МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №4

по курсу объектно-ориентированное программирование I семестр, 2021/22 уч. год

Студент *Ядров Артем Леонидович, группа М8О-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

Условие

Задание: Вариант 27: Прямоугольник, 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 соджержать 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. rectangle.h: описание класса прямоугольника, наