

*/\** *circq.h*

*\**

*\** *Declares* *CircularQ* *data* *type* *and* *associated* *operators*

*\**

*\** *Johnathan* *Lee* *CSCI* *1107*

*\** *Lab* *10* *Due* *04/03/18*

*\*/*

#ifndef CIRCQ\_H

#define CIRCQ\_H

#include <iostream>

*using* *namespace* std;

*typedef* int QueueElement; ***///>!*** ***The*** ***type*** ***of*** ***element*** ***stored*** ***in*** ***the*** ***queue.***

***///*** \class ***CircularQ***

***///*** \brief ***A*** ***Queue*** ***implemented*** ***with*** ***a*** ***circular*** ***linked*** ***list.***

***///*** \note ***Uses*** ***sizeof(QueueElement)*** ***+*** ***sizeof(POINTER)*** ***per*** ***element.***

*class* CircularQ {

*public*:

***///*** \brief ***Default*** ***(Empty)*** ***constructor***

***///*** \param ***None***

***///*** \post ***\*this*** ***is*** ***initialized*** ***with*** ***absolutely*** ***no*** ***elements.***

**CircularQ**();

***///*** \brief ***Copy*** ***constructor:***

***///*** \param ***orig*** ***The*** ***CircularQ*** ***to*** ***copy*** ***elements*** ***from***

***///*** \post ***All*** ***elements*** ***from*** ***orig*** ***are*** ***now*** ***ALSO*** ***contained*** ***in*** ***\*this.***

***///*** \note ***This*** ***does*** ***NOT*** ***transfer*** ***ownership,*** ***it*** ***is*** ***a*** ***deep*** ***copy,*** ***orig*** ***is***

***///*** ***unchanged.***

**CircularQ**(*const* CircularQ& orig);

***///*** \brief ***Destructor***

***///*** \post ***All*** ***elements*** ***in*** ***this*** ***queue*** ***are*** ***deleted***

~**CircularQ**();

***///*** \brief ***Assignment***

***///*** \param ***rhs*** ***The*** ***queue*** ***to*** ***copy*** ***elements*** ***from.***

***///*** \post ***All*** ***elements*** ***are*** ***deep*** ***copied*** ***to*** ***\*this.*** ***See*** ***copy*** ***constructor.***

*const* CircularQ& *operator*=(*const* CircularQ& rhs);

***///*** \brief ***Check*** ***if*** ***the*** ***queue*** ***is*** ***empty.***

***///*** \return ***Whether*** ***there*** ***are*** ***no*** ***elements*** ***in*** ***the*** ***queue***

bool **empty**() *const*;

***///*** \brief ***Add*** ***a*** ***value*** ***to*** ***the*** ***end*** ***of*** ***the*** ***queue.***

***///*** \param ***value*** ***The*** ***value*** ***to*** ***add*** ***into*** ***the*** ***queue.***

***///*** \post ***value*** ***is*** ***now*** ***at*** ***the*** ***end*** ***of*** ***the*** ***queue.***

void **enqueue**(*const* QueueElement& value);

***///*** \brief ***Outputs*** ***\*this***

***///*** \param ***out*** ***The*** ***stream*** ***to*** ***write*** ***to***

***///*** \post ***The*** ***contents*** ***of*** ***this*** ***list*** ***are*** ***written*** ***to*** ***out,*** ***space*** ***delimited.***

void **display**(ostream& out) *const*;

***///*** \brief ***Get*** ***the*** ***front*** ***of*** ***the*** ***queue.***

***///*** \returns ***The*** ***first*** ***element*** ***in*** ***the*** ***queue.*** ***(I.E*** ***been*** ***there*** ***the*** ***longest).***

QueueElement **front**() *const*;

***///*** \brief ***Remove*** ***the*** ***element*** ***at*** ***the*** ***front*** ***of*** ***the*** ***queue.***

***///*** \post ***Current*** ***this->front()*** ***is*** ***no*** ***longer*** ***in*** ***memory.***

void **dequeue**();

*private*:

***///*** \class ***Node***

***///*** \brief ***Internal*** ***storage*** ***for*** ***data*** ***elements.*** ***Not*** ***relevent*** ***outside***

***///*** ***CircularQueue*** ***class*** ***internals.***

*class* Node {

*public*:

***///*** \brief ***Constructor***

***///*** \param ***value*** ***The*** ***data*** ***value*** ***to*** ***assign*** ***to*** ***data***

***///*** \param ***link*** ***The*** ***Node*** ***\*this*** ***should*** ***link*** ***to.***

**Node**(QueueElement value, Node\* link = NULL) : data(value), next(link) {

}

QueueElement data; ***///>!*** ***Our*** ***data*** ***element***

Node\* next; ***///>!*** ***The*** ***next*** ***link*** ***in*** ***the*** ***chain***

};

*typedef* Node\* NodePointer;

NodePointer myBack; ***///>!*** ***Our*** ***view*** ***into*** ***the*** ***list.*** ***Use*** ***myBack->next*** ***for*** ***front***

};

***///*** \brief ***Output*** ***stream*** ***operator*** ***for*** ***CircularQ***

***///*** \param ***out*** ***The*** ***stream*** ***to*** ***write*** ***to.***

***///*** \param ***q*** ***The*** ***queue*** ***to*** ***display***

***///*** \post ***Writes*** ***all*** ***elements*** ***in*** ***q*** ***to*** ***out.*** ***See*** ***CircularQ::display*** ***for*** ***more.***

ostream& *operator*<<(ostream& out, *const* CircularQ& q);

#endif

*/\** *circq.cpp*

*\**

*\** *Defines* *CircularQ* *functions* *and* *associated* *operators.*

*\**

*\** *Johnathan* *Lee* *CSCI* *1107*

*\** *Lab* *10* *Due* *04/03/18*

*\*/*

#include "circq.h"

CircularQ::**CircularQ**() {

myBack = NULL;

}

*//* *Since* *it's* *just* *2* *statements,* *should* *really* *be* *inlined...*

CircularQ::**CircularQ**(*const* CircularQ& orig) {

myBack = NULL;

*//* *Since* *it's* *the* *same* *code* *and* *the* *(this* *==* *&rhs)* *still* *works...*

\**this* = orig;

}

CircularQ::~**CircularQ**() {

*if* (myBack) {

Node\* cur = myBack->next;

myBack->next = NULL;

*while* (cur != myBack) {

Node\* old = cur;

cur = cur->next;

*delete* old;

}

*delete* myBack; *//* *Putting* *out* *here* *to* *avoid* *problems* *with* *size()* *==* *1.*

}

myBack = NULL; *//* *Accomodate* *Microsoft*

}

*const* CircularQ& CircularQ::*operator*=(*const* CircularQ& rhs) {

*if* (*this* != &rhs) {

*this*->~CircularQ(); *//* *Delete* *old* *list*

Node\* curRhs = rhs.myBack->next;

*//* *Use* *our* *methods* *-* *think* *deeply* *of* *simple* *things.*

*//* *(I'm* *gonna* *have* *nightmares* *about* *that* *phrase)*

*do* {

enqueue(curRhs->data);

curRhs = curRhs->next;

} *while* (curRhs != rhs.myBack->next);

}

*return* \**this*;

}

bool CircularQ::**empty**() *const* {

*return* myBack == NULL; *//* *If* *myBack* *is* *null* *we* *have* *no* *nodes,* *so* *empty.*

}

void CircularQ::**enqueue**(*const* QueueElement& value) {

*if* (myBack) { *//* *!empty*

Node\* newNode = *new* Node(value, myBack->next);

myBack->next = newNode;

myBack = newNode;

} *else* {

myBack = *new* Node(value, myBack);

*//* *Because* *assigning* *it* *in* *the* *constructor* *would* *assign* *to* *NULL.*

*//* *You* *don't* *wanna* *know* *how* *long* *that* *one* *took* *me.*

myBack->next = myBack;

}

}

void CircularQ::**display**(ostream& out) *const* {

*if* (myBack) {

Node\* cur = myBack->next;

*do* {

out << cur->data << " ";

cur = cur->next;

} *while* (cur != myBack->next);

} *else* {

*//* *To* *help* *distinguish* *between* *no* *output* *and* *outputting* *blank* *queue.*

*/\*//\*/* out << "EMPTY\_CIRCULAR\_QUEUE ";

}

}

QueueElement CircularQ::**front**() *const* {

*return* myBack->next->data;

}

void CircularQ::**dequeue**() {

*if* (myBack->next == myBack) {

*delete* myBack;

myBack = NULL;

} *else* {

Node\* oldFront = myBack->next;

myBack->next = oldFront->next;

*delete* oldFront;

}

}

ostream& *operator*<<(ostream& out, *const* CircularQ& q) {

q.display(*out*);

*return* out;

}

#include <iostream>

#include "circq.h"

*using* *namespace* std;

int **main**() {

*//* *Note:* *I* *added* *some* *comments* *for* *clarity* *of* *testing,* *but* *file* *is* *otherwise*

*//* *unchanged.*

CircularQ q1;

q1.enqueue(10);

q1.enqueue(20);

q1.enqueue(30);

q1.display(*cout*); *//* *Expected:* *10* *20* *30*

cout << q1 << endl; *//* *Expected:* *10* *20* *30*

{

CircularQ q;

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

cout << "in { } for queue" << q << endl; *//* *Expected:* *10* *20* *30*

*//* *Expected:* *No* *crashes.* *(Please,* *compiler?)*

cout << "Testing destructor " << endl;

}

CircularQ q2(q1);

cout << "Testing copy constructor " << endl;

cout << q2 << endl; *//* *Expected:* *10* *20* *30*

CircularQ q3;

q3 = q2;

cout << "Testing assignment operator." << endl;

cout << q3 << endl; *//* *Expected:* *10* *20* *30*

cout << "Testing front method: " << q3.front() << endl; *//* *Expected:* *10*

*//* *Expected:*

*//* *20* *30*

*//* *30*

*//* *EMPTY\_CIRCULAR\_QUEUE* *(Or* *nothing,* *if* *that* *output* *is* *disabled.)*

*while* (!q3.empty()) {

q3.dequeue();

cout << q3 << endl;

} *//* *end* *while*

*return* 0;

} *//* *end* *main*