《数据结构》实验报告

姓名：高心阳　学号：084623237

班级：计算机 232　日期：2024/09/11

程序名：线性表应用实验：集合运算的实现

一、上机实验的问题和要求：

设计一个程序来实现集合（元素类型为 int）的运算：

1. 生成两个存有集合的单链表 A(4,1,2,3)和 B(5,1,3,6)。
2. 实现 A 与 B 的并、交、差及判断相等的功能，并设计主函数对各功能进行测试。
3. （选做）请修改顺序表或单链表的类，做到表中元素按值大小升序存放，再完成集合并、交、差及判断相等运算，总结其不同。

二、算法设计和基本思想描述：

（分析问题中的数据及数据元素之间的关系，确定选择何种逻辑结构；再根据操作特点分析选择何种存储结构，最后再具体分析每个功能，看需要用到哪些基本操作，如何用基本操作组合出该功能。）

LinkList:

1. int Length():返回链表长度
2. int Empty():判断链表是否为空
3. DataType Get(int i):根据值查抄数据位置（可用于判断是否存在某数据）
4. int Equal(LinkList<DataType> x):判断是否相等（先判长度，再判值，忽略顺序）
5. LinkList<DataType> Union(LinkList<DataType> x):并集（两层循环增加，判重防止增加重复值）
6. LinkList<DataType> Intersection(LinkList<DataType> x):交集（两层循环，判重加入）
7. LinkList<DataType> Difference(LinkList<DataType> x):差集（单层循环，遍历自身链表，再参数链表中通过 Get（）查询）
8. LinkList(const LinkList<DataType>& other):拷贝构造函数（接收函数返回值）

Ordered_LinkList：

1. void Insert(DataType x):重写插入函数，去除指定下标
2. int Locate(DataType x):重写值查询操作，排序后可简化复杂度
3. int Equal(Ordered_LinkList<DataType> x):重写判等操作，排序后可简化复杂度
4. Ordered_LinkList<DataType> Union(Ordered_LinkList<DataType> x):重写并集操作，排序后通过双指针简化复杂度
5. Ordered_LinkList<DataType> Intersection(Ordered_LinkList<DataType> x):重写交集操作，排序后通过双指针简化复杂度

