МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №7**

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:**

Используя структуры данных, разработанные для лабораторной работы №6, спроектировать и разработать итератор для динамической структуры данных.

Итератор должен быть разработан в виде шаблона, должен работать со всеми типами фигур, согласно варианту задания.

Итератор должен позволять использовать структуру данных в операторах типа for:

for (auto i : list) {

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

}

**Вариант №26:**

* + Фигуры: Квадрат, Прямоугольник, Трапеция
  + Контейнер: Очередь

**Описание программы:**

Исходный код разделён на 15 файлов:

* figure.h – описание класса фигуры
* point.h – описание класса точки
* point.cpp – реализация класса точки
* square.h – описание класса квадрата (наследуется от фигуры)
* square.cpp – реализация класса квадрата
* rectangle.h – описание класса прямоугольника (наследуется от фигуры)
* rectangle.cpp – реализация класса прямоугольника
* trapezoid.h – описание класса трапеции (наследуется от фигуры)
* trapezoid.cpp – реализация класса трапеции
* TQueueItem.h – описание элемента очереди
* TQueueItem.cpp – реализация элемента очереди
* TQueueItem.h – описание очереди
* TQueueItem.cpp – реализация очереди
* TIterator.h – реализация итератора
* main.cpp – основная программа

**Дневник отладки:**

Неправильно написал часть, связанную с выводом данных при использовании цикла for, печаталось не то, что ожидалось, исправлено

**Вывод:**  
 В данной лабораторной работе я познакомился с итераторами: они позволяют легко реализовать обход всех элементов некоторой структуры данных, а также позволяют использовать цикл range-based-for для самописных структур. Поэтому я уверен, что знания об итераторах, полученные в этой лабораторной работе, обязательно пригодятся мне в будущем.

**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

**rectangle.h:**

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <iostream>

#include "point.h"

#include "figure.h"

class Rectangle : Figure {

public:

Rectangle();

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

Rectangle(std::istream& is);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

private:

Point a\_;

Point b\_;

Point c\_;

Point d\_;

};

#endif //RECTANGLE\_H

**rectangle.cpp:**

#include <iostream>

#include "point.h"

#include "rectangle.h"

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

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

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

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

}

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

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

}

size\_t Rectangle::VertexesNumber(){

return 4;

}

double Rectangle::Area(){

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

}

**square.h:**

#ifndef SQUARE\_H

#define SQUARE\_H

#include <iostream>

#include "point.h"

#include "figure.h"

class Square : Figure {

public:

Square();

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

Square(std::istream& is);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

private:

Point a\_;

Point b\_;

Point c\_;

Point d\_;

};

#endif //SQUARE\_H

**square.cpp:**

#include <iostream>

#include "point.h"

#include "square.h"

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\_;

}

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

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

}

size\_t Square::VertexesNumber() {

return 4;

}

double Square::Area() {

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

}

**trapezoid.h:**

#ifndef TRAPEZOID\_H

#define TRAPEZOID\_H

#include <iostream>

#include "point.h"

#include "figure.h"

class Trapezoid : Figure {

public:

Trapezoid();

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

Trapezoid(std::istream& is);

size\_t VertexesNumber();

double Area();

void Print(std::ostream& os);

private:

Point a\_;

Point b\_;

Point c\_;

Point d\_;

};

#endif //TRAPEZOID\_H

**trapezoid.cpp:**

#include <iostream>

#include <cmath>

#include "point.h"

#include "trapezoid.h"

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

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

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

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

}

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

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

}

size\_t Trapezoid::VertexesNumber() {

return 4;

}

double Trapezoid::Area() {

double det1 = a\_.fx() \* b\_.fy() + b\_.fx() \* c\_.fy() + c\_.fx() \* d\_.fy() + d\_.fx() \* a\_.fy();

double det2 = a\_.fy() \* b\_.fx() + b\_.fy() \* c\_.fx() + c\_.fy() \* d\_.fx() + d\_.fy() \* a\_.fx();

double det = abs(det1 - det2);

return 0.5 \* det;

} // Gauss's Area Calculation Formula (Shoelace Theorem)

**TQueueItem.h:**

#ifndef FIGURE\_H\_TQUEUEITEM\_H

#define FIGURE\_H\_TQUEUEITEM\_H

#include "square.h"

#include "trapezoid.h"

#include "rectangle.h"

#include <memory>

template <class T> class TQueueItem {

public:

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

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

~TQueueItem();

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

public:

std::shared\_ptr<T> polygon;

std::shared\_ptr<TQueueItem<T>> next;

};

#define TQUEUEITEM\_FUNCTIONS

#include "TQueueItem.cpp"

#endif //FIGURE\_H\_TQUEUEITEM\_H

**TQueueItem.cpp:**

#ifndef TQUEUEITEM\_FUNCTIONS

#include "TQueueItem.h"

#else

template <class T>

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

this->polygon = poly;

this->next = nullptr;

}

template <class T>

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

this->polygon = other->polygon;

this->next = other->next;

}

template <class A>

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

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

return os;

}

template <class T>

TQueueItem<T>::~TQueueItem() = default;

#endif

**TQueue.h:**

#ifndef FIGURE\_H\_TQUEUE\_H

#define FIGURE\_H\_TQUEUE\_H

#include "TQueueItem.h"

#include "TIterator.h"

