**实验6 进阶版：链表类+文件读写**

**姓名\_\_李飞飞\_\_\_班级\_\_软工2206\_\_\_学号\_202105710309\_\_**

* **请阅读此说明：实验6满分120分，其中实验一含20分附加得分；做完实验后请按要求将代码和截图贴入该文档。然后将此文档、源代码文件（.hpp, .cpp）打包上传到学习通。**

**1、（总分50+20分）面向对象的链表类可以设计成：**

**typedef int dataType;**

**class node{**

**public:**

**node(dataType d,node\* ptr=nullptr) {data=d; next=ptr; }**

**dataType data;**

**node\* next;**

**};**

class linked\_list{

public:

linked\_list();

linked\_list(const linked\_list& other);

linked\_list& operator=(const linked\_list& right);

int list\_size() const;

//集合并交叉也可以考虑设计为类外函数

//linked\_list operator+(const linked\_list& right); //链表集合并

//linked\_list operator-(const linked\_list& right);//链表集合差

//linked\_list intersectionSet(const linked\_list& right); //链表集合交

node\* find(dataType value); //查找

bool find(dataType value,node\*& pre,node\*&p);

//考虑修改：bool find(dataType value,node\*& pre,node\*&p);

//找到返回真：p为目标点，pre为前驱点; 找不到返回假：p和pre均为nullptr

void add\_front(dataType value); //添加到首

void add\_tail(dataType value);//添加到尾

void add\_pos\_after(dataType value, node \* pos); //添加到指定位置之后

void add\_pos\_before(dataType value,node \*pos);//添加到指定位置之前

void Delete(dataType value);//删除指定值

void delete\_pos\_after(node\* pos);//删除指定位置之后

void delete\_pos\_before(node\* pos);//删除指定位置之前

//void delete\_all (dataType value);//删除所有重复值

void reverse();//逆置链表

void sort();//升序排列当前链表

void display();//遍历链表

~linked\_list();

private:

node\* head,\*tail;

int size; //若使用哨兵链，则可使用哨兵的数据域代替size

}; //可以考虑设计带哨兵的链表，哨兵结点的信息可以代替size 存储表中结点的个数

* **实验要求：红色部分的成员函数我们在课堂已经讨论过他们的实现，请你完成蓝色标记的成员函数。并通过给出的测试程序。**
* **附加：绿色部分的4个成员函数为可选功能，每完成一个+5分。**
* **测试程序：**

**//测试程序：**

**#include<iostream>  
#include “linkedList.hpp”**

**Int main()**

**{ linked\_list a1,a2, b，c;**

**dataType data;**

**//正向和逆向建链测试**

**//输入2 ,6, 7, 3, 5, 9,12, 4 ,0**

**while(cin>>data){**

**if(data==0) break; //输入0结束**

**a1.add\_front(data);**

**a2.add\_tail(data);**

**}**

**a1.display();**

**a2.display();**

**//链表转置测试**

**//输入2 ,16, 3, 8, 15, 4, 9, 7 ,0**

**while(cin>>data) {**

**if(data==0) break; //输入0结束**

**b.add\_tail(data);**

**}**

**b.display();**

**b.reverse();**

**b.display();**

**//c=a1+b; //测试集合并**

**//c.display();**

**//c=a1-b; //测试集合差**

**//c.display();**

**//c=a1.intersectionSet(b); //测试集合交**

**//c.display();**

**a1.sort(); //测试升序排序**

**a1.display();**

**//思考需要降序排序如何做？**

**b.add\_tail(16); b.add\_tail(3); b.display();**

**b.delete\_repeat(); b.display();**

**node\* pos=b.find(15);**

**b. add\_pos\_after(18,pos); b.display();**

**b.add\_pos\_before(23,pos); b.display();**

**b.delete\_pos\_after(pos); b.display();**

**b.delete\_pos\_before(pos); b.display();**

**b.Delete(8); b.display();**

**return 0;**

**}**

**实验解答：**

**❶黏贴链表类的实现代码：（标注你使用了基本单向链表 or带哨兵的单向链表？）**

**用了基本单向链表**

#pragma once

#include<iostream>

#include"linkedList.hpp"

using namespace std;

linked\_list::linked\_list()

{

    head = tail = nullptr; size = 0;

}   //初始化构造函数

linked\_list::linked\_list(const linked\_list& other)