三、源程序及注释（代码直接复制到下面，标好文件名）：

LinkList.h

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

```
template<class DataType>
struct Node {
    DataType data;
    Node<DataType>* next;
};

template<class DataType>
class LinkList {
private:
    Node<DataType>* first;
public:
    LinkList();
    LinkList(DataType a[], int n);
    ~LinkList();
    int Locate(DataType x);
    void Insert(int i, DataType x);
    DataType Delete(int i);
    void PrintList();
    int Length();
    int Empty();
    DataType Get(int i);
    int Equal(LinkList<DataType> x);
    LinkList<DataType> Union(LinkList<DataType> x);//并集
    LinkList<DataType> Intersection(LinkList<DataType> x);//交集
    LinkList<DataType> Difference(LinkList<DataType> x);//差集
    LinkList(const LinkList<DataType>& other);
};

template<class DataType>
LinkList<DataType>::LinkList() {
    first = new Node<DataType>;
    first->next = NULL;
}

template<class DataType>
LinkList<DataType>::LinkList(DataType a[], int n) {
    Node<DataType>* r, * s;
    first = new Node<DataType>;
    r = first;
    for (int i = 0; i < n; i++) {
        s = new Node<DataType>;
        s->data = a[i];
        r->next = s;
        r = s;
```

```
    }
    r->next = NULL;
}

template<class DataType>
LinkList<DataType>::~LinkList() {
    Node<DataType>* q = NULL;
    while (first != NULL) {
        q = first;
        first = first->next;
        delete q;
    }
}

template<class DataType>
int LinkList<DataType>::Locate(DataType x) {
    Node<DataType>* q = first->next;
    int count = 1;
    while (q != NULL) {
        if (q->data == x)return count;
        q = q->next;
        count++;
    }
    return 0;
}

template<class DataType>
void LinkList<DataType>::Insert(int i, DataType x) {
    Node<DataType>* p = first, * s = NULL;
    int count = 0;
    while (p != NULL && count < i - 1) {
        p = p->next;
        count++;
    }
    if (p == NULL)throw"非法位置";
    else {
        s = new Node<DataType>;
        s->data = x;
        s->next = p->next;
        p->next = s;
    }

}
```

```
template<class DataType>
DataType LinkList<DataType>::Delete(int i) {
    Node<DataType>* p = first, * q = NULL;
    DataType x;
    int count = 0;
    while (p != NULL && count < i - 1) {
        p = p->next;
        count++;
    }
    if (p == NULL || p->next == NULL)throw"位置非法";
    else {
        q = p->next;
        x = q->data;
        p->next = q->next;
        delete q;
        return x;
    }
}

template<class DataType>
void LinkList<DataType>::PrintList() {
    Node<DataType>* p = first->next;
    while (p != NULL) {
        cout << p->data << " ";
        p = p->next;
    }
    cout << endl;
}

template<class DataType>
int LinkList<DataType>::Length() {
    Node<DataType>* p = first->next;
    int length = 0;
    while (p != NULL) {
        length++;
        p = p->next;
    }
    return length;
}

template<class DataType>
int LinkList<DataType>::Empty()
{
    if (first->next == NULL)return 1;
```

```
    return 0;
}

template<class DataType>
DataType LinkList<DataType>::Get(int i) {
    Node<DataType>* p = first->next;
    int k = 1;
    while (p != NULL && k < i) {
        p = p->next;
        k++;
    }
    if (k == i)return p->data;
}

template<class DataType>
int LinkList<DataType>::Equal(LinkList<DataType> Compare) {
    int len = Length(), clen = Compare.Length();
    if (len != clen)return 0;
    for (int i = 1; i <= len; i++) {
        DataType aelement = Get(i);
        for (int j = 1; j <= clen; j++) {
            DataType celement = Compare.Get(j);
            if (aelement == celement)break;
            if (j == clen)return 0;
        }
    }
    return 1;
}

template<class DataType>
LinkList<DataType> LinkList<DataType>::Union(LinkList<DataType> x)//并集
{
    LinkList<DataType> C;
    int len = 1;
    Node<DataType>* p = first->next;
    while (p != NULL) {
        C.Insert(len, p->data);
        p = p->next;
        len++;
    }
    p = x.first->next;
    while (p != NULL) {
        DataType xelement = p->data;
        if (Locate(xelement) == 0) {
```

```
            C.Insert(len, xelement);
            len++;
        }
        p = p->next;
    }
    return C;
}

template<class DataType>
LinkList<DataType> LinkList<DataType>::Intersection(LinkList<DataType> x)//交集
{
    LinkList<DataType> C;
    int len = 1;
    Node<DataType>* p = first->next;
    while (p != NULL) {
        if (x.Locate(p->data) != 0) {
            C.Insert(len, p->data);
            len++;
        }
        p = p->next;
    }
    return C;
}

template<class DataType>
LinkList<DataType> LinkList<DataType>::Difference(LinkList<DataType> x)//差集
{
    LinkList<DataType> C;
    int len = 1;
    Node<DataType>* p = first->next;
    while (p != NULL) {
        if (x.Locate(p->data) == 0) {
            C.Insert(len, p->data);
            len++;
        }
        p = p->next;
    }
    return C;
}

template<class DataType>
LinkList<DataType>::LinkList(const LinkList<DataType>& other) {
    if (other.first != NULL) {
        first = new Node<DataType>;
```

```
        first->data = other.first->data;
        Node<DataType>* p = first;
        Node<DataType>* otherP = other.first->next;
        while (otherP != NULL) {
            p->next = new Node<DataType>;
            p = p->next;
            p->data = otherP->data;
            otherP = otherP->next;
        }
        p->next = NULL;
    }
    else {
        first = NULL;
    }
}
```

main.cpp

```
#include <iostream>
#include "LinkList.h"

using namespace std;

int main() {
    int a[4] = { 4,1,2,3 }, b[4] = { 5,1,3,6 };

    LinkList<int> A(a, 4), B(b, 4);

    cout << "A:";
    A.PrintList();
    cout << "B:";
    B.PrintList();

    LinkList<int> C(A.Union(B));
    cout << "并集 C:";
    C.PrintList();

    LinkList<int> D(A.Intersection(B));
    cout << "交集 D:";
    D.PrintList();

    LinkList<int> E(A.Difference(B));
    cout << "差集 E:";
    E.PrintList();
```

```
    return 0;
}
```

Ordered_LinkList.h

```
#pragma once
#include <iostream>
#include <algorithm>
using namespace std;
/*
    name: Ordered_LinkList.h

    creat_time: 2024/9/11

    based_files: LinkList.h(2024/9/11)

    change:
        add function: void Insert(DataType x)
        delete funtion: void Insert(int i,DataType x)
        rebuild function: int Locate(DataType x)
        rebuild function: int Equal(Ordered_LinkList<DataType> x);
        rebuild function: Ordered_LinkList<DataType> Union(Ordered_LinkList<DataType> x)
        rebuild function: Ordered_LinkList<DataType>
Intersection(Ordered_LinkList<DataType> x)
*/

template<class DataType>
struct Node {
    DataType data;
    Node<DataType>* next;
};

template<class DataType>
class Ordered_LinkList {
private:
    Node<DataType>* first;
public:
    Ordered_LinkList();
    Ordered_LinkList(DataType a[], int n);
    Ordered_LinkList(const Ordered_LinkList<DataType>& other);
    ~Ordered_LinkList();
    int Locate(DataType x);
    void Insert(DataType x);
    DataType Delete(int i);
```

```
    void PrintList();
    int Length();
    int Empty();
    DataType Get(int i);
    int Equal(Ordered_LinkList<DataType> x);
    Ordered_LinkList<DataType> Union(Ordered_LinkList<DataType> x);//并集
    Ordered_LinkList<DataType> Intersection(Ordered_LinkList<DataType> x);//交集
    Ordered_LinkList<DataType> Difference(Ordered_LinkList<DataType> x);//差集

};

template<class DataType>
Ordered_LinkList<DataType>::Ordered_LinkList() {
    first = new Node<DataType>;
    first->next = NULL;
}

template<class DataType>
Ordered_LinkList<DataType>::Ordered_LinkList(DataType a[], int n) {
    sort(a, a + n);
    Node<DataType>* r, * s;
    first = new Node<DataType>;
    r = first;
    for (int i = 0; i < n; i++) {
        s = new Node<DataType>;
        s->data = a[i];
        r->next = s;
        r = s;
    }
    r->next = NULL;
}

template<class DataType>
Ordered_LinkList<DataType>::~Ordered_LinkList() {
    Node<DataType>* q = NULL;
    while (first != NULL) {
        q = first;
        first = first->next;
        delete q;
    }
}

template<class DataType>
int Ordered_LinkList<DataType>::Locate(DataType x) {
```

```
    Node<DataType>* q = first->next;
    int count = 1;
    while (q != NULL) {
        if (q->data > x)return 0;
        if (q->data == x)return count;
        q = q->next;
        count++;
    }
    return 0;
}

template<class DataType>
void Ordered_LinkList<DataType>::Insert(DataType x) {
    Node<DataType>* p = first, * s = NULL;
    while (p->next != NULL) {
        if (p->next->data > x)break;
        p = p->next;
    }
    s = new Node<DataType>;
    s->data = x;
    s->next = p->next;
    p->next = s;
}

template<class DataType>
DataType Ordered_LinkList<DataType>::Delete(int i) {
    Node<DataType>* p = first, * q = NULL;
    DataType x;
    int count = 0;
    while (p != NULL && count < i - 1) {
        p = p->next;
        count++;
    }
    if (p == NULL || p->next == NULL)throw"位置非法";
    else {
        q = p->next;
        x = q->data;
        p->next = q->next;
        delete q;
        return x;
    }
}

template<class DataType>
```

```
void Ordered_LinkList<DataType>::PrintList() {
    Node<DataType>* p = first->next;
    while (p != NULL) {
        cout << p->data << " ";
        p = p->next;
    }
    cout << endl;
}

template<class DataType>
int Ordered_LinkList<DataType>::Length() {
    Node<DataType>* p = first->next;
    int length = 0;
    while (p != NULL) {
        length++;
        p = p->next;
    }
    return length;
}

template<class DataType>
int Ordered_LinkList<DataType>::Empty()
{
    if (first->next == NULL)return 1;
    return 0;
}

template<class DataType>
DataType Ordered_LinkList<DataType>::Get(int i) {
    Node<DataType>* p = first->next;
    int k = 1;
    while (p != NULL && k < i) {
        p = p->next;
        k++;
    }
    if (k == i)return p->data;
}

template<class DataType>
int Ordered_LinkList<DataType>::Equal(Ordered_LinkList<DataType> Compare) {
    int len = Length(), clen = Compare.Length();
    if (len != clen)return 0;
    for (int i = 1; i <= len; i++) {
        DataType aelement = Get(i);
```

```
        DataType celement = Compare.Get(i);
        if (aelement != celement)return 0;
    }
    return 1;
}

template<class DataType>
Ordered_LinkList<DataType> Ordered_LinkList<DataType>::Union(Ordered_LinkList<DataType>
x)//并集
{
    Ordered_LinkList<DataType> C;
    Node<DataType>* p = first->next, * xp = x.first->next;
    while (p != NULL && xp != NULL) {
        if (p->data > xp->data) {
            C.Insert(xp->data);
            xp = xp->next;
        }
        else if (p->data < xp->data) {
            C.Insert(p->data);
            p = p->next;
        }
        else {
            C.Insert(p->data);
            p = p->next;
            xp = xp->next;
        }
    }
    if (p == NULL)
        while (xp != NULL) {
            C.Insert(xp->data);
            xp = xp->next;
        }
    if (xp == NULL)
        while (p != NULL) {
            C.Insert(p->data);
            p = p->next;
        }
    return C;
}

template<class DataType>
Ordered_LinkList<DataType>
Ordered_LinkList<DataType>::Intersection(Ordered_LinkList<DataType> x)//交集
{
```

```
    Ordered_LinkList<DataType> C;
    Node<DataType>* p = first->next, * xp = x.first->next;
    while (p->next != NULL) {
        while (p->data != xp->data) {
            if (p->data < xp->data)p = p->next;
            else xp = xp->next;
        }
        if (p->next == NULL || xp->next == NULL)break;
        if (x.Locate(p->data) != 0) {
            C.Insert(p->data);
        }
        p = p->next;
    }
    return C;
}

template<class DataType>
Ordered_LinkList<DataType>
Ordered_LinkList<DataType>::Difference(Ordered_LinkList<DataType> x)//差集
{
    Ordered_LinkList<DataType> C;
    Node<DataType>* p = first->next;
    while (p != NULL) {
        if (x.Locate(p->data) == 0) {
            C.Insert(p->data);
        }
        p = p->next;
    }
    return C;
}

template<class DataType>
Ordered_LinkList<DataType>::Ordered_LinkList(const Ordered_LinkList<DataType>& other) {
    if (other.first != NULL) {
        first = new Node<DataType>;
        first->data = other.first->data;
        Node<DataType>* p = first;
        Node<DataType>* otherP = other.first->next;
        while (otherP != NULL) {
            p->next = new Node<DataType>;
            p = p->next;
            p->data = otherP->data;
            otherP = otherP->next;
        }
```

```
        p->next = NULL;
    }
    else {
        first = NULL;
    }
}
```

