上 海 大 学

2021-2022 冬季学期

《数据结构（1）》实验报告

实 验 组 号：08

上 课 老 师：沈　俊

小 组 成 绩：

小组成员成绩表

序号	学号	姓名	贡献因子	成绩
1	20120500	王静颐	20	
2	20120796	康高熙	20	
3	20121034	胡才郁	20	
4	20121076	刘元	20	
5	20124633	金之谦	20	

注：小组所有成员的贡献因子之和为 100.

计算机工程与科学学院

2021 年 12 月 20 日

实验二 线性表

1 验证性实验

1.1 不带头结点的单循环链表类模板

1.1.1 实验内容

本项目为不带头结点的单循环链表类模板。本模板严格遵守程序设计的三层结构、模块化编程与项目管理基本原则。

1.1.2 主要算法设计

构造函数：

由于该链表不带头结点，所以需要对插入第一个结点进行特殊处理，并且使用了冒号语法对成员属性进行赋值处理。

```
template<typename T>
CircularLinkList<T>::CircularLinkList(T *v, int size): head(NULL),length(size)
{
    Node<T> *p = head = new Node<T>(v[0], head);
    head->next = head;
    for (int i = 1; i < size; i++) {
        p->next = new Node<T>(v[i], head);
        p = p->next; // 循环结束时 p 指向 head
    }
}
```

图 1. 构造函数

基本显示功能：

进入函数体后，先判断链表是否为空，这个操作避免了在使用清空链表后出现异常的情况。在函数体中，通过定义一个辅助指针遍历链表的每一个节点，输出每一个节点的数据域（data）信息。

```
template<typename T>
void CircularLinkList<T>::Show() const {
    if (head == NULL)
        return;
    Node<T> *p = head;
    while (p->next != head) {
        cout << p->data << " ";
        p = p->next;
    }
    cout << p->data << endl;
}
```

图 2. Show 功能函数

基本添加功能：

本项目可以在不带头节点的单向循环链表的任意位置插入元素。在函数体内，首先判断插入的请求

插入的位置 n 是否合法。在本函数中需要注意的一点是要记得 length 属性的更新。由于此链表不带头节点，需要对两种情况进行处理，链表为空和链表不为空时在第一个位置插入元素，通过判断 i 是否等于 1 和 head 是否为 NULL，进行了特殊处理。由于在函数体内进行了遍历操作，此算法时间复杂度为$\boldsymbol{O(n)}$。

```
template<typename T>
Status CircularLinkList<T>::InsertElem(int i, const T &e) {
    if (i < 1 || i > length + 1)
        return RANGE_ERROR;
    Node<T> *p, *q;
    if (i == 1) {
        if (head == NULL) {
            head = new Node<T>(e, head);
            head->next = head;
        } else {
            p = head;
            int count = 1;
            for (; count < length; count++) {p = p->next;} // 寻找末尾节点的位置
            q = new Node<T>(e, head);
            head = q;
            p->next = head;
        }
    } else {
        p = head;
        for (int count = 2; count < i; count++) {p = p->next;}
        q = new Node<T>(e, p->next);
        p->next = q;
    }
    length++;
    return SUCCESS;
}
```

图 3. InsertElem 功能函数

1.1.3 主要数据组织

在原有模板的基础上，添加了拷贝构造函数与赋值运算符函数的设计，并且实现了基本的增删改查功能。这里主要对其整体结构进行一个梳理：

表 1. CircularLinkList 类中各函数的声明

函数原型	返回值类型	功能
CircularLinkList();		空参构造函数，初始化链表（空链表）
CircularLinkList(T v[], int size);		初始化链表
virtual ~CircularLinkList();		析构函数，清空链表
CircularLinkList(const CircularLinkList<T> &l);		拷贝构造函数
CircularLinkList<T> &operator= (const CircularLinkList<T> &l);	CircularLinkList<T>&	赋值运算符函数
void Clear(); // 清空循环单链表	void	清空链表

void Show() const;	void	输出链表信息
int LocateElem(const T &e) const;	int	按值查找
Status GetElem(int i, T &e) const;	Status	按位查找
Status SetElem(int i, const T &e);	Status	修改指定位置元素值
Status DeleteElem(int i, T &e);	Status	删除指定元素
Status InsertElem(int i, const T &e);	Status	插入指定元素

