МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №4**

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: Меджидли Махмуд Ибрагим оглы, группа М80-208Б-20

Преподаватель: Дорохов Евгений Павлович, каф. 806

**Задание:** Спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру. Классы должны удовлетворять следующим правилам:

* Требования к классу фигуры аналогичны требованиям из лабораторной работы 2.
* Классы фигур должны содержать набор следующих методов:
  + Перегруженный оператор ввода координат вершин фигуры из потока std::istream (>>)
  + Перегруженный оператор вывода в поток std::ostream (<<)
  + Оператор копирования (=)
  + Оператор сравнения с такими же фигурами (==)
* Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке).
* Класс-контейнер должен содержать набор следующих методов:
  + Length() – возвращает количество элементов в контейнере
  + Empty() – для пустого контейнера возвращает 1, иначе – 0
  + First() – возвращает первый (левый) элемент списка
  + Last() – возвращает последний (правый) элемент списка
  + InsertFirst(elem) – добавляет элемент в начало списка
  + RemoveFirst() – удаляет элемент из начала списка
  + InsertLast(elem) – добавляет элемент в конец списка
  + RemoveLast() – удаляет элемент из конца списка
  + Insert(elem, pos) – вставляет элемент на позицию pos
  + Remove(pos) – удаляет элемент, находящийся на позиции pos
  + Clear() – удаляет все элементы из списка
  + operator<< – выводит список поэлементно в поток вывода (слева направо)

Нельзя использовать:

* Стандартные контейнеры std.
* Шаблоны (template).
* Различные варианты умных указателей (shared\_ptr, weak\_ptr).

Программа должна позволять:

* Вводить произвольное количество фигур и добавлять их в контейнер.
* Распечатывать содержимое контейнера.
* Удалять фигуры из контейнера.

**Вариант №12:**

* + Фигура: Трапеция (Trapezoid)
  + Контейнер: Очередь (TQueue)

**Описание программы:**

Исходный код разделён на 10 файлов:

* figure.h – описание класса фигуры
* point.h – описание класса точки
* point.cpp – реализация класса точки
* trapezoid.h – описание класса трапеция
* trapezoid.cpp – реализация класса трапеция
* tqueue\_item.h – описание элемента очереди
* tqueue\_item.cpp – реализация элемента очереди
* tqueue.h – описание очереди
* tqueue.cpp – реализация очереди
* main.cpp – основная программа

**Дневник отладки:**

Программа в отладке не нуждалась.

**Вывод**:

Данная лабораторная работа научила меня, как реализовать контейнер для хранения трапеции по принципам ООП и на С++. Подобную работу я делал на 1 курсе на языке Си. Эта лабораторная работа помогла мне лучше разобраться в инкапсуляции.

**Исходный код:**

**point.h:**

#ifndef POINT\_H

#define POINT\_H

#include <iostream>

class Point {

public:

Point();

Point(std::istream &is);

Point(double x, double y);

double dist(Point& other);

void SetX(double x);

void SetY(double y);

double GetX();

double GetY();

friend std::istream& operator>>(std::istream& is, Point& p);

friend std::ostream& operator<<(std::ostream& os, Point& p);

friend std::ostream& operator<<(std::ostream& os, const Point& p);

public:

double x\_;

double y\_;

};

#endif // POINT\_H  
  
 **point.cpp:**

#include "point.h"

#include <iostream>

#include <cmath>

Point::Point() : x\_(0.0), y\_(0.0) {}

Point::Point(double x, double y) : x\_(x), y\_(y) {}

Point::Point(std::istream &is) {

is >> x\_ >> y\_;

}

void Point::SetX(double x) {

this->x\_ = x;

}

void Point::SetY(double y) {

this->y\_ = y;

}

double Point::GetX() {

return this->x\_;

}

double Point::GetY() {

return this->y\_;

}

double Point::dist(Point& other) {

double dx = (other.x\_ - x\_);

double dy = (other.y\_ - y\_);

return std::sqrt(dx\*dx + dy\*dy);

}

std::istream& operator>>(std::istream& is, Point& p) {

is >> p.x\_ >> p.y\_;

return is;

}

std::ostream& operator<<(std::ostream& os, Point& p) {

os << "(" << p.x\_ << ", " << p.y\_ << ")";

return os;

}

std::ostream& operator<<(std::ostream& os, const Point& p) {

os << "(" << p.x\_ << ", " << p.y\_ << ")";

return os;

}

**figure.h:**

#ifndef FIGURE\_H

#define FIGURE\_H

#include <iostream>

class Figure {

public:

virtual size\_t VertexesNumber() = 0;

virtual double Area() = 0;

