МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №5

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:

Дополнить класс-контейнер из лабораторной работы №4 умными указателями.

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 трапеции
- trapezoid.cpp – реализация класса трапеции
- tqueue_item.h – описание элемента очереди
- tqueue_item.cpp – реализация элемента очереди
- tqueue.h – описание очереди
- queue.cpp – реализация очереди
- main.cpp – основная программа

Дневник отладки: Ошибок не возникло.

Вывод:

В ходе проделанной я получил практические навыки по созданию и использованию умных указателей. Оказывается, умные указатели намного лучше обычных указателей, ведь они сами удаляются, что обеспечивает гарантию отсутствия утечки памяти. Любой хороший программист должен уметь работать с умными указателями и я рад, что данная лабораторная работа научила меня этому.

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 <memory>
#include "point.h"


class Trapezoid : public Figure {
public:
   Trapezoid();
   Trapezoid(double a, double b, double c, double d);
   Trapezoid(std::shared_ptr<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(std::shared_ptr<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 <memory>
#include "trapezoid.h"

class TQueueItem {
public:
```

```
  TQueueItem(const std::shared_ptr<Trapezoid>& trapezoid);
  TQueueItem(const TQueueItem& other);

  std::shared_ptr<TQueueItem> SetNext(std::shared_ptr<TQueueItem>& next);
  std::shared_ptr<TQueueItem> GetNext();

  std::shared_ptr<Trapezoid> GetTrapezoid() const;

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

  virtual ~TQueueItem();

public:
  std::shared_ptr<Trapezoid> trapezoid;
  std::shared_ptr<TQueueItem> next;
};

#endif // TQUEUE_ITEM_H
```

tqueue_item.cpp:

```
#include "tqueue_item.h"
#include <iostream>

TQueueItem::TQueueItem(const std::shared_ptr<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;
}

std::shared_ptr<TQueueItem> TQueueItem::SetNext(std::shared_ptr<TQueueItem> &next) { /////////////// added &
	std::shared_ptr<TQueueItem> old = this->next;
	this->next = next;
	return old;
}

std::shared_ptr<Trapezoid> TQueueItem::GetTrapezoid() const {
	return this->trapezoid;
}


std::shared_ptr<TQueueItem> TQueueItem::GetNext() {
	return this->next;
}

TQueueItem::~TQueueItem() {
	std::cout << "Queue item: deleted" << std::endl;
}

std::ostream& operator<<(std::ostream& os, const TQueueItem& obj) {
	os << obj.trapezoid->Area();
	return os;
}
```

tqueue.h:

```
#ifndef TQUEUE_H
#define TQUEUE_H

#include "tqueue_item.h"
#include <memory>

class TQueue {
public:
    TQueue();
    TQueue(const TQueue& other);
    void Push(std::shared_ptr<Trapezoid> &&trapezoid);
    void Pop();
    std::shared_ptr<Trapezoid>& Top();
    bool Empty();
    size_t Length();
    friend std::ostream& operator<<(std::ostream& os, const TQueue& queue);
    void Clear();
    virtual ~TQueue();
private:
    std::shared_ptr<TQueueItem> head, tail;
};

#endif // TQUEUE_H
```

TBinaryTree.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;
    std::cout << "Queue copied" << std::endl;
}


void TQueue::Push(std::shared_ptr<Trapezoid> &&trapezoid) {
    std::shared_ptr<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->SetNext(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 from head."  << "Coordinates: " << *head->trapezoid << ". Area = " << head->trapezoid-
>Area() << std::endl;

        head = head->GetNext();

        if (head == nullptr)
            tail = nullptr;
    }

    std::shared_ptr<Trapezoid>& TQueue::Top() {
        return head->trapezoid;
    }

    bool TQueue::Empty() {
        return (head == nullptr) && (tail == nullptr);
    }

    size_t TQueue::Length() {
        if (head == nullptr && tail == nullptr)
            return 0;
        std::shared_ptr<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) {
        std::shared_ptr<TQueueItem> temp = queue.head;
        std::vector<std::shared_ptr<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;
    }
```