

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

ЛАБОРАТОРНАЯ РАБОТА №2
по курсу объектно-ориентированное программирование I семестр, 2021/22
уч. год

Студент *Шандрюк Пётр Николаевич, группа М8О-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

Условие

Задание: Вариант 25: Треугольник, очередь. Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру, согласно вариантам задания. Классы должны удовлетворять следующим правилам:

1. Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
2. Классы фигур должны содержать набор следующих методов:
 - Перегруженный оператор ввода координат вершин фигуры из потока `std::istream` («»). Он должен заменить конструктор, принимающий координаты вершин из стандартного потока.
 - Перегруженный оператор вывода в поток `std::ostream` («»), заменяющий метод `Print` из лабораторной работы 1.
 - Оператор копирования (`=`)
 - Оператор сравнения с такими же фигурами (`==`)
3. Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке).
4. Класс-контейнер должен содержать набор следующих методов:
 - `push` - вставка в очередь
 - `pop` - удаление из очереди
 - `top` - первый элемент в очереди
 - `empty` - пустая ли очередь
 - `size` - количество элементов в очереди
 - Перегруженный оператор вывода в поток `std::ostream` («»)

Описание программы

Исходный код лежит в 9 файлах:

1. `main.cpp`: основная программа, взаимодействие с пользователем посредством команд из меню
2. `tqueueitem.h`: описание класса предмета очереди
3. `point.h`: описание класса точки
4. `tqueue.h`: описание класса очереди

5. triangle.h: описание класса треугольника, наследующегося от figures
6. point.cpp: реализация класса точки
7. tqueue.cpp: реализация класса очереди
8. triangle.cpp: реализация класса треугольника
9. tqueueitem.cpp: реализация класса предмета очереди

Дневник отладки

Недочёты

Выводы

Я научился создавать простые динамические структуры данных, а также работать с объектами, передаваемыми "по значению".

Исходный код

tqueue.h

```
#ifndef INC_2_LAB_TQUEUE_H
#define INC_2_LAB_TQUEUE_H

#include "tqueueitem.h"

class TQueue {
public:
    TQueue();

    TQueue(const TQueue &other);

    ~TQueue();

    friend ostream &operator<<(ostream &os, const TQueue &q);

    bool push(Triangle &&Triangle);

    bool pop();

    Triangle top();

    bool empty();

    size_t size();

    void Clear();
private:
    TQueue_item *first;
    TQueue_item *last;
    size_t n;
};

#endif // INC_2_LAB_TQUEUE_H
```

tqueue.cpp

```
//  
// Created by Temi4 on 13.09.2021.  
//  
  
#include "tqueue.h"  
  
TQueue::TQueue() : first(nullptr), last(nullptr), n(0) {}  
  
TQueue::TQueue(const TQueue &other){  
    n = 0;  
    auto a = other.first;  
    for (int i = 0; i < other.n; ++i){  
        this->push(std::move(Triangle(a->GetTriangle())));  
        a = a->GetNext();  
    }  
}  
  
ostream &operator<<(ostream &os, const TQueue &q) {  
    os.precision(3);  
    TQueue_item *item = q.first;  
    auto s = new double[q.n];  
    for (int i = 0; i < q.n; ++i){  
        s[i] = item->GetTriangle().Area();  
        item = item->GetNext();  
    }  
    for (int i = (int) q.n - 1; i >= 0; --i){  
        os << "> " << s[i] << " ";  
    }  
    return os;  
}  
  
bool TQueue::push(Triangle &&triangle) {  
    auto *tail = new TQueue_item(triangle);  
    if (tail == nullptr) {  
        return false;  
    }  
    if (this->empty()) { // если пустая очередь, то голова и хвост - один и тот же  
                        // элемент  
        this->first = this->last = tail;  
    } else if (n == 1) { // хвост - вставляемый элемент, а также следующий элемент
```

```

        // от первого
        last = tail;
        first->SetNext(tail);
    } else {
        this->last->SetNext(tail); // хвост - следующий элемент от последнего
        last = tail;
    }
    n++;
    return true;
}

bool TQueue::pop() {
    if (first) {
        first = first->GetNext();
        return true;
    }
    return false;
}

Triangle TQueue::top() {
    if (first) {
        return first->GetTriangle();
    }
}

size_t TQueue::size() { return n; }

bool TQueue::empty() { return first == nullptr; }

TQueue::~TQueue() { delete first; }

void TQueue::Clear(){
    delete first;
    first = last = nullptr;
    n = 0;
}

```

tqueueitem.h

```
#ifndef INC_2_LAB_QUQUE_ITEM_H
#define INC_2_LAB_QUQUE_ITEM_H

#include "triangle.h"

class TQueue_item {
public:
    TQueue_item(const Triangle &triangle);

    TQueue_item(const TQueue_item &other);

    TQueue_item *
    SetNext(TQueue_item *next_); // присваивает значение следующему элементу

    TQueue_item *GetNext(); // возвращает указатель на следующий элемент

    Triangle GetTriangle();

    friend ostream &operator<<(ostream &os, const TQueue_item &obj);

    virtual ~TQueue_item();

private:
    Triangle triangle;
    TQueue_item *next;
};

#endif // INC_2_LAB_QUQUE_ITEM_H
```

tqueueitem.cpp

```
#include "tqueueitem.h"

TQueue_item::TQueue_item(const Triangle &triangle) {
    this->triangle = triangle;
    this->next = nullptr;
    cout << "Queue item: created" << endl;
}

TQueue_item::TQueue_item(const TQueue_item &other) {
    this->triangle = other.triangle;
    this->next = other.next;
    cout << "Stack item: copied" << endl;
}

TQueue_item *TQueue_item::SetNext(TQueue_item *next_) {
    TQueue_item *prev = this->next;
    this->next = next_;
    return prev;
}

Triangle TQueue_item::GetTriangle() { return this->triangle; }

TQueue_item *TQueue_item::GetNext() { return this->next; }

TQueue_item::~TQueue_item() {
    cout << "Queue item: deleted" << endl;
    delete next;
}

ostream &operator<<(ostream &os,
                    const TQueue_item &obj) { // перегруженный оператор вывода
    os << "{";
    os << obj.triangle;
    os << "}" << endl;
    return os;
}
```


point.h

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point {
public:
    Point();