以下介绍部分函数以及设计思路：

1.1.3 测试分析

功能菜单如下，以下使用了一种边界情况进行测试，即生成单链表后，依次删除元素至链表为空，当链表为空时进行显示操作，并且在此时进行插入操作。对于此测试样例，本程序运行成功。

```
1. 生成单链表.
2. 显示单链表.
3. 取指定位置的元素.
4. 设置元素值.
5. 删除元素.
6. 插入元素.
7. 元素定位
8. 取单链表长度
0. 退出
```

以下为链表的生成与删除表头位置的元素。

```
选择功能(0~8):1
输入e(e = 0时退出):1 5 2 0
输入位置:1
被删除元素值:1
输入位置:1
被删除元素值:5
输入位置:1
被删除元素值:2
```

此时链表为空，以下测试在链表为空时的显示与插入操作：

```
选择功能(0~8):6
输入位置:1
输入元素值:999
成功:999
选择功能(0~8):2
-----head为NULL-----
_____Funtion Show()_____
999
```

程序运行成功！

1.2 不带头结点的双向非循环链表类模板

1.2.1 实验内容

不带头节点的双向非循环链表类模板。在实现了不循环的双向链表之后，在此基础上实现了循环双向链表。本模板严格遵守程序设计的三层结构、模块化编程与项目管理基本原则

1.2.1 主要算法设计

插入功能：

由于该链表不带头结点，所以需要对插入第一个结点进行特殊处理。本项目可以在不带头节点的双向非循环链表的任意位置插入元素。在函数体内，首先判断插入的请求插入的位置 n 是否合法，如果不合法，通过自定义的异常类 illegalIndex 进行抛出并报错。如果要插入的位置是尾部的话，单独调用尾插

入函数进行处理。在插入后进行进行数据成员的更新。

```
template<class T>
void DoubleLinkedList<T>::insert_index(T data, int index){
    if (index < 0 || index > this->_size){
        throw(illegalIndex(index));
    }
    if (index == this->_size){
        insert_tail(data);
        return;
    }
    Node<T>* newNode = new Node<T>;
    newNode->set_data(data);
    Node<T>* traversal = HEAD;
    int count = 0;
    for (; traversal != nullptr; traversal = traversal->get_next(), count++){
        if (count == index){
            newNode->set_pre(traversal->get_pre());
            if (traversal->get_pre() != nullptr){
                traversal->get_pre()->set_next(newNode);
            }
            else{
                HEAD = newNode;
            }
            newNode->set_next(traversal);
            traversal->set_pre(newNode);
            this->_size++;
            return;
        }
    }
}
```

图 4. 插入函数

删除功能：

与插入函数相同，首先需要判断申请删除的位置是否合法，如果不合法，同样通过 illegalIndex 抛出。并且需要注意一些特殊情况的处理：

1. 如果删除的是头部，需要更新成员数据
2. 如果删除的节点之后还有节点，需要更新下一个节点的前键

```
template<class T>
void DoubleLinkedList<T>::delete_index(int index){
    if (index < 0 || index >= this->_size){
        throw(illegalIndex(index));
    }
    Node<T>* traversal = HEAD;
    int count = 0;
```

```
    for (; traversal != nullptr; traversal = traversal->get_next(), count++){
        if (count == index){
            if (traversal->get_pre() != nullptr){
                traversal->get_pre()->set_next(traversal->get_next());
            }
            else{
                HEAD = traversal->get_next();
            }
            if (traversal->get_next() != nullptr){
                traversal->get_next()->set_pre(traversal->get_pre());
            }
            this->_size--;
            if (traversal)
                delete traversal;
            return;
        }
    }
}
```

