
    }
    else{ //其它
        p= phead->next;
        for(int i =0;i<k-1;++i){
            p = p->next;
        }
        Node* ptemp;
        ptemp = p->next;
        datatemp = ptemp->data;
        p->next = ptemp->next;
```

```
    delete ptemp;
    ptemp=NULL;
    //
    delete删除的只是该指针指向的内存，断开了指向，该指针变成一个野指针，可能指向某个重要变量的地址，需要重置为
}
cout<<"delete successfully!"<<endl;
len--;
return datatemp;
}

//遍历整个链表，并输出
void printAll() {
    Node *p= phead->next;

    //两种方法判断遍历到尾结点

//    while(p!=NULL){
//        cout<<p->data<<" ";
//        p = p->next;
//    }

    for(int i=0;i<len;i++){
        cout<<p->data<<" ";
        p = p->next;
    }
    cout << endl;
}

//递归方式实现反转
Node * ReverseLinkList_1(Node *head){
    if(head==NULL||head->next==NULL){//递归终止条件：找到最后一个节点
        , 每次传的参数时head->next
        return head;
    }
    else{
        Node *newhead = ReverseLinkList_1(head->next);//先从末尾反转，从倒数第二个开始
        head ->next ->next = head;//反转指向
        head->next = NULL;//断开原来的
        return newhead;
    }
}

//非递归方式,依次定义三个指针，时间复杂度O(n)
Node*ReverseLinkList_2(Node*head){
    if(head==NULL||head->next==NULL){
        return head;
    }
    //定义三个紧邻的移动指针
```

```
Node*pre = head;
Node*cur = head->next;
Node*temp = head->next->next;

while(cur){
    temp = cur->next;//记录当前节点cur下一个节点的位置
    cur->next = pre;//反转
    pre = cur;    //移位
    cur = temp;
}
head->next = NULL;//反转后的最后一个结点（即反转前的第一个结点）置空
return pre;
}
};

int main(){
    char ch;
    LinkList a;

    a.Create_List();

    bool flag = 1;
    while(flag){
        cout<<"*****"<<endl;
        cout<<"a(InsertHead),b(InsertTail),c(DeleteHead),d(DeleteTail),e(find),f(insert_before)"<<endl;
        cout<<"g(insert_Back),h(delete),i(visit),j(Recursive_Reverse),k(General_Reverse),l(quit)"<<endl;
        cout<<"*****"<<endl;
        cin>>ch;
        switch(ch){
            case 'a':{
                int ina;
                cout<<"input the num:" ;
                cin>>ina;
                a.InsertHead(ina);
                cout<<"insert successfully!"<<endl;
                break;
            }

            case 'b':{
                int inb;
                cout<<"input the num:";
                cin>>inb;
                a.InsertTail(inb);
                cout<<"insert successfully!"<<endl;
                break;
            }

            case 'c':{
```



```
        cout<<"the head_num you delete is "<<a.DeleteHead()<<endl;
        break;
    }

    case 'd':{
        cout<<"the tail_num you delete is "<<a.DeleteTail()<<endl;
        break;
    }

    case 'e':{
        int k;
        cout<<"input the num:";
        cin>> k;
        int ans = a.LocateItem(k);
        if(ans==-1){
            cout<<"not exist!"<<endl;
        }
        else{
            cout<<"the idx is "<<ans<<endl;
        }
        break;
    }

    case 'f':{
        int idx,value;
        cout<<"input the idx and value:";
        cin>>idx>>value;
        a.InsertBefore(idx,value);
        break;
    }

    case 'g':{
        int idx,value;
        cout<<"input the idx and value:";
        cin>>idx>>value;
        a.InsertBack(idx,value);
        break;
    }

    case 'h':{
        int inh,ans;
        cout<<"input the idx you wanna delete:" ;
        cin>>inh;
        ans = a.DeleteItem(inh);
        if(ans==-1){
            cout<<"delete illegal!"<<endl;
        }
        else{
            cout<<"the value you delete is "<<ans<<endl;
        }
    }
```