    Point(double x, double y);

    Point &operator++();

    friend Point operator+(const Point &left, const Point &right);

    double dist(Point &other);

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

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

    friend class Triangle; // Дружественные классы, чтобы были доступны
                        // координаты точки

private:
    double y_;
    double x_;
};

#endif // POINT_H
```

point.cpp

```
#include "point.h"
```

```
#include <cmath>
```

```
Point::Point() : x_(0.0), y_(0.0) {} // стандартный конструктор
```

```
Point::Point(double x, double y) : x_(x), y_(y) {} // конструктор через значения
```

```
double Point::dist(Point &other) { // расстояние между двумя точками
    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) { // перегруженный оператор >>
    is >> p.x_ >> p.y_;
    return is;
}
```

```
std::ostream &operator<<(std::ostream &os,
                        const Point &p) { // перегруженный оператор <<
    os << "(" << p.x_ << ", " << p.y_ << ")";
    return os;
}
```

```
Point &Point::operator++() {
    this->x_ += 1;
    this->y_ += 1;
    return *this;
}
```

```
Point operator+(const Point &left, const Point &right) {
    return Point(left.x_ + right.x_, left.y_ + right.y_);
}
```

triangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

#include "point.h"
#include <vector>

using namespace std;

class Triangle {
public:
    Triangle();

    Triangle(vector<Point>);

    Triangle(const Triangle &other);

    virtual ~Triangle();

    friend istream &operator>>(istream &is,
                               Triangle &obj); // перегруженный оператор >>

    friend ostream &operator<<(ostream &os, const Triangle &obj);

    Triangle &operator++();

    friend Triangle operator+(const Triangle &left, const Triangle &right);

    Triangle &operator=(const Triangle &right);

    size_t VertexNumbers();

    double Area();

private:
    Point a, b, c; // a - левая нижняя вершина, b - левая верхняя, c - правая
                  // верхняя, d - правая нижняя
};

#endif
```

triangle.cpp

```
#include "triangle.h"
```

```
#include <cmath>
```

```
double Triangle::Area() {  
    double first = sqrt(pow(a.x_ - b.x_, 2) + pow(a.y_ - b.y_, 2));  
    double second = sqrt(pow(b.x_ - c.x_, 2) + pow(b.y_ - c.y_, 2));  
    double third = sqrt(pow(a.x_ - c.x_, 2) + pow(a.y_ - c.y_, 2));  
    double p = (first + second + third) / 2;  
    return sqrt(p * (p - first) * (p - second) * (p - third));  
}
```

```
Triangle::Triangle() : a(), b(), c(){  
    cout << "Default triangle is created" << endl;  
}
```

```
Triangle::Triangle(const Triangle &other) {  
    a = other.a;  
    b = other.b;  
    c = other.c;  
    cout << "Made copy of triangle" << endl;  
}
```

```
Triangle::Triangle(vector<Point> v) : a(v[0]), b(v[1]), c(v[2]) {  
    cout << "Triangle with vertices " << a << ", " << b << ", " << c << " was created" << endl;  
}
```

```
istream &operator>>(istream &is, Triangle &obj) {  
    cout << "Enter cords" << endl;  
    is >> obj.a >> obj.b >> obj.c;  
    return is;  
}
```

```
ostream &operator<<(ostream &os, const Triangle &obj) {  
    os << "Rectangle: " << obj.a << ", " << obj.b << ", " << obj.c;  
    return os;  
}
```

```
Triangle &Triangle::operator++() { // инкрементируем каждую вершину  
    ++this->a;  
    ++this->b;  
    ++this->c;  
}
```

```

    return *this;
}

Triangle operator+(const Triangle &left, const Triangle &right) {
    vector<Point> v{left.a + right.a, left.b + right.b, left.c + right.c};
    return Triangle(v);
}

Triangle &Triangle::operator=(const Triangle &other) {
    if (this == &other) {
        return *this;
    }
    this->a = other.a;
    this->b = other.b;
    this->c = other.c;
    return *this;
}

Triangle::~Triangle() { cout << "Triangle was deleted" << endl; }

size_t Triangle::VertexNumbers() { return 3; }

```

main.cpp

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

using namespace std;

int main() {
    auto *q = new TQueue;
    vector<Point> v{Point(0, 0), Point(0, 1), Point(1, 1)};
    /*Triangle r(v);
    ++r;
    Triangle r1;
    cin >> r1;
    Triangle r2 = r1 + r;
    TQueue_item q1(r1);
    cout << q1 << endl; */
    cout << Triangle(v).Area() << endl;
    q->push(Triangle(v));
    q->push(Triangle());
    cout << *q << endl;
    q->Clear();
    cout << q->size() << endl;
    return 0;
}
```