图 5.delete_index 函数

显示功能：

在此部分通过展示前后指针所指向的地址，展现了此链表的逻辑结构，保证了算法的正确性

```
template<class T>
void DoubleLinkedList<T>::show_list()
{
    std::cout << "+--------------DOUBLE LINKED LIST---------------+" << std::endl;
    std::cout << "| List Head: " << this->_head << "\t\t\t\t|" << std::endl;
    std::cout << "| List Size: " << this->_size << "\t\t\t\t\t|" << std::endl;
    std::cout << "|-----------------------------------------------|" << std::endl;
    int i = 0;
    for (Node<T>* traversal = HEAD; i < this->_size && traversal != nullptr; i++,
traversal = traversal->get_next())
    {
        std::cout << "| \t |" << " \tDATA: " << traversal->get_data() << "(" <<
traversal->get_dataP() << ")\t\t|" << std::endl;
        std::cout << "| Node " << (i + 1) << " |" << "\tPre: " << traversal->get_pre()
<< "\t\t\t| " << std::endl;
        std::cout << "| \t |\tNext: " << traversal->get_next() << "\t\t\t|" <<
std::endl;
        std::cout << "| \t |\tADDRESS: " << traversal << "\t\t|" << std::endl;
        if (i != this->_size - 1)
        {
            std::cout << "|-----------------------------------------------|" <<
std::endl;
```

```
        }
    }
    std::cout << "+-----------------------------------------------+" << std::endl
<< std::endl << std::endl;
}
```

图 6. delete_index 函数

1.2.3 主要数据组织

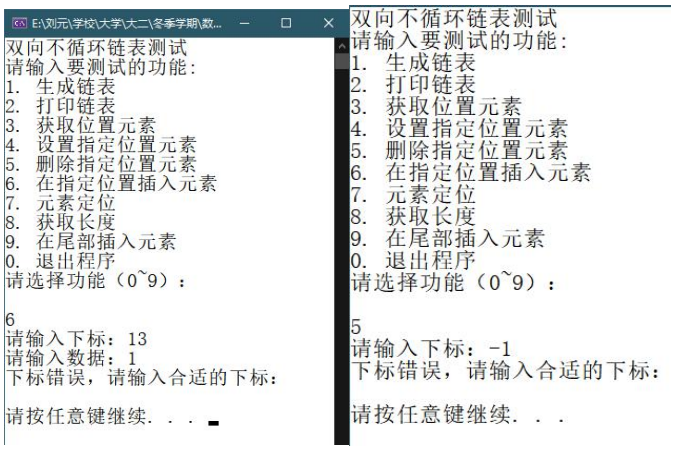
完成了四大函数的设计，并且实现了基本的增删查改功能，还自行设计了 illegalIndex 异常类，在输入不合法下标的时候抛出、抓取异常。

表 2. DoubleLinkedList 类中各函数的声明

函数原型	返回值类型	功能
DoubleLinkedList();		空参构造函数，初始化链表（空链表）
DoubleLinkedList(int size);		析构函数，清空链表
DoubleLinkedList(const DoubleLinkedList<T>& list);	DoubleLinkedList<T>&	赋值运算符函数
~DoubleLinkedList();	无	释放堆空间，析构对象
insert_head(T data);	void	头插入，插入数据类型
insert_tail(T data);	void	尾插入，插入数据类型
insert_index(T data,int index);	void	按下标插入，并检查下标是否合法
delete_head();	void	删除头节点
delete_index(int index);	void	删除指定下标元素
delete_tail();	void	删除尾结点
set_ele(T data,int index);	void	设定指定下标的数据
get_index(T data);	int	获得第一个符合条件的元素的下标
get_elem(int index);	T	获取对应下标元素
getSize() { return this->_size; }	int	获取链表长度
show_list();	void	打印链表

1.2.3 测试分析

以下测试了该链表的插入与删除操作，并展示出链表前后指针所指向的内存地址。



```
+--------------DOUBLE LINKED LIST---------------+
| List Head: 00755A90                           |
| List Size: 2                                  |
|-----------------------------------------------|
|          |      DATA: 1(0075D088)             |
| Node 1   |      Pre: 00000000                 |
|          |      Next: 00755A60                |
|          |      ADDRESS: 00755A90             |
|-----------------------------------------------|
|          |      DATA: 2(0075D098)             |
| Node 2   |      Pre: 00755A90                 |
|          |      Next: 00000000                |
|          |      ADDRESS: 00755A60             |
+-----------------------------------------------+
```

2 设计性实验

2.1面试安排

2.1.1实验内容

题意是说，X 和 Y 两个人，在一个圆环上不断找点，然后删除节点。圆环结构，很自然想到我们要写一个循环链表。而每次 X 是逆时针找点，Y 是顺时针找点，因此我们还需要写双向链表，最终我们需要实现的是双向循环链表。