//virtual void Print(std::ostream& os) = 0;

virtual ~Figure() {};

};

#endif // FIGURE\_H

**trapezoid.h:**

#ifndef TRAPEZOID\_H

#define TRAPEZOID\_H

#include "figure.h"

#include <iostream>

#include "point.h"

class Trapezoid : public Figure {

public:

Trapezoid();

Trapezoid(double a, double b, double c, double d);

Trapezoid(const Trapezoid& other);

friend std::istream& operator>>(std::istream& is, Trapezoid& obj);

friend std::ostream& operator<<(std::ostream& os, const Trapezoid& obj);

Trapezoid& operator=(const Trapezoid& right);

bool operator==(const Trapezoid& right);

virtual ~Trapezoid();

size\_t VertexesNumber();

double Area();

public:

double len\_ab, len\_bc, len\_cd, len\_da;

Point a\_, b\_, c\_, d\_;

};

#endif // TRAPEZOID\_H

**trapezoid.cpp:**

#include "trapezoid.h"

#include <cmath>

Trapezoid::Trapezoid()

: len\_ab(0.0),

len\_bc(0.0),

len\_cd(0.0),

len\_da(0.0) {

}

Trapezoid::Trapezoid(double ab, double bc, double cd, double da)

: len\_ab(ab),

len\_bc(bc),

len\_cd(cd),

len\_da(da) {

}

Trapezoid::Trapezoid(const Trapezoid& other)

: Trapezoid(other.len\_ab, other.len\_bc, other.len\_cd, other.len\_da) {

}

std::istream& operator>>(std::istream& is, Trapezoid& obj) {

std::cout << "Enter points: ";

is >> obj.a\_;

is >> obj.b\_;

is >> obj.c\_;

is >> obj.d\_;

obj.len\_ab = obj.a\_.dist(obj.b\_);

obj.len\_bc = obj.b\_.dist(obj.c\_);

obj.len\_cd = obj.c\_.dist(obj.d\_);

obj.len\_da = obj.d\_.dist(obj.a\_);

return is;

} //

std::ostream& operator<<(std::ostream& os, const Trapezoid& obj) {

std::cout << "Trapezoid: ";

os << obj.a\_; std::cout << " ";

os << obj.b\_; std::cout << " ";

os << obj.c\_; std::cout << " ";

os << obj.d\_; std::cout << std::endl;

return os;

}

Trapezoid& Trapezoid::operator=(const Trapezoid& other) {

if (this == &other)

return \*this;

len\_ab = other.len\_ab;

len\_bc = other.len\_bc;

len\_cd = other.len\_cd;

len\_da = other.len\_da;

a\_.x\_ = other.a\_.x\_;

a\_.y\_ = other.a\_.y\_;

b\_.x\_ = other.b\_.x\_;

b\_.y\_ = other.b\_.y\_;

c\_.x\_ = other.c\_.x\_;

c\_.y\_ = other.c\_.y\_;

d\_.x\_ = other.d\_.x\_;

d\_.x\_ = other.d\_.x\_;

std::cout << "Trapezoid copied" << std::endl;

return \*this;

} //

bool Trapezoid::operator==(const Trapezoid& other) {

if (this->len\_ab == other.len\_ab &&

this->len\_bc == other.len\_bc &&

this->len\_cd == other.len\_cd &&

this->len\_da == other.len\_da) {

std::cout << "Trapezoids are equal" << std::endl;

return 1;

} else {

std::cout << "Trapezoids are not equal" << std::endl;

return 0;

}

} //

size\_t Trapezoid::VertexesNumber() {

return 4;

}

double Trapezoid::Area() {

double p = (len\_ab + len\_bc + len\_cd + len\_da) / 2;

return (len\_bc + len\_da) \*

std::sqrt((p - len\_bc) \*

(p - len\_da) \*

(p - len\_da - len\_ab) \*

(p - len\_da - len\_cd)) /

std::abs(len\_bc - len\_da);

}

Trapezoid::~Trapezoid() {

std::cout << "Trapezoid deleted" << std::endl;

}   
 **tqueue\_item.h:**

#ifndef TQUEUE\_ITEM\_H

#define TQUEUE\_ITEM\_H

#include "trapezoid.h"

class TQueueItem {

public:

TQueueItem();

TQueueItem(const Trapezoid& trapezoid);

TQueueItem(const TQueueItem& other);

TQueueItem\* SetNext(TQueueItem\* next);

TQueueItem\* GetNext();

Trapezoid GetTrapezoid() const;

friend std::ostream& operator<<(std::ostream& os, const TQueueItem& obj);

virtual ~TQueueItem();

public:

Trapezoid trapezoid;

TQueueItem \*next;