```
        break;
    }

    case 'i':{
        a.printAll();
        break;
    }

    case 'j':{
        a.getHead()->next = a.ReverseLinkList_1(a.getHead()->next);
        cout<<"Recursion Reverse successfully!"<<endl;
        a.printAll();
        break;
    }

    case 'k':{
        a.getHead()->next = a.ReverseLinkList_2(a.getHead()->next);
        cout<<"General Reverse successfully!"<<endl;
        a.printAll();
        break;
    }

    case 'l':{
        flag =0;
        break;
    }

    default:{
        cout<<"input illegal! please input again"<<endl;
        break;
    }
}

return 0;
}
```

(3) 实验结果

3.3 项目 3

大致思想：除了反转以外，其他方法除了多处理一个 pre 指针以外，复杂度相同

1. 反转，也需要遍历走一遍（复杂度 $O(n)$ ），但是只需要定义一个 cur 指针遍历，phead 和 ptial 重点处理

(1) 实验步骤

(2) 必要代码

```

create a LinkList(5)
1 2 3 4 5
*****
a(InsertHead),b(InsertTail),c(DeleteHead),d(DeleteTail),e(find),f(insert_before)
g(insert_Back),h(delete),i(visit),j(Recursive_Reverse),k(General_Reverse),l(quit)
*****
Recursion Reverse successfully!
5 4 3 2 1
*****
a(InsertHead),b(InsertTail),c(DeleteHead),d(DeleteTail),e(find),f(insert_before)
g(insert_Back),h(delete),i(visit),j(Recursive_Reverse),k(General_Reverse),l(quit)
*****
k
General Reverse successfully!
1 2 3 4 5
*****
a(InsertHead),b(InsertTail),c(DeleteHead),d(DeleteTail),e(find),f(insert_before)
g(insert_Back),h(delete),i(visit),j(Recursive_Reverse),k(General_Reverse),l(quit)
*****
f
input the idx and value:0 0
insert Before successfully!
*****
a(InsertHead),b(InsertTail),c(DeleteHead),d(DeleteTail),e(find),f(insert_before)
g(insert_Back),h(delete),i(visit),j(Recursive_Reverse),k(General_Reverse),l(quit)
*****
g
input the idx and value:1 9
insert Before successfully!
*****
a(InsertHead),b(InsertTail),c(DeleteHead),d(DeleteTail),e(find),f(insert_before)
g(insert_Back),h(delete),i(visit),j(Recursive_Reverse),k(General_Reverse),l(quit)
*****
l
0 1 9 2 3 4 5
*****
a(InsertHead),b(InsertTail),c(DeleteHead),d(DeleteTail),e(find),f(insert_before)
g(insert_Back),h(delete),i(visit),j(Recursive_Reverse),k(General_Reverse),l(quit)
*****
a
input the num:99
insert successfully!
*****
a(InsertHead),b(InsertTail),c(DeleteHead),d(DeleteTail),e(find),f(insert_before)
g(insert_Back),h(delete),i(visit),j(Recursive_Reverse),k(General_Reverse),l(quit)
*****
l
99 0 1 9 2 3 4 5

```

图 2: 问题二的结果

```

/*
    subject: Data Structure
    To: experiment1_problem3
    Creator: 张丹颖2017011760
    date:2018.10.08
*/

#include<iostream>
using namespace std;

struct DoubleNode{
    int data;
    DoubleNode *next = NULL;
    DoubleNode *pre = NULL;
};

class LinkList{
private:
    int len ;
    DoubleNode *phead = NULL;
    DoubleNode *ptail = NULL;

public:
    LinkList (){//构造
        phead = new DoubleNode;
        ptail = phead;
        phead->next = NULL;
        phead->pre = NULL;
    }

```

```
}

~LinkedList (){//析构
    phead->next = NULL;
    phead->pre = NULL;
    delete phead ;
    phead = NULL;
}