Omain.cpp

```
#include <iostream>
#include "Ordered_LinkList.h"

using namespace std;

int main() {
    int a[4] = { 4,1,2,3 }, b[4] = { 5,1,3,6 };

    Ordered_LinkList<int> OA(a, 4), OB(b, 4);

    cout << "OA:";
    OA.PrintList();
    cout << "OB:";
    OB.PrintList();

    Ordered_LinkList<int> OC(OA.Union(OB));
    cout << "并集 OC:";
    OC.PrintList();

    Ordered_LinkList<int> OD(OA.Intersection(OB));
    cout << "交集 OD:";
    OD.PrintList();

    Ordered_LinkList<int> OE(OA.Difference(OB));
    cout << "差集 OE:";
    OE.PrintList();

    return 0;
}
```

四、运行输出结果：

（可以将运行结果抓图贴至此处）

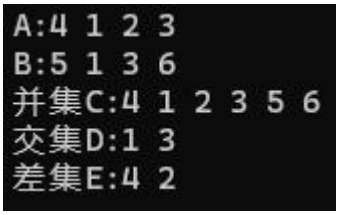

图 1 main.cpp 程序运行结果

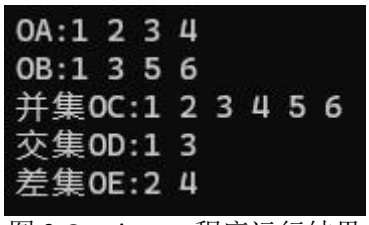

图 2 Omain.cpp 程序运行结果

五、调试和运行程序过程中产生的问题及采取的措施：

1. LinkList<DataType>&类型函数返回值赋值失败（未对符号=重载）
   措施：增加拷贝构造函数，通过构造方式赋值

六、分析主要算法的时间复杂度，如果可以优化，提出优化方案：

LinkList:

1. int Length(): O(1)
2. int Empty(): O(1)
3. DataType Get(int i): O(n)
4. int Equal(LinkList<DataType> x):O(n^2)
5. LinkList<DataType> Union(LinkList<DataType> x):O(n+m)
6. LinkList<DataType> Intersection(LinkList<DataType> x):O(n)
7. LinkList<DataType> Difference(LinkList<DataType> x):O(n)
8. LinkList(const LinkList<DataType>& other):O(n)

Ordered_LinkList：

1. void Insert(DataType x):int Locate(DataType x):O(n)
2. int Equal(Ordered_LinkList<DataType> x):O(n)
3. Ordered_LinkList<DataType> Union(Ordered_LinkList<DataType> x):O(n+m)
4. Ordered_LinkList<DataType> Intersection(Ordered_LinkList<DataType> x):O(n)

七、对数据结构教学的意见和建议：

无