};

#endif **tqueue\_item.cpp:**  
  
#include "tqueue\_item.h"

#include <iostream>

TQueueItem::TQueueItem() : trapezoid(Trapezoid()), next(nullptr) {}

TQueueItem::TQueueItem(const Trapezoid& trapezoid) {

this->trapezoid = trapezoid;

this->next = nullptr;

std::cout << "Queue item: created" << std::endl;

}

TQueueItem::TQueueItem(const TQueueItem& other) {

this->trapezoid = other.trapezoid;

this->next = other.next;

std::cout << "Queue item: copied" << std::endl;

}

TQueueItem\* TQueueItem::SetNext(TQueueItem\* next) {

TQueueItem\* old = this->next;

this->next = next;

return old;

}

Trapezoid TQueueItem::GetTrapezoid() const {

return this->trapezoid;

}

TQueueItem\* TQueueItem::GetNext() {

return this->next;

}

TQueueItem::~TQueueItem() {

std::cout << "Queue item: deleted" << std::endl;

if (next != nullptr)

delete next;

}

std::ostream& operator<<(std::ostream& os, const TQueueItem& obj) {

os << (const\_cast<TQueueItem&>(obj)).trapezoid.Area();

return os;

} **tqueue.h:**

#ifndef TQUEUE\_H

#define TQUEUE\_H

#include "trapezoid.h"

#include "tqueue\_item.h"

class TQueue {

public:

TQueue();

TQueue(const TQueue& other);

void Push(const Trapezoid& trapezoid);

void Pop();

Trapezoid& Top();

bool Empty();

size\_t Length();

friend std::ostream& operator<<(std::ostream& os, const TQueue& queue);

void Clear();

virtual ~TQueue();

private:

TQueueItem \*head, \*tail;

};

#endif // TQUEUE\_H **tqueue.cpp:**

#include "tqueue.h"

#include <vector>

TQueue::TQueue() : head(nullptr), tail(nullptr) {

std::cout << "Default queue created" << std::endl;

}

TQueue::TQueue(const TQueue& other) {

head = other.head;

tail = other.tail;

}

void TQueue::Push(const Trapezoid& trapezoid) {

TQueueItem\* other = new TQueueItem(trapezoid);

if (tail == nullptr) {

head = tail = other;

std::cout << "Added one trapezoid to tail. " << "Coordinates: " << other->trapezoid << ". Area = " << other->trapezoid.Area() << std::endl;

return;

}

tail->next = other;

tail = other;

tail->next = nullptr;

std::cout << "Added one trapezoid to tail. " << "Coordinates: " << other->trapezoid << ". Area = " << other->trapezoid.Area() << std::endl;

}

void TQueue::Pop() {

if (head == nullptr)

return;

std::cout << "Removed one trapezoid " << head->trapezoid << " from head" << std::endl;

TQueueItem \*temp = head;

head = head->next;

delete temp;

if (head == nullptr)

tail = nullptr;

}

Trapezoid& TQueue::Top() {

Trapezoid& out = head->trapezoid;

return out;

}

bool TQueue::Empty() {

return (head == nullptr) && (tail == nullptr);

}

size\_t TQueue::Length() {

if (head == nullptr && tail == nullptr)

return 0;

TQueueItem \*temp = head;

int counter = 0;

while (temp != tail->GetNext()) {

temp = temp->GetNext();

counter++;

}

return counter;

}

std::ostream& operator<<(std::ostream& os, const TQueue& queue) {

TQueueItem \*temp = queue.head;

std::vector<TQueueItem \*> v;

os << "Queue: ";

os << "=> ";

while (temp != nullptr) {

v.push\_back(temp);

temp = temp->GetNext();

}

for (int i = v.size() - 1; i >= 0; --i)

os << \*v[i] << " ";

os << "=>";

return os;

}

void TQueue::Clear() {

for (int i = 0; i < this->Length(); i++) {

this->Pop();

}

std::cout << "Queue was cleared but still exist" << std::endl;

}

TQueue::~TQueue() {

std::cout << "Queue was deleted" << std::endl;

} **main.cpp:**#include <iostream>

#include "trapezoid.h"

#include "tqueue.h"

int main(int argc, char\*\* argv) {

TQueue queue;

Trapezoid tr;

std::cout << "Enter n: ";

int n; std::cin >> n;

for (int i = 0; i < n; i++) {

std::cin >> tr;

std::cout << tr << std::endl;

queue.Push(tr);

std::cout << queue;

std::cout << std::endl;

std::cout << "Length: " << queue.Length() << std::endl;

}

Trapezoid tr1;

tr1 = queue.Top();

std::cout << tr1;

return 0;

}