DoubleNode* getHead(){
    return phead;
}

//初始化链表
void Create_List(){
    len = 0;
    DoubleNode *p = phead;
    cout<<"create a LinkedList(5)"<<endl;
    for(int i=0;i<5;++i){
        DoubleNode *pnew = new DoubleNode;
        cin>>pnew->data;
        pnew->next = NULL;

        pnew->pre = p;    //增加前指针
        p->next = pnew;
        p = p->next;
        len++;
    }

    ptail = p;
}

//第一个元素之前插入
void InsertHead(int d){//即在phead后插入一个结点
    DoubleNode* newphead =new DoubleNode;
    //指针域操作
    newphead->next = phead->next;//新结点指向原来的phead的下一个结点
    phead->next->pre = newphead;//原来的phead的下一个结点指向新结点
    newphead->pre = phead; //新结点的前一个结点为phead
    phead->next = newphead;//更新phead的下一个结点为这个新结点

    newphead->data = d;
    len++;
}

//最后一个元素之后插入新数据
void InsertTail(int d){
    DoubleNode * newPtail = new DoubleNode;
    newPtail->next = NULL;
```

```
newPtail->data = d;
ptail->next = newPtail; //直接在结尾ptail后追加新结点
newPtail->pre = ptail; //新尾结点的前一个结点尾原尾结点
ptail = newPtail; //更新ptail
len++;
}

//删除第一位数，即删除phead紧接的那个结点
int DeleteHead(){
    int datatemp;
    DoubleNode* p;
    datatemp = phead->next->data ; //存储删除那个结点的数据域
    p = phead->next;

    phead->next = phead->next->next; //断开phead和它的直接后继
    phead->next->next->pre = phead; //接上前指针
    delete p;
    p = NULL;

    len--;
    return datatemp;
}

//删除尾结点
int DeleteTail(){
    DoubleNode* p = phead->next;
    for(int i = 0; i < len - 2; ++i){ //遍历到尾结点的前一个结点
        p = p->next;
    }
    int datatemp = p->next->data;

    ptail = p; //更新ptail
    p = p->next;
    ptail->next = NULL; //注意！ 更新尾结点的指针域
    delete p;
    p = NULL;
    len--;
    return datatemp;
}

//查找表中是否有特定元素
int LocateItem(int num){
    DoubleNode* p = phead->next;
    int i = 0;
    for( i = 0; i < len; ++i){
        if(p->data == num){

            return i;
        }
    }
}
```

```
p = p->next;
}

return -1;

}

//表中特定元素之前插入 （包含在第一个元素之前插入）
void InsertBefore(int idx,int num){
    if(idx==0) {        //判断是不是再第一位插入（即替换phead的后继）
        DoubleNode* newphead =new DoubleNode;

        newphead->next = phead->next;
        phead->next->pre = newphead;
        newphead->pre = phead;
        phead->next = newphead;

        newphead->data = num;
        len++;
        cout<<"insert Before successfully!"<<endl;
    }
    else if (idx>=0&&idx<len)//除了插入第一位后的任意位置之前插入
    {
        DoubleNode* p = phead->next;
        for(int i=0;i<idx-1;++i){//遍历到指定结点的上一个结点
            p = p->next;
        }

        DoubleNode *newDoubleNode = new DoubleNode;
        newDoubleNode->next =NULL;
        newDoubleNode->data = num;

        newDoubleNode->next = p->next ;//新结点的指针域指向指定结点上一个结点的指针域
        p->next->pre = newDoubleNode;//指定结点的前驱为这个新结点
        p->next = newDoubleNode; //更新指定结点上一个结点的指针域
        newDoubleNode->pre = p;//新结点的前驱为指定结点的前驱

        len++;
        cout<<"insert Before successfully!"<<endl;
    }
    else{
        cout<<"input illegal!"<<endl;
    }
}

// 表中特定元素之后插入 （包含最后一个元素之后插入）
void InsertBack(int idx,int num){
    if(idx==len-1){        //判断是不是在尾结点之后加入
```

```
DoubleNode * newPtail = new DoubleNode;
newPtail->next = NULL;

newPtail->data = num;
ptail->next = newPtail;
newPtail->pre = ptail; //新尾结点的前一个结点尾原尾结点
ptail = newPtail;
len++;
cout<<"insert Before successfully!"<<endl;
}
else if(idx>=0&&idx<len){
    DoubleNode* p = phead->next;
    for(int i=0;i<idx;++i){//遍历到指定结点，在下一个结点插入
        p = p->next;
    }

    DoubleNode *newDoubleNode = new DoubleNode;
    newDoubleNode->next = NULL;
    newDoubleNode->data = num;

    newDoubleNode->next = p->next;
    p->next->pre = newDoubleNode;
    p->next = newDoubleNode;
    newDoubleNode->pre = p;

    len++;
    cout<<"insert Before successfully!"<<endl;
}
else{
    cout<<"input illegal!"<<endl;
}
}

//删除特定元素 （包含删除第一个元素和最后一个元素）
int DeleteItem(int k){
    int datatemp;
    DoubleNode* p;

    if(k<0||k>=len){
        return -1;
    }
    else if(k==0){ //删除第一个元素
        datatemp = phead->next->data ;
        p = phead->next;
        phead->next = phead->next->next;
        phead->next->next->pre = phead; //接上前指针
    }
```

```
        delete p;
        p = NULL;
    }
    else if(k==len-1){//删除最后一个元素
        p = phead->next;
        for(int i =0;i<len-2;++i){
            p = p->next;
        }
        datatemp = p->next->data;
        ptail = p;
        p = p->next;
        ptail->next = NULL;
        delete p;
        p = NULL;
    }
    else{                //其它
        p= phead->next;
        for(int i =0;i<k-1;++i){
            p = p->next;
        }
        DoubleNode* ptemp;
        ptemp = p->next;
        datatemp = ptemp->data;
        p->next = ptemp->next;
        ptemp->next->pre = p;
        delete ptemp;
        ptemp=NULL;
    }
    cout<<"delete successfully!"<<endl;
    len--;
    return datatemp;
}
```

//遍历整个链表，并输出

```
void printAll() {
    DoubleNode *p = phead->next;
```

//两种方法判断遍历到尾结点

```
// while(p!=NULL){
//     cout<<p->data<<" ";
//     p = p->next;
// }
```

```
for(int i=0;i<len;i++){
    cout<<p->data<<" ";
    p = p->next;
}
```



```
        cout << endl;
    }

    //非递归方式,时间复杂度O(n)
    DoubleNode*ReverseLinkList(DoubleNode*head){
        if(head==NULL||head->next==NULL){ //处理空链表的情况
            return head;
        }

        DoubleNode*cur = head,*p;

        //开始反转
        while(cur->next!=NULL){ //遍历到倒数第二个结点,新ptail和phead 单独处理
            p = cur->next; //p记住cur的下一个结点
            cur->next = cur->pre; //当前的结点下一个结点为其前一个结点
            cur->pre = p; //当前结点的前一个结点=当前结点的下一个节点
            cur = cur->pre; //后移
        }

        head->next = NULL; //反转后的最后一个结点(即反转前的第一个结点)置空

        ptail->next = ptail->pre; //更新ptail 的下一个结点
        ptail->pre = phead; //ptail前一个结点为头结点
        ptail = head; //更新尾结点
        return cur;
    }
};

int main(){
    char ch;
    LinkList a;

    a.Create_List();

    bool flag = 1;
    while(flag){
        cout<<"*****" <<endl;
        cout<<"a(InsertHead),b(InsertTail),c(DeleteHead),d(DeleteTail),e(find),f(insert_before)" <<endl;
        cout<<"g(insert_Back),h(delete),i(visit),k(General_Reverse),l(quit)" <<endl;
        cout<<"*****" <<endl;
        cin>>ch;
        switch(ch){
            case 'a':{
                int ina;
                cout<<"input the num:" ;
```