{

    head = tail = nullptr;

    size = other.size;

    node\* p = other.head;

    while (p != nullptr)

    {

        add\_tail(p->data);

        p = p->next;

    }

}

linked\_list& linked\_list::operator=(const linked\_list& right)//赋值运算符重载

{

    if (this == &right) return \*this; // 如果自我赋值，则直接返回

    // 删除原有元素

    node\* p = head;

    while (p != nullptr)

    {

        node\* q = p->next;

        delete p;

        p = q;

    }

    head = tail = nullptr;

    size = 0;

    // 复制头和尾指针

    head = tail = nullptr;

    // 复制元素个数

    size = right.size;

    // 遍历右链表

    for (node\* p = right.head; p != nullptr; p = p->next)

    {

        add\_tail(p->data); // 在尾添加元素

    }

    return \*this; // 返回本对象

}

int linked\_list::list\_size() const

{

    return size;

}

//集合并交叉也可以考虑设计为类外函数

//linked\_list linked\_list::operator+(const linked\_list& right); //链表集合并

//linked\_list linked\_list::operator-(const linked\_list& right);//链表集合差

//linked\_list linked\_list::intersectionSet(const linked\_list& right); //链表集合交

node\* linked\_list::find(dataType value)//查找

{

    node\* p = head;

    while (p != nullptr && p->data != value)

    {

        p = p->next;

    }

    return p;

}

bool linked\_list::find(dataType value, node\*& pre, node\*& p)

{

    p = head;

    while (p != nullptr && p->data != value)

    {

        pre = p;

        p = p->next;

    }

    if (p == nullptr) return false;

    else return true;

}

//考虑修改：bool  find(dataType value,node\*& pre,node\*&p);

//找到返回真：p为目标点，pre为前驱点; 找不到返回假：p和pre均为nullptr

void linked\_list::add\_front(dataType value) //添加到首

{

    node\* newNode = new node(value);

    newNode->next = head;

    if (head == nullptr) tail = newNode;

    head = newNode;

    size++;

}

void linked\_list::add\_tail(dataType value)//添加到尾

{

    node\* newNode = new node(value);

    if (tail == nullptr) head = newNode;

    else tail->next = newNode;

    tail = newNode;

    size++;

}

void linked\_list::add\_pos\_after(dataType value, node\* pos) //添加到指定位置之后

{

    node\* newNode = new node(value);

    newNode->next = pos->next;

    pos->next = newNode;

    if (newNode->next == nullptr) tail = newNode;

    size++;

}

void linked\_list::add\_pos\_before(dataType value, node\* pos)

{//添加到指定位置之前

    node\* newNode = new node(value);

    if (head == pos)

    {

        newNode->next = head;

        head = newNode;

    }

    else

    {

        newNode->next = pos;

        node\* pre = nullptr;

        node\* p = head;

        while (p != pos)

        {

            pre = p;

            p = p->next;

        }

        pre->next = newNode;

    }

    size++;

}

//删除指定值

void linked\_list::Delete(dataType value)

{

    node\* pre = nullptr, \* p = head;

    while (p != nullptr && p->data != value)

    {

        pre = p;

        p = p->next;

    }

    if (p == nullptr) return;

    if (pre == nullptr) head = p->next;

    else pre->next = p->next;

    delete p;

    size--;

}

void linked\_list::delete\_pos\_after(node\* pos)//删除指定位置之后

{

    if (pos == tail) return ;

    else

    {

        node\* pre = pos;

        node\* p = pre->next;

        pre->next = p->next;

        delete p;

        size--;

    }

}

void linked\_list::delete\_pos\_before(node\* pos)//删除指定位置之前

{

    if (pos == head) return ;

    else

    {

        node\* pre = head;

        node\* p = pre->next;

        while (p->next != pos)

        {

            p = p->next;

            pre = pre->next;

        }

        pre->next = p->next;

        delete p;

        size--;

    }

}

void linked\_list::delete\_all(dataType value)//删除所有重复值

{

    node\* p = head;

    while (p != nullptr)

    {

        node\* q = p->next;

        if (p->data == value)

        {

            if (p == head) head = q;

            else p->next = q->next;

            delete p;

            size--;

        }

        p = q;

    }

}

void linked\_list::delete\_repeat()

{

    node\* q = head;

    while (q != nullptr)

