

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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