```
        cin>>ina;
        a.InsertHead(ina);
        cout<<"insert successfully!"<<endl;
        break;
    }

    case 'b':{
        int inb;
        cout<<"input the num:";
        cin>>inb;
        a.InsertTail(inb);
        cout<<"insert successfully!"<<endl;
        break;
    }

    case 'c':{
        cout<<"the head_num you delete is "<<a.DeleteHead()<<endl;
        break;
    }

    case 'd':{
        cout<<"the tail_num you delete is "<<a.DeleteTail()<<endl;
        break;
    }

    case 'e':{
        int k;
        cout<<"input the num:";
        cin>> k;
        int ans = a.LocateItem(k);
        if(ans==-1){
            cout<<"not exist!"<<endl;
        }
        else{
            cout<<"the idx is "<<ans<<endl;
        }
        break;
    }

    case 'f':{
        int idx,value;
        cout<<"input the idx and value:";
        cin>>idx>>value;
        a.InsertBefore(idx,value);
        break;
    }

    case 'g':{
        int idx,value;
        cout<<"input the idx and value:";
```

```
        cin>>idx>>value;
        a.InsertBack(idx,value);
        break;
    }

    case 'h':{
        int inh,ans;
        cout<<"input the idx you wanna delete:" ;
        cin>>inh;
        ans = a.DeleteItem(inh);
        if(ans!=-1){
            cout<<"delete illegal!"<<endl;
        }
        else{
            cout<<"the value you delete is "<<ans<<endl;
        }
        break;
    }

    case 'i':{
        a.printAll();
        break;
    }

    case 'k':{
        a.getHead()->next= a.ReverseLinkedList(a.getHead()->next);
        cout<<"General Reverse successfully!"<<endl;
        a.printAll();
        break;
    }

    case 'l':{
        flag =0;
        break;
    }

    default:{
        cout<<"input illegal! please input again"<<endl;
        break;
    }
}

return 0;
}
```

```

create a LinkList(5)
1 2 3 4 5
*****
a(InsertHead), b(InsertTail), c(DeleteHead), d(DeleteTail), e(find), f(insert_before)
g(insert_Back), h(delete), i(visit), k(General Reverse), l(quit)
*****
k
General Reverse successfully!
5 4 3 2 1
*****
a(InsertHead), b(InsertTail), c(DeleteHead), d(DeleteTail), e(find), f(insert_before)
g(insert_Back), h(delete), i(visit), k(General Reverse), l(quit)
*****
a
input the num:55
insert successfully!
*****
a(InsertHead), b(InsertTail), c(DeleteHead), d(DeleteTail), e(find), f(insert_before)
g(insert_Back), h(delete), i(visit), k(General Reverse), l(quit)
*****
i
55 5 4 3 2 1
*****
a(InsertHead), b(InsertTail), c(DeleteHead), d(DeleteTail), e(find), f(insert_before)
g(insert_Back), h(delete), i(visit), k(General Reverse), l(quit)
*****
f
input the idx and value:2 99
insert Before successfully!
*****
a(InsertHead), b(InsertTail), c(DeleteHead), d(DeleteTail), e(find), f(insert_before)
g(insert_Back), h(delete), i(visit), k(General Reverse), l(quit)
*****
i
55 5 99 4 3 2 1
*****
a(InsertHead), b(InsertTail), c(DeleteHead), d(DeleteTail), e(find), f(insert_before)
g(insert_Back), h(delete), i(visit), k(General Reverse), l(quit)
*****
g
input the idx and value:0 0
insert Before successfully!
*****
a(InsertHead), b(InsertTail), c(DeleteHead), d(DeleteTail), e(find), f(insert_before)
g(insert_Back), h(delete), i(visit), k(General Reverse), l(quit)
*****
i
55 0 5 99 4 3 2 1
*****

```

图 3: 问题三的结果

(3) 实验结果

3.4 项目 4.1

大致思想：TransStringToArray 函数将 String 中的数字摘取存入数组中，Plus 函数计算相加

1. TransStringToArray 函数，用到 while 循环，循环次数与字符串长度有关（复杂度 $O(n)$ ）；
2. Plus 函数，执行相加的次数，也与 String 长度相关（复杂度 $O(n)$ ）

(1) 实验步骤

(2) 必要代码

```

/*
subject: Data Structure
To: experiment1_problem4.1
Creator: 张丹颖2017011760
date:2018.10.0
*/

#include<iostream>
#include<string>

```

```
using namespace std;