2.1.2 算法分析

接下来考虑如何把题目中的过程用链表模拟出来。注意点题目中要求的是按照逆时针的顺序节点从 1 到 N。但其实我们一般喜欢的是从 1 到 N 为顺时针方向。在写链表时，可以把从 1 到 N 变为顺时针的方向，然后 X 变为顺时针找点，Y 变为逆时针找点，最终结果殊途同归。用链表来模拟过程的话，我们需要一个静态指针成员 X 和 Y 来记录小 X 和小 Y 两个人所在的位置。因为两个人同时选中简历后拿走，再移动一个位置。因此我们需要先找到 X 要删除的点，Y 要删除的点，在找点过程中就把静态指针的值更新好，然后两个点同时删掉。如果恰好要删的是同一个点，那么只需要删一次。删除时，可能会遇到，小 X 或小 Y 再找到简历后，移动一个位置恰好到了这里。那么需要把小 X 或小 Y 再移动一个位置。举例子，N 是 5，第一轮的时候，小 X 要删掉 3 号点，小 Y 要删掉 4 号点。如果小 Y 直接删掉 3 号点后移动一个位置应该到达 4 号点，小 Y 直接删掉 4 号点后移动一个位置应该到达 3 号点。而 3 号点和 4 号点恰恰被删除了。所以小 X 最终到了 5 号点，小 Y 到了 2 号点。

2.1.3 主要数据组织

Node 类有数据成员：

int data;Node *pre,*nxt;分别是节点中存的节点编号，节点的前驱和后继。

有成员函数：

基本构造函数；

getdata 函数；//用来返回节点中存的节点编号

令 Circle 类为 Node 的友元类。

Circle 类有数据成员：

```
static Node *x,*y;//记录小 X 和小 Y 所在位置
 int len;//当前链表剩余长度
 Node *head;//链表头结点
```

有成员函数：

基本构造函数即析构函数；//该函数通过调用 Append 函数，生成一个顺时针从 1 到 N 的双向循环链表。

Apppend(int a)； //将 a 插入当前链表的末尾

Node* Findx(int cnt); //返回从当前小 x 所在位置顺时针走 cnt 步所到达的位置，同时更新 Node *x；

Node* Findy(int cnt); //返回从当前小 y 所在位置逆时针走 cnt 步所到达的位置，同时更新 Node *y；

void Delete(Node *p);//删除 p 节点；如果 p 是 X 或者 Y，先把 X 或者 Y 移动到下一个位置，再删除 p；

bool Isempty(); //判断链表是否为空

static void setpos(); //初始时将 X 和 Y 的位置设置为 1 和 N，即 head 和 head->pre；

2.1.4 数据测试

首先测试题目样例，N 为 10，k 为 4，m 为 3。

```
10 4 3
5,7;1,2;9,6;10,8;3;4。
--------------------------------
Process exited after 5.154 seconds with return value 0
请按任意键继续. . .
```

输出结果与题目所给答案并不吻合。手推发现，题中所给样例其实是有问题的，他每次只走了 k-1 步和 m-1 步就把到达节点删除了。而根据题意，应该是走 k 步和 m 步。

倘若 N 为 0 呢？

```
0 1 2
链表为空!

--------------------------------
Process exited after 3.419 seconds with return value 0
请按任意键继续. . .
```

Main 函数中加入了对此情况的特判，每次对类进行操作时，为了防止出现不合法情况，都要先判断链表是否为空。

再测一组上面提到的例子。N=5，k=2，m=1；

```
5 2 1
3,4;2,1;5。
--------------------------------
Process exited after 2.717 seconds with return value 0
请按任意键继续. . .
```

手推一下，很明显，并无问题。

再来一组大一点的。N=20，k=2，m=5；

```
20 2 5
3,15;6,9;10,1;13,14;18,4;2,12;8,19;17,20;11;7,16;5。
--------------------------------
Process exited after 2.721 seconds with return value 0
请按任意键继续. . .
```

2.2物流管理

2.2.1 实验内容

本实验实现了模拟入库、出库和仓库盘点操作。通过提供相应的函数接口，利用不带头结点的单循环链表的数据结构实现了功能。

2.2.1 主要算法设计

根据题意，我们需要一个链表来维护信息。链表中的每个节点就对应于每一种货物存放的空间，他有自己的位置，有货物数量，有后继。添加货物时，若链表中不存在该种货物，就在链表中添加一个节点。如果存在，就更新节点信息。出货类似，若不存在该类货物，则不合法。如果出货量大于等于存货量，则在链表中删除该类货物的节点。