    {

        node\* p = q->next;

        node\* pre = q;

        while (p != nullptr)

        {

            if (p->data == q->data)

            {

                pre->next = p->next;

                delete p;

                size--;

                p = pre->next;

            }

            else

            {

                pre = p;

                p = p->next;

            }

        }

        q = q->next;

    }

    tail = q;

}

void linked\_list::reverse()//逆置链表

{

    node\* p = head, \* q = nullptr, \* r = nullptr;

    while (p != nullptr)

    {

        r = q;

        q = p;

        p = p->next;

        q->next = r;

    }

    tail = head;

    head = q;

}

void linked\_list::sort()//升序排列当前链表

{

    node\* p = head;

    dataType temp;

    while (p != nullptr)

    {

        node\* q = p->next;

        while (q != nullptr)

        {

            if (p->data > q->data)

            {

                temp = p->data;

                p->data = q->data;

                q->data = temp;

            }

            q = q->next;

        }

        p = p->next;

    }

}

void linked\_list::display()//遍历链表

{

    node\* p = head;

    while (p != nullptr)

    {

        cout << p->data << " ";

        p = p->next;

    }

    cout << endl;

}

linked\_list::~linked\_list()

{

    node\* p = head;

    while (p != nullptr)

    {

        node\* q = p->next;

        delete p;

        p = q;

    }

    head = tail = nullptr;

    size = 0;

}

linked\_list linked\_list::operator+(const linked\_list& right)//链表集合并

{

    linked\_list result;

    node\* p = head;

    while (p != nullptr)

    {

        result.add\_tail(p->data);

        p = p->next;

    }

    p = right.head;

    while (p != nullptr)

    {

        result.add\_tail(p->data);

        p = p->next;

    }

    return result;

}

linked\_list linked\_list::operator-(const linked\_list& right)//链表集合差

{

    linked\_list result;

    node\* p = head;

    while (p != nullptr)

    {

        result.add\_tail(p->data);

        p = p->next;

    }

    p = right.head;

    while (p != nullptr)

    {

        result.Delete(p->data);

        p = p->next;

    }

    return result;

}

linked\_list linked\_list::intersectionSet(const linked\_list& right) //链表集合交

{

    linked\_list result;

    node\* p = head;

    while (p != nullptr)

    {

        result.add\_tail(p->data);

        p = p->next;

    }

    p = right.head;

    while (p != nullptr)

    {

        result.find(p->data, result.head, result.tail);