//定义函数将string转换存储有效数字的数组
int TransStringToArray(string a ,int *orig){

    int length=0,flag=0;

    int sum = 0;
    while(length++<(int)a.size()){//string类的size()函数返回的是unsigned
        integer(无符号数)类型。而用在for循环时，
        //正常不会出错，但作为判断条件时，当s.length()等于0时，s.length()-1 不等于
        -1
        //所以这里用（int）强制转换

        //将字符转化为对应大小的数字
        if(a[length-1]!='.'){//遍历到有效数字，从后往前遍历
            sum*=10; //sum 确定该数是处于个位，十位还是百位等
            int temp = a[length-1]-'0';//字符数字转化为int型数字
            sum+=temp;
        }
        else{
            //遇到分割标记“.”或者空格，代表该数字由字符转化为数字完毕，可以将sum存入数组中
            orig[flag++] = sum;
            sum = 0;//sum 清零
        }
    }

    orig[flag++] = sum;//第一个没有进入else,也要注意存入
    return flag;
}

//定义Plus函数
void Plus(string m,string n){
    int m_numLength ,n_numLength;
    int carry= 0,i=0,m_num[100],n_num[100];//carry进位，数组存数字
    //25个质数存储在数组中，用空间换时间
    const int primeNumber[25] ={2,3,5,7,11,13,17,19,23,29,
        31,37,41,43,47,53,59,61,67,71,
        73,79,83,89,97};

    if(m.size()<n.size()){//将被加数规定为较长的string
        string temp = m;
        m = n;
        n = temp;
    }

    m_numLength = TransStringToArray(m,m_num);
    n_numLength = TransStringToArray(n,n_num);
    int t = m_numLength;
```

```
while(m_numLength>0){
    int a,b,sum;
    a = m_num[m_numLength-1]; //取出个位数

    if(n_numLength>0){
        b = n_num[n_numLength-1];
    }
    else{
        b = 0;
    }

    sum = a+b+carry;

    if(sum>=primeNumber[i]){ //需要进位
        m_num[m_numLength-1] = sum%primeNumber[i]; //大的数组的最后一位更新为进位后的余数
        carry = 1;
    }
    else{
        m_num[m_numLength-1] = sum; //不需进位，直接更新
        carry = 0;
    }

    m_numLength--; //移位遍历
    n_numLength--;
    i++;
}

if(carry ==1){ //注意！！考虑最后一位数进位的问题，仅需要在输出时补上“1,”
    cout<<"1,";
}

m_numLength = t ;
int j = 0;
while (m_num[j] == 0 && !carry)
    j++; //无效零（本来就有无效零或者不是需要补上“1,”的情况），不输出
for(;j<m_numLength-1;++j){
    cout<<m_num[j]<<',';
}
cout<<m_num[m_numLength-1]<<endl; //标准化格式，最后一位单独输出
}

int main(){
    string a,b;
    while(1){
        cin>>a>>b;
```

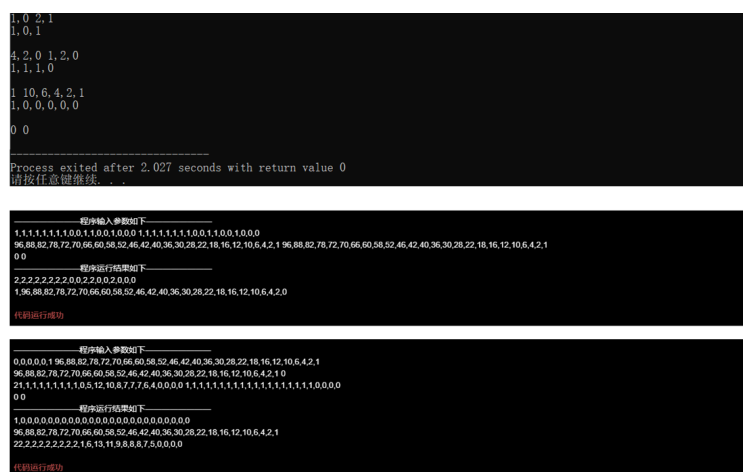
```

if(a=="0"&&b=="0"){//a and b都是零时结束!
    break;
}
else{
    Plus(a,b);
}

}
return 0;
}

```

(3) 实验结果



```

1 0 2 1
1 0 1
4 2 0 1 2 0
1 1 1 0
1 10 6 4 2 1
1 0 0 0 0 0
0 0
Process exited after 2.027 seconds with return value 0
请按任意键继续. . .

