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: *Тихонов Фёдор Андреевич, группа М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

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

**Вариант №26:**

* + Фигура: Квадрат
  + Контейнер: Очередь

**Описание программы:**

Исходный код разделён на 10 файлов:

* figure.h – описание класса фигуры
* point.h – описание класса точки
* point.cpp – реализация класса точки
* square.h – описание класса квадрата
* square.cpp – реализация класса квадрата
* TQueueItem.h – описание элемента очереди
* TQueueItem.cpp – реализация элемента очереди
* TQueueItem.h – описание очереди
* TQueueItem.cpp – реализация очереди
* main.cpp – основная программа

**Дневник отладки:**

При замене на умные указатели ошибок не возникло.

**Вывод:**  
 В данной лабораторной работе я познакомился с умными указателями, поэтому изменил реализацию классов фигур и класса-контейнера очередь, и для каждого из классов - функции, заменив переменные обычных указателей умными.

**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 fx();

double fy();

double dist(Point& other);

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

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

private:

double x\_;

double y\_;

};

#endif //POINT\_H

**point.cpp:**

#include <iostream>

#include <cmath>

#include "point.h"

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\_;

}

double Point::fx(){

return x\_;

};

double Point::fy(){

return 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;

}

**figure.h:**

#ifndef FIGURE\_H

#define FIGURE\_H

#include <iostream>

#include "point.h"

class Figure {

public:

virtual size\_t VertexesNumber() = 0;

virtual double Area() = 0;

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

~Figure() {};

};

#endif //FIGURE\_H

**square.h:**

#ifndef SQUARE\_H

#define SQUARE\_H

#include "point.h"

#include "figure.h"

#include <memory>

class Square : public Figure {

public:

Square();

Square(Point a, Point b, Point c, Point d);

Square(std::istream& is);

size\_t VertexesNumber() override;

double Area() override;

void Print(std::ostream& os) override;

friend std::ostream& operator<<(std::ostream& os, const Square& ot);

friend std::istream& operator>>(std::istream& is, Square& ot);

Square& operator=(const Square& ot);

Square& operator=(const std::shared\_ptr<Square> &ot);

bool operator==(Square& ot);

private:

Point a\_;

Point b\_;

Point c\_;

Point d\_;

};

#endif //SQUARE\_H

**square.cpp:**

#include <iostream>

#include "point.h"

#include "square.h"

#include <memory>

Square::Square() : a\_(Point()), b\_(Point()), c\_(Point()), d\_(Point()) { }

Square::Square(Point a, Point b, Point c, Point d) : a\_(a), b\_(b), c\_(c), d\_(d) { }

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

is >> a\_ >> b\_ >> c\_ >> d\_;

}

std::ostream& operator<<(std::ostream &os, const Square &ot) {

os << "Square: " << ot.a\_ << " " << ot.b\_ << " " << ot.c\_ << " " << ot.d\_;

return os;

}

std::istream &operator>>(std::istream &is, Square &it) {

is >> it.a\_ >> it.b\_ >> it.c\_ >> it.d\_;

return is;

}

Square& Square::operator=(const Square &ot) {

this->a\_ = ot.a\_;

this->b\_ = ot.b\_;

this->c\_ = ot.c\_;

this->d\_ = ot.d\_;

return \*this;

}

Square& Square::operator=(const std::shared\_ptr<Square> &ot) {

this->a\_ = ot->a\_;

this->b\_ = ot->b\_;

this->c\_ = ot->c\_;

this->d\_ = ot->d\_;

return \*this;

}

bool Square::operator==(Square &ot) {

int flag = 0;

if (this->a\_ == ot.a\_)

flag++;

if (this->b\_ == ot.b\_)

flag++;

if (this->c\_ == ot.c\_)

flag++;

if (this->d\_ == ot.d\_)

flag++;

return (flag == 4);

}

size\_t Square::VertexesNumber() {

return 4;

}

double Square::Area() {

return a\_.dist(b\_) \* a\_.dist(b\_);

}

void Square::Print(std::ostream& os) {

os << "Square: " << a\_ << " " << b\_ << " " << c\_ << " " << d\_ << std::endl;

}

**TQueueItem.h:**

#ifndef FIGURE\_H\_TQUEUEITEM\_H

#define FIGURE\_H\_TQUEUEITEM\_H

#include "square.h"

#include <memory>

class TQueueItem {

public:

TQueueItem(const std::shared\_ptr<Square> &square);

TQueueItem(const std::shared\_ptr<TQueueItem> &other);

~TQueueItem();

friend std::ostream& operator<<(std::ostream& os, const std::shared\_ptr<TQueueItem> &poly);

public:

std::shared\_ptr<Square> square;

std::shared\_ptr<TQueueItem> next;

};

#endif //FIGURE\_H\_TQUEUEITEM\_H

**TQueueItem.cpp:**

#include "TQueueItem.h"

#include <iostream>

TQueueItem::TQueueItem(const std::shared\_ptr<Square> &square) {

this->square = square;

this->next = nullptr;

}

TQueueItem::TQueueItem(const std::shared\_ptr<TQueueItem> &other) {

this->square = other->square;

this->next = other->next;