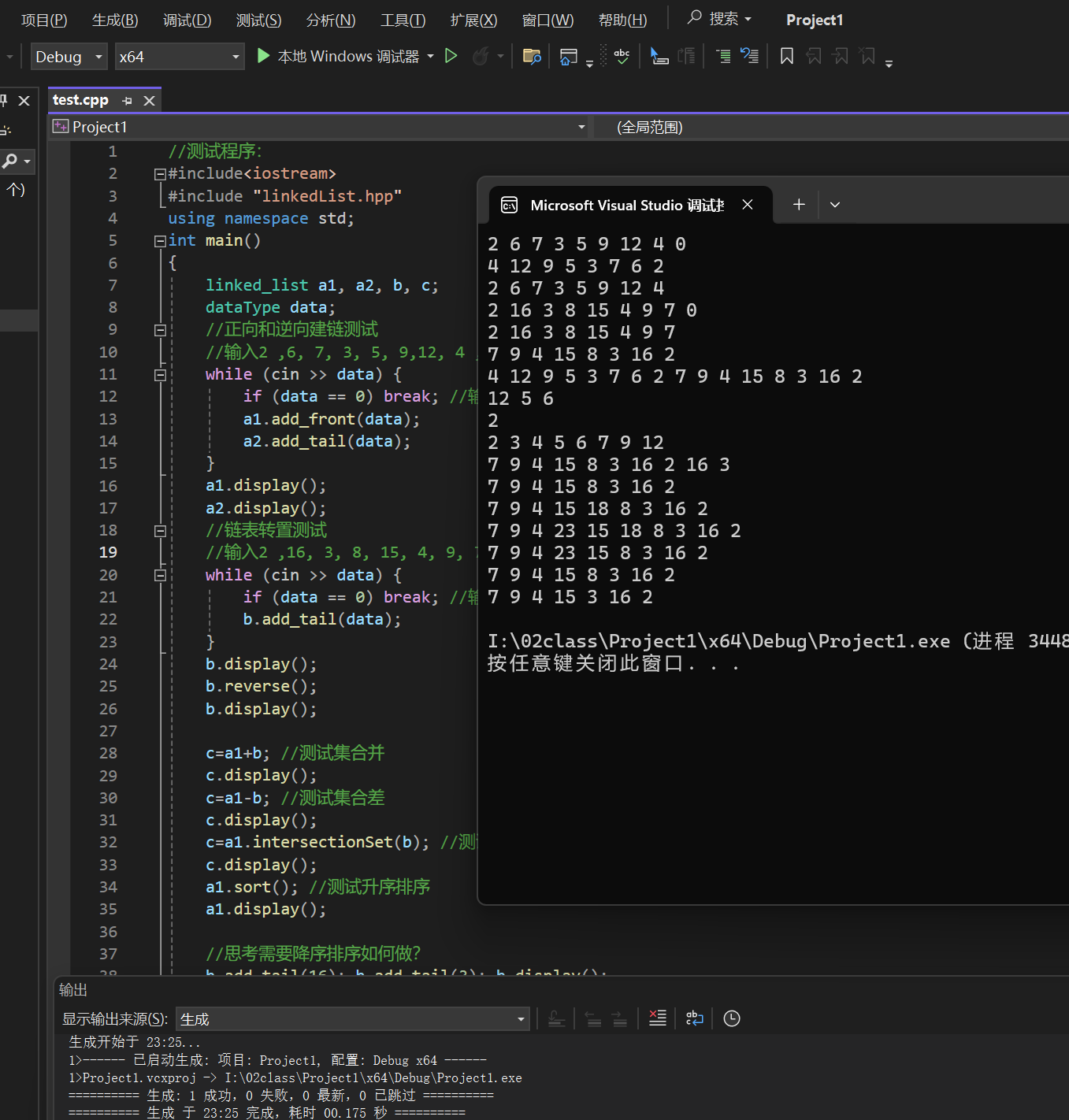
        p = p->next;

    }

    return result;

}

**❷黏贴程序测试运行结果窗口（运行结果截屏）：**

****

**2、文件部分实验：学会ASCII文件的基本输入输出流读写（50分）**

**（1）写一个程序实现：随机生成两批数据，每批10个整数，范围自定。将第一组数据随机数写入 data1.txt；第二组随机书写入data2.txt （25分）**

**（2）写一个程序实现：读取data1.txt和data2.txt的共20个整数有序插入到数组a，再将数组a的结果输出到屏幕同时写入文件data3.txt存档。 （25分）**

**实验解答：**

**❶黏贴（1）的实现代码：**

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <ctime>

using namespace std;

int main() {

    // 初始化随机数种子

    srand(static\_cast<unsigned int>(time(nullptr)));

    // 第一组数据

    int data1[10];

    for (int i = 0; i < 10; ++i)

    {

        data1[i] = rand() % 100; // 假设范围是0-99

    }

    // 写入data1.txt

    ofstream file1("data1.txt");

    for (int i = 0; i < 10; ++i)

    {

        file1 << data1[i] << endl;

    }

    file1.close();

    // 第二组数据

    int data2[10];

    for (int i = 0; i < 10; ++i)

    {

        data2[i] = rand() % 100;

    }

    // 写入data2.txt

    ofstream file2("data2.txt");

    for (int i = 0; i < 10; ++i)

    {

        file2 << data2[i] << endl;

    }

    file2.close();

    return 0;

}

**❷黏贴（2）的实现代码：**

#include <iostream>

#include <fstream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

    vector<int> a;

    int number;

    // 读取data1.txt

    ifstream file1("data1.txt");

    while (file1 >> number) {

        a.push\_back(number);

    }

    file1.close();

    // 读取data2.txt

    ifstream file2("data2.txt");

    while (file2 >> number) {

        a.push\_back(number);

    }

    file2.close();

    // 排序数组

    sort(a.begin(), a.end());

    // 输出到屏幕

    cout << "Array: ";

    for (int i = 0; i < a.size(); ++i) {

        cout << a[i] << " ";

    }

    cout << endl;

    // 写入data3.txt

    ofstream file3("data3.txt");

    for (int i = 0; i < a.size(); ++i) {

        file3 << a[i] << endl;

    }

    file3.close();

    return 0;

}