程序输入参数如下
1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 1 1 1 1 1 0 0 1 0 0 1 0 0 0
96 88 82 78 72 70 66 60 58 52 46 42 40 36 30 28 22 18 16 12 10 6 4 2 1
0 0
程序运行结果如下
2 2 2 2 2 2 0 0 2 2 0 0 0 0
1 96 88 82 78 72 70 66 60 58 52 46 42 40 36 30 28 22 18 16 12 10 6 4 2 0
代码运行成功

程序输入参数如下
0 0 0 0 1 96 88 82 78 72 70 66 60 58 52 46 42 40 36 30 28 22 18 16 12 10 6 4 2 1
96 88 82 78 72 70 66 60 58 52 46 42 40 36 30 28 22 18 16 12 10 6 4 2 1 0
21 1 1 1 1 1 1 0 5 12 10 8 7 7 6 4 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
0 0
程序运行结果如下
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
96 88 82 78 72 70 66 60 58 52 46 42 40 36 30 28 22 18 16 12 10 6 4 2 1
22 2 2 2 2 2 2 2 1 6 13 11 9 8 8 8 7 5 0 0 0 0
代码运行成功

```

图 4: 问题 4.1 的结果

3.5 项目 4.2

大致思想：对每个读入的成绩，在链表中，则该成绩出现次数 ++，否则，链表增加一个新结点

1. 复杂度 ($M*N$)，其中 M 代表要读入数字的规模， N 代表已存在的链表结点规模（由于每次都要遍历，所以复杂度很高）

(1) 实验步骤

(2) 必要代码

```

/*
subject: Data Structure
To: experiment1_problem4.2
Creator: 张丹颖2017011760
date:2018.10.08
*/

```

```
#include<iostream>
using namespace std;

struct scoreNode{    //定义链表，存储这个数的相关信息
    int data ;        //这个数的数值
    int times;        //这个数出现的次数
    scoreNode *next; //用链表这一结构存储这些不同的数字
};

int main(){
    int N ,find_score;
    cin>>N;
    while(N!=0){
        scoreNode * phead = new scoreNode;
        phead->next = NULL;
        phead->data = -1;
        phead->times = 1;
        scoreNode *p,*ptail;

        while(N-->0){ //对于每个测试用例
            int temp,flag=0 ;
            cin>>temp;
            p = phead;
            while(p!=NULL){
                if(temp==p->data){//对于每个输出的成绩，检查在链表中是否有相同的成绩
                    p->times++;//有，出现的次数++
                    flag++;
                }
                ptail = p;
                p = p->next;
            }
            if(flag == 0){ //遍历完链表后发现都没有这个数，则在链表后面追加新成绩结点
                scoreNode* ScoreBox = new scoreNode;
                ptail->next = ScoreBox;
                ScoreBox->data = temp;;
                ScoreBox->next = NULL;
                ScoreBox->times = 1;
                ptail = ScoreBox;
            }
        }

        cin>>find_score;//输入目标成绩
        p = phead->next;
        int star=0;
        while(p!=NULL){
            if(find_score==p->data){ //遍历寻找
                cout<<p->times<<endl;//找到就输入对应输入的个数
                star++;
                break;
            }
        }
    }
}
```



```

    p = p->next;
}

if(star==0){//若找不到，就输出0
    cout<<star<<endl;
}
cin>>N;
}

return 0;
}

```

(3) 实验结果



图 5: 问题 4.2 的结果

3.6 项目 4.3

大致思想：plus 函数中，首先去掉无效零，逐位将字符转换为数字再相加，满 10 进位，最后一次若要进位别忘了加上；

1. plus 函数转换相加的复杂度取决于字符串长度（复杂度 $O(n)$ ）；

(1) 实验步骤

(2) 必要代码