2.2.3 主要数据组织

分析结束以后发现，题目所需的所有功能，验证性实验 1 的单向链表类模板都能实现，只需要将其中的模板 T 改为 Gift 即可。Gift 这个类需要单独设计一下。

Gift 类有数据成员：

Name：记录该节点的货物名称；
Code：记录该节点的货物编号；
Num：记录该节点的货物数量；

实现成员函数如下：

基本构造与析构函数；
Askname：查询该节点货物名称；
Askcode：查询该节点货物编号；
Asknum：查询该节点货物数量；
Setdata：设置货物的属性
Addnum：添加该节点货物数量，且实现超出限制自动报警以及处理。
Delnum：减少该节点货物数量，且实现减少货物数量超过该节点货物数量的情况下清空库存，并在测试函数中实现了对该节点货物的删除操作；
ShowData：输出货物信息；
重载 = 实现类的赋值；
重载 == 和 != 实现货物的比较，使其能够通过两个运算符来查找货物位置；
重载 << 实现信息的输出；

Gift 类的设计如下：

```
class Gift{
public:
    Gift(string name="noname",string code="0000",int num=0) :
Name(name),Code(code),Num(num){
         if(Num>1000){
            Num=1000;
            WarnFilled(cout);
         }
    }//构造函数
    friend class CircularLinkList<Gift>; //将链表类设为它的友元类
    string Askname() const; //返回货物的名字
    string Askcode() const;//返回货物的编号
    int Asknum() const;//返回货物的数量
    void Setdata(string name,string code,int num);//设置货物的属性
    void Addnum(int x);//添加货物数量
    bool Delnum(int x);//减去货物
    void ShowData(ostream &out) const;//输出货物信息
    operator =(const Gift & b){ Name=b.Name;Code=b.Code;Num=b.Num; }
    friend ostream & operator <<(ostream &out,const Gift &a);
    bool operator ==(const Gift & b) const{
        return Name==b.Name && Code==b.Code;
    }
    bool operator !=(const Gift & b) const{
        return !(*this==b);
    }
private:
    string Name, Code;
    int Num;
```

```
};
```

2.2.4 测试分析

基本菜单如下：

```
欢迎使用物流库存管理系统!
请输入数字以选择服务:
1 -- 入库/增加货物库存
2 -- 出库/减少货物库存
3 -- 查询货物信息
4 -- 盘点输出所有货物信息
0 -- 退出
请选择：1

请输入货物编号:
20124663
请输入货物名称:
处理器
当前列表中未找到该货物，已自动添加该货物。

请输入需要增加的货物数量：10

请按任意键继续
```

添加货物数量测试，此处测试了超过 1000 库存上限的情况：

```
欢迎使用物流库存管理系统!
请输入数字以选择服务:
1 -- 入库/增加货物库存
2 -- 出库/减少货物库存
3 -- 查询货物信息
4 -- 盘点输出所有货物信息
0 -- 退出
请选择：1

请输入货物编号:
20124663
请输入货物名称:
处理器
找到该货物。

请输入需要增加的货物数量：1000
报警Warning：当前操作产品库存已超过1000，系统自动设置上限为1000，多余部分将不会被计入。
请按任意键继续
```

减少货物数量测试，此处同时测试输出全部货物信息的操作，由于上一次增加操作超出库存，因此系统自动设定货物数量为 1000，减少 100 之后变为 900：

```
欢迎使用物流库存管理系统!
请输入数字以选择服务:
1 -- 入库/增加货物库存
2 -- 出库/减少货物库存
3 -- 查询货物信息
4 -- 盘点输出所有货物信息
0 -- 退出
请选择：2

请输入货物编号:
20124663
请输入货物名称:
处理器
找到该货物。

请输入需要减少的货物数量：100

请按任意键继续
```

```
欢迎使用物流库存管理系统!
请输入数字以选择服务:
1 -- 入库/增加货物库存
2 -- 出库/减少货物库存
3 -- 查询货物信息
4 -- 盘点输出所有货物信息
0 -- 退出
请选择：4

接下来将输出列表中所有货物的信息：

编号：20124663   名称：处理器      数量：900


请按任意键继续
```

