#include<iostream>

using namespace std;

template<class ElemType>

struct ListQueueNode

{

//成员变量:

ElemType data;

ListQueueNode<ElemType>\* next;

//相关的成员函数:

ListQueueNode<ElemType>(ListQueueNode<ElemType>\* ptr = NULL) { next = ptr; } //构造函数1，用于构造头结点

ListQueueNode<ElemType>(const ElemType& item, ListQueueNode<ElemType>\* ptr = NULL) { //构造函数2，用于构造其他结点

//函数参数表中的形参允许有默认值，但是带默认值的参数需要放后面

next = ptr;

data = item;

}

ElemType getData() { return data; } //取得结点中的数据

void setPointer(ListQueueNode<ElemType>\* link) { next = link; } //修改结点的next域

void setData(ElemType value) { data = value; } //修改结点的data域

};

//不带头结点的链队列

template<class ElemType>

class ListQueue {

private:

ListQueueNode<ElemType>\* front; // 队头指针

ListQueueNode<ElemType>\* rear; // 队尾指针，指向队列中最后一个元素

int length; //队列当前元素个数

public:

//无参数的构造函数

ListQueue() {

front = new ListQueueNode<ElemType>;

rear = front;//当队列为空时，front和rear均指向头节点，但是因为这个是不带头结点的链表，

//因此这个front->data可以存储数据，但目前不存储数据

front->next = NULL;

rear->next = NULL;

length = 0;

}

//队列的拷贝构造函数，未改

ListQueue(const ListQueue<ElemType>& Q);//以下两个函数除名字不一样外，其他都基本完全一样

//队列的拷贝赋值运算符，未改

ListQueue<ElemType>& operator=(const ListQueue<ElemType>& Q);

//析构函数

~ListQueue() { ListQueueDestroy(); cout << "调用了析构函数" << endl; }

//销毁链队列

bool ListQueueDestroy();

//清空链表

bool ListQueueClear();

//返回链队列的长度

int getLength() const { return length; }

//判断链队列是否为空队列

bool isEmpty() const;

//出队

bool deQueue(ElemType& e);

//入队

bool enQueue(ElemType e);

//获取链队列头结点指针

ListQueueNode<ElemType>\* getFrontPointer() const { return front; }

//获取队头元素

ElemType getFrontElem() const { return front->data; }

//获取链队列队尾指针

ListQueueNode<ElemType>\* getRearPointer() const { return rear; }

//遍历链队列

bool traverse() const;

};

//队列的拷贝构造函数

template<class ElemType>

ListQueue<ElemType>::ListQueue(const ListQueue<ElemType>& Q) {

front = new ListQueueNode<ElemType>;

rear = front;

front->next = NULL;

length = 0;

if (!Q.isEmpty()) {

ListQueueNode<ElemType>\* p = Q.getFrontPointer();

while (p) {

if(p == Q.getFrontPointer()) {

front->data = p->data;

++length;

p = p->next;

}

else {

ListQueueNode<ElemType>\* tmp = new ListQueueNode<ElemType>;

tmp->data = p->data;

rear->next = tmp;

tmp->next = NULL;

rear = tmp;

++length;

p = p->next;

}

}

}

cout << "调用了拷贝构造函数" << endl;

}

//队列的拷贝赋值运算符

template<class ElemType>

ListQueue<ElemType>& ListQueue<ElemType>::operator=(const ListQueue<ElemType>& Q) {

front = new ListQueueNode<ElemType>;

rear = front;

front->next = NULL;

length = 0;

if (!Q.isEmpty()) {

ListQueueNode<ElemType>\* p = Q.getFrontPointer();

while (p) {

if (p == Q.getFrontPointer()) {

front->data = p->data;

++length;

p = p->next;

}

else {

ListQueueNode<ElemType>\* tmp = new ListQueueNode<ElemType>;

tmp->data = p->data;

rear->next = tmp;

tmp->next = NULL;

rear = tmp;

++length;

p = p->next;

}

}

}

cout << "调用了拷贝赋值运算符函数" << endl;

return \*this;

}

//销毁链队列

template<class ElemType>

bool ListQueue<ElemType>::ListQueueDestroy() {

if (front==NULL) {//因为没有了头节点，因此front指向的那个结点的data域存储数据，所以原来有头节点的队列

//判断空的时候的条件就是看看是否front==rear，而如果用这个条件判断没有头节点的链队列，就有可能使得

//这个链队列到时候删不干净

return false;//表明要删除的链表已经删完，没有要删的结点了

}

else {

ListQueueNode<ElemType>\* p = front->next;

while (front->next) {

front->next = p->next;

if (p == rear) {

rear = front;

}

delete p;

p = front->next;

}

delete front;

return true;

}

}

//清空链表

template<class ElemType>

bool ListQueue<ElemType>::ListQueueClear() {

if (this->isEmpty()) {

return false;

}

else {

ListQueueNode<ElemType>\* p = front->next;

while (front->next) {

front->next = p->next;

if (p == rear) {

rear = front;

}

delete p;

p = front->next;

--length;

}

front->data = (ElemType)0;//不知道这句能不能行，感觉够呛，待测试

--length;

//length = 0;

return true;

}

}

//判断链队列是否为空队列

template<class ElemType>

bool ListQueue<ElemType>::isEmpty() const {//队列为空的意义就是队列中没有元素，不是队列不存在了，而是队列中

//不存在有效的元素了，此时只有一个结点，而且这个结点即使存着元素，我们也认为它空着，或者这唯一的一个结点

//也不存在有效的值

if (length==0) { return true; }

else return false;

}

//出队

template<class ElemType>

bool ListQueue<ElemType>::deQueue(ElemType& e) {

if (this->isEmpty()) {//队列中没有有效元素

return false;

}

else {//队列中有有效元素，但要考虑又可能只有一个元素的结点

ListQueueNode<ElemType>\* p = front;//这是存储有效元素的首结点

e = p->data;

if (front->next != NULL) {//这里要考虑NULL，即如果只有一个结点的有效元素，那么此时我们不把这个front指针往后移

front = front->next;//也就是说pop操作后

if (p == rear) {

rear = front;

}

delete p;

--length;

}

else {

--length;

front->data = (ElemType)0;//不知道这么写行不？待检验

}

return true;

}

}

//入队

template<class ElemType>

bool ListQueue<ElemType>::enQueue(ElemType e) {

if (length == 0) {

front->data = e;

++length;

return true;

}

else {

ListQueueNode<ElemType>\* p = new ListQueueNode<ElemType>;

p->data = e;

rear->next = p;

p->next = NULL;

++length;

rear = p;

return true;

}

}

//遍历链队列

template<class ElemType>

bool ListQueue<ElemType>::traverse() const {

if (this->isEmpty()) {

return false;

}

else {

ListQueueNode<ElemType>\* p = front;

while (p) {

cout << p->data << ' ';

p = p->next;

}

return true;

}

}//经测试暂时不认为有bug