#include <iostream>

template <class T>

class TQueue {

public:

TQueue();

TQueue(const TQueue<T>& other);

void Push(const std::shared\_ptr<T> &&polygon);

void Pop();

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

bool Empty();

size\_t Length();

TIterator<TQueueItem<T>, T> begin();

TIterator<TQueueItem<T>, T> end();

template<class A>

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

void Clear();

~TQueue();

private:

size\_t len;

std::shared\_ptr<TQueueItem<T>> head;

std::shared\_ptr<TQueueItem<T>> tail;

};

#define TQUEUE\_FUNCTIONS

#include "TQueue.cpp"

#endif //FIGURE\_H\_TQUEUE\_H

**TQueue.cpp:**

#ifndef TQUEUE\_FUNCTIONS

#include "TQueue.h"

#else

template <class T>

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

template <class T>

TQueue<T>::TQueue(const TQueue<T>& other) {

head = other.head;

tail = other.tail;

len = other.len;

}

template <class T>

void TQueue<T>::Push(const std::shared\_ptr<T> &&polygon) {

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

std::make\_shared<TQueueItem<T>>(TQueueItem<T>(polygon));

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++;

}

template <class T>

void TQueue<T>::Pop() {

if (len)

head = head->next, len--;

}

template <class T>

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

if (len)

return head->polygon;

}

template <class T>

bool TQueue<T>::Empty() {

return (len == 0);

}

template <class T>

size\_t TQueue<T>::Length() {

return len;

}

template <class T>

TIterator<TQueueItem<T>, T> TQueue<T>::begin() {

return TIterator<TQueueItem<T>, T>(head);

}

template <class T> TIterator<TQueueItem<T>, T> TQueue<T>::end() {

return TIterator<TQueueItem<T>, T>(nullptr);

}

template <class T>

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

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

double sq[queue.len];

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

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

item = item->next;

}

os.precision(5);

os << "=> ";

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

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

}

os << "=>";

return os;

}

template <class T>

void TQueue<T>::Clear() {

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

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

while (elem) {

fore.reset();

fore = elem;

elem = elem->next;

}

len = 0;

}

template <class T>

TQueue<T>::~TQueue() { }

#endif

**TIterator.h:**

#ifndef LAB5\_TITERATOR\_H

#define LAB5\_TITERATOR\_H

#include "square.h"

#include <iostream>

template <class item, class T> class TIterator {

public:

TIterator(std::shared\_ptr<item> n) {

item\_ptr = n;

}

std::shared\_ptr<T> operator \*() {

return item\_ptr->polygon;

}

std::shared\_ptr<T> operator ->() {

return item\_ptr->polygon;

}

void operator ++() {

item\_ptr = item\_ptr->next;

}

TIterator operator ++(int) {

TIterator iter(\*this);

++(\*this);

return iter;

}

bool operator ==(TIterator const& i) {

return (item\_ptr == i.item\_ptr);

}

bool operator !=(TIterator const& i) {

return (item\_ptr != i.item\_ptr);

}

private:

std::shared\_ptr<item> item\_ptr;

};

#endif //LAB5\_TITERATOR\_H

**main.cpp:**

#include <iostream>

#include <memory>

#include "point.h"

#include "figure.h"

#include "square.h"

#include "trapezoid.h"

#include "rectangle.h"

#include "TQueue.h"

void menu() {

using namespace std;

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

cout << "Enter 1 to print length 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 51 to push new Square to queue\n";

cout << "Enter 52 to push new Rectangle to queue\n";

cout << "Enter 53 to push new Trapezoid to queue\n";

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

}

int main() {

TQueue<Figure> a;

std::shared\_ptr<Figure> ptr;

int n = -1;

menu();

while (n != 0) {

std::cin >> n;

if (n == 1) {

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

}

if (n == 2) {

a.Clear();

std::cout << "Cleared" << 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 << "Popped" << std::endl;

}

if (n == 51) {

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

a.Push( std::make\_shared<Square>(Square(std::cin)));

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

}

if (n == 52) {

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

a.Push( std::make\_shared<Rectangle>(Rectangle(std::cin)));

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

}

if (n == 53) {

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

a.Push( std::make\_shared<Trapezoid>(Trapezoid(std::cin)));

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

}

if (n == 6) {

std::cout << a << std::endl;

}

if (n == 7) {

for (auto x : a) {

x->Print(std::cout);

}

}

}

return 0;

}

**Пример работы:**

Enter 0 to exit

Enter 1 to print length 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 51 to push new Square to queue

Enter 52 to push new Rectangle to queue

Enter 53 to push new Trapezoid to queue

Enter 6 to print queue

51

Please, enter coordinates of Square

0 1

1 0

0 0

1 1

Done

51

Please, enter coordinates of Square

1 0

0 1

0 0

1 1

Done

52

Please, enter coordinates of Rectangle

2 0

0 2

0 0

2 2

Done

53

Please, enter coordinates of Trapezoid

28 29 17 16 28 88 29 21

Done

7

Square: (0, 1) (1, 0) (0, 0) (1, 1)

Square: (1, 0) (0, 1) (0, 0) (1, 1)

Rectangle: (2, 0) (0, 2) (0, 0) (2, 2)

Trapezoid: (28, 29) (17, 16) (28, 88) (29, 21)

0

Process finished with exit code 0