减少货物数量测试 2，此处测试没有该种货物时的输出情况：

```
欢迎使用物流库存管理系统!
请输入数字以选择服务:
1 -- 入库/增加货物库存
2 -- 出库/减少货物库存
3 -- 查询货物信息
4 -- 盘点输出所有货物信息
0 -- 退出
请选择：2

请输入货物编号:
12345
请输入货物名称:
1
当前列表中未找到该货物。

请按任意键继续
```

入库第二种货物：

```
欢迎使用物流库存管理系统!
请输入数字以选择服务:
1 -- 入库/增加货物库存
2 -- 出库/减少货物库存
3 -- 查询货物信息
4 -- 盘点输出所有货物信息
0 -- 退出
请选择：1

请输入货物编号:
12345
请输入货物名称:
电脑
当前列表中未找到该货物，已自动添加该货物。

请输入需要增加的货物数量：100

请按任意键继续
```

```
欢迎使用物流库存管理系统!
请输入数字以选择服务:
1 -- 入库/增加货物库存
2 -- 出库/减少货物库存
3 -- 查询货物信息
4 -- 盘点输出所有货物信息
0 -- 退出
请选择：4

接下来将输出列表中所有货物的信息：

编号：12345      名称：电脑        数量：100
 编号：20124663  名称：处理器      数量：900


请按任意键继续
```

货物出库测试（即减少该货物的全部库存数量）：

```
欢迎使用物流库存管理系统!
请输入数字以选择服务:
1 -- 入库/增加货物库存
2 -- 出库/减少货物库存
3 -- 查询货物信息
4 -- 盘点输出所有货物信息
0 -- 退出
请选择：2

请输入货物编号:
12345
请输入货物名称:
电脑
找到该货物。

请输入需要减少的货物数量：100
当前货物库存已空，将删除该货物。

请按任意键继续
```

```
请输入需要减少的货物数量：100
当前货物库存已空，将删除该货物。

请按任意键继续


欢迎使用物流库存管理系统!
请输入数字以选择服务:
1 -- 入库/增加货物库存
2 -- 出库/减少货物库存
3 -- 查询货物信息
4 -- 盘点输出所有货物信息
0 -- 退出
请选择：4

接下来将输出列表中所有货物的信息：

编号：20124663   名称：处理器      数量：900


请按任意键继续
```

查询货物信息测试：

```
欢迎使用物流库存管理系统!
请输入数字以选择服务:
1 -- 入库/增加货物库存
2 -- 出库/减少货物库存
3 -- 查询货物信息
4 -- 盘点输出所有货物信息
0 -- 退出
请选择：3

请输入货物编号:
20124663
请输入货物名称:
处理器
找到该货物，下为该货物具体信息。
编号：20124663   名称：处理器      数量：900


请按任意键继续
```

```
欢迎使用物流库存管理系统!
请输入数字以选择服务:
1 -- 入库/增加货物库存
2 -- 出库/减少货物库存
3 -- 查询货物信息
4 -- 盘点输出所有货物信息
0 -- 退出
请选择：3

请输入货物编号:
1
请输入货物名称:
12345
当前列表中未找到该货物。

请按任意键继续
```

3.1 课程设计中遇到的问题和解决方法

在设计不带头结点的链表时，我们遇到插入了删除头结点位置的困难，经过了小组讨论与搜集资料，我们对特殊位置、临界位置进行了特殊处理，保证了程序的健壮性。

3.2 实验总结

在完成老师要求的三个项目的基础上，我们多完成了一个项目。验证性实验分为单循环链表类模板和双向非循环链表类模板，要求是不带头结点。验证性实验的带头结点版本，课上老师已经详细为我们介绍过，这个实验让我们改为不带头结点，只需要轻微修改就可以。目的在于熟练掌握这两种类模板，并可以拿他们解决问题。设计性实验，第一个可以用双向循环链表解决，第二个用单循环链表解决。如果说前两个验证性实验是夯实基础的话，那么后两个设计性实验就是锻炼综合运用能力，在理解题意后转化问题，套用模板，解决问题。严谨性也同样重要，代码实现以后，一定要通过多组数据检验，确保其正确性。