```
/*
    subject: Data Structure
    To: experiment1_problem4.3
    Creator: 张丹颖2017011760
    date:2018.10.08
*/

#include<iostream>
#include<string>
using namespace std;

string Plus(string m,string n){
    int m_length = m.size(),n_length = n.size(),carry= 0;

    //对于少数无效零的数字，首先外层用if进行简单判断，若首位不是无效零，不需进入while,省时

    //用erase函数去掉无效零
    if(m[0]=='0'){
        while(m_length-->0){
            if(m[0]=='0'){
                m.erase(0,1);
            }
            if(m[0]!='0'){
                break;
            }
        }
    }

    if(n[0]=='0'){
        while(n_length-->0){
            if(n[0]=='0'){
                n.erase(0,1);
            }
            if(n[0]!='0'){
                break;
            }
        }
    }

    //人为规定，被加数为较长的字符串
    if(m.size()<n.size()){
        string temp = m;
        m = n;
        n = temp;
    }
}
```

```
}

m_length = m.size();
n_length = n.size();

while(m_length>0){
    int a,b,sum;
    a = m[m_length-1]-'0';//字符转换为数字

    if(n_length>0){
        b = n[n_length-1] - '0';
    }
    else{
        b = 0;
    }

    sum = a+b+carry;

    if(sum>=10){
        m[m_length-1] = '0' +sum%10;//相加满10进位
        carry = 1;
    }
    else{
        m[m_length-1] = '0' +sum;
        carry = 0;
    }

    m_length--;
    n_length--;
}

if(carry ==1){//主语最后两个数相加可能进位
    m = "1" + m;
}

return m;
}

int main(){
    int times ,i = 1;
    cin>>times;
    //格式化输出
    while(times-->0){
        string a,b;
        cin>>a>>b;
        cout<<"Case "<<i<<":"<<endl;
        cout<<a<<" + "<<b<<" = "<< Plus(a,b)<<endl;
        i++;
    }
}
```

(3) 实验结果

图 6: 问题 4.3 的结果

大致思想: Polynomial 函数, 对于系数数组每个数, 都和其对应的 x 的几次方相乘, 再相加, 返回结果, 不要用 pow, 可以直接利用上一次计算的结果;

1. (复杂度 $O(M*N)$), N 代表 x 最高项指数, M 代表不同 x 的取值

- (1) 实验步骤
- (2) 必要代码

35

```
Creator: 张丹颖2017011760
date:2018.10.08
*/

#include<iostream>
using namespace std;

double Polynomail(double *a,double x,int n){
    double sum = a[0];
    double multy = 1;
    for(int i=1;i<n+1;++i){//对于系数数组每个数，都和其对应的x的几次方相乘，再相加
        multy*=x;    //不用pow函数，由于x指数递增，本次直接上一次的x基础上再乘以x
        sum+=a[i]*multy;
    }
    return sum;
}

int main(){
    int T,N,M;
    cin>>T;
    while(T-->0){
        double coef[1001],x[10];
        cin>>N;
        for(int i=0;i<N+1;++i){//输入指数
            cin>>coef[i];
        }
        cin>>M;
        for(int i=0;i<M;++i){//输入要计算的不同的x的值
            cin>>x[i];
        }

        for(int i=0;i<M;++i){
            printf("%.10e\n",Polynomail(coef,x[i],N));//%.10e代表精确到小数点后10位，用科学记数法表示结果
        }

    }

    return 0;
}
```

(3) 实验结果

四、实验总结

4.1 优点

1. 熟练掌握顺序存储和链式存储的线性表的增删改查等基本操作；
2. 掌握单链表和双向链表的反转，体会特殊结点特殊处理的思想；



3. 学会增加程序的鲁棒性；
4. 熟悉计算函数的复杂度；
5. 体会一个问题的多种解法，通过比较复杂度取优，以及用空间换时间；
6. 意识到线性表可以解决实际问题，如计算多项式，进位换算，查重；

4.2 不足

1. 算法思想有待加强，逻辑思维能力不足，考虑问题不全面；