}

std::ostream& operator<<(std::ostream& os, const std::shared\_ptr<TQueueItem> &poly) {

os << "(" << poly->square << ")" << std::endl;

return os;

}

TQueueItem::~TQueueItem() = default;

**TQueue.h:**

#ifndef FIGURE\_H\_TQUEUE\_H

#define FIGURE\_H\_TQUEUE\_H

#include "TQueueItem.h"

class TQueue {

public:

TQueue();

TQueue(const TQueue& other);

void Push(const std::shared\_ptr<Square> &&square);

void Pop();

std::shared\_ptr<Square> Top();

bool Empty();

size\_t Length();

friend std::ostream& operator<<(std::ostream& os, const TQueue& queue); // "=> Sn Sn-1 ... S1 =>"

void Clear();

~TQueue();

private:

size\_t len;

std::shared\_ptr<TQueueItem> head;

std::shared\_ptr<TQueueItem> tail;

};

#endif //FIGURE\_H\_TQUEUE\_H

**TQueue.cpp:**

#include "TQueue.h"

TQueue::TQueue() : head(nullptr), tail(nullptr), len(0) { }

TQueue::TQueue(const TQueue& other) {

head = other.head;

tail = other.tail;

len = other.len;

}

void TQueue::Push(const std::shared\_ptr<Square> &&square) {

std::shared\_ptr<TQueueItem> new\_tail =

std::make\_shared<TQueueItem>(TQueueItem(square));

if (head != nullptr)

tail->next = new\_tail, tail = new\_tail;

else if (len == 1)

head->next = new\_tail, tail = new\_tail;

else

head = tail = new\_tail;

len++;

}

void TQueue::Pop() {

if (len)

head = head->next, len--;

}

std::shared\_ptr<Square> TQueue::Top() {

if (!len)

return head->square;

}

bool TQueue::Empty() {

return (len == 0);

}

size\_t TQueue::Length() {

return len;

}

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

std::shared\_ptr<TQueueItem> item = queue.head;

double sq[queue.len];

for (int i = 0; i < (int)queue.len; i++) {

sq[i] = item->square->Area();

item = item->next;

}

os.precision(5);

os << "=> ";

for (int i = (int)queue.len - 1; i >= 0; i--) {

os << sq[i] << " ";

}

os << "=>";

return os;

}

void TQueue::Clear() {

std::shared\_ptr<TQueueItem> elem = head;

std::shared\_ptr<TQueueItem> fore = head;

while (elem) {

fore.reset();

fore = elem;

elem = elem->next;

}

len = 0;

}

TQueue::~TQueue() { }

**main.cpp:**

#include <iostream>

#include "point.h"

#include "figure.h"

#include "square.h"

#include "TQueue.h"

void menu() {

using namespace std;

cout << "Enter 0 to exit\n";

cout << "Enter 1 to print lenght of queue\n";

cout << "Enter 2 to clear the queue\n";

cout << "Enter 3 to know if the queue is empty\n";

cout << "Enter 4 to pop the first element from queue\n";

cout << "Enter 5 to push new element to queue\n";

cout << "Enter 6 to print queue\n";

cout << "Enter 7 to print the first element in queue\n";

}

int main() {

auto \*a = new TQueue;

int n = -1;

menu();

while (n != 0) {

std::cin >> n;

if (n == 1) {

std::cout << "Lenght of queue is " << a->Length() << std::endl;

}

if (n == 2) {

a->Clear();

std::cout << "Done" << std::endl;

}

if (n == 3) {

if (a->Empty())

std::cout << "Queue is empty" << std::endl;

else

std::cout << "Queue is not empty" << std::endl;

}

if (n == 4) {

a->Pop();

std::cout << "Done" << std::endl;

}

if (n == 5) {

std::cout << "Please, enter coordinates of Square" << std::endl;

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

std::cin >> a\_ >> b\_;

std::cin >> c\_ >> d\_;

a->Push(std::make\_shared<Square>(Square(a\_, b\_, c\_, d\_)));

std::cout << "Done" << std::endl;

}

if (n == 6) {

std::cout << \*a << std::endl;

}

if (n == 7) {

try {

if (a->Empty())

throw "No elements in queue";

std::cout << \*a->Top() << std::endl;

}

catch (const char \*exception) {

std::cerr << "ERROR: " << exception << std::endl;

}

}

}

return 0;

}

**Пример работы:**

Enter 0 to exit

Enter 1 to print lenght of queue

Enter 2 to clear the queue

Enter 3 to know if the queue is empty

Enter 4 to pop the first element from queue

Enter 5 to push new element to queue

Enter 6 to print queue

Enter 7 to print first element of queue

1

Lenght of queue is 0

4

Done

6

=> =>

7

ERROR: No elements in queue

2

Done

5

Please, enter coordinates of Square

1 1 1 1 1 1 1 1

Done

5

Please, enter coordinates of Square

2 2 2 2 2 2 2 2

Done

1

Lenght of queue is 2

7

Square: (1, 1) (1, 1) (1, 1) (1, 1)

4

Done

1

Lenght of queue is 1

7

Square: (2, 2) (2, 2) (2, 2) (2, 2)

3

Queue is not empty

2

Done

1

Lenght of queue is 0

3

Queue is empty

0

Process finished with exit code 0