МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №3

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 (колонка фигура 1), согласно вариантам задания. Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы №1;
- Требования к классу контейнера аналогичны требованиям из лабораторной работы №2;
- Класс-контейнер должен содержать объекты используя std::shared _ ptr<...>

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

Дневник отладки

Недочёты

Выводы

Я научился использовать "умные"указатели и применять их в построении программировании классов.

Исходный код

tqueue.h

```
#ifndef LAB_3_TQUEUE_H
#define LAB_3_TQUEUE_H

#include "tqueueitem.h"

class TQueue {
public:
    TQueue();
    TQueue(const TQueue &other) = default;
    ~TQueue();

    friend ostream &operator<<(ostream &os, const TQueue &q);
    bool push(shared_ptr<Triangle> &&triangle);
    bool pop();
    shared_ptr<Triangle> top();
    bool empty();
    size_t size();

private:
    shared_ptr<TQueue_item> first;
    shared_ptr<TQueue_item> last;
    size_t n;
};


#endif //LAB_3_TQUEUE_H
```

tqueue.cpp

```cpp
#include "tqueue.h"

TQueue::TQueue() : first(nullptr), last(nullptr), n(0) {}

ostream &operator<<(ostream &os, const TQueue &q) {
  shared_ptr<TQueue_item> item = q.first;
  while (item) {
    os << *item;
    item = item->GetNext();
  }
  return os;
}

bool TQueue::push(shared_ptr<Triangle> &&triangle) {
  shared_ptr<TQueue_item> tail =
      make_shared<TQueue_item>(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;
}
```

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

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

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

TQueue::~TQueue() {}
```

tqueueitem.h

```
#ifndef LAB_3_QUEUE_ITEM_H
#define LAB_3_QUEUE_ITEM_H

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

class TQueue_item {
public:
    TQueue_item(const shared_ptr<Triangle> &triangle);
    TQueue_item(const shared_ptr<TQueue_item> &other);
    shared_ptr<TQueue_item> SetNext(shared_ptr <TQueue_item> &next_);
    shared_ptr<TQueue_item> GetNext();
    shared_ptr<Triangle> GetTriangle();
    friend ostream &operator<<(ostream &os, const TQueue_item &obj);
    virtual ~TQueue_item();

private:
    shared_ptr <Triangle> triangle;
    shared_ptr <TQueue_item> next;
};


#endif //LAB_3_QUEUE_ITEM_H
```

tqueueitem.cpp

```cpp
#include "tqueueitem.h"

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

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

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

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

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

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

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

```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 TRIANGLE_H
#define TRIANGLE_H

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

using namespace std;

class Triangle {
public:
    Triangle();
    Triangle(vector <Point> v);
    Triangle(const Triangle& other);
    virtual ~Triangle();
    friend istream &operator>>(istream &is, Triangle &obj);
    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;
    Point b;
    Point c;
};

#endif //TRIANGLE_H
```

triangle.cpp

```cpp
#include "triangle.h"


double Triangle::Area() { return (abs(a.x_ - b.x_) * abs(a.y_ - b.y_)); }

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" <<
}

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 << "Triangle: " << 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 <iostream>
#include "tqueue.h"

using namespace std;

int main() {

    TQueue queue;
    vector <Point> v {Point(0, 1), Point(0,0), Point(1,1)};
    queue.push(make_shared <Triangle>(Triangle(v)));
    queue.push(make_shared <Triangle>(Triangle()));
    cout << queue;

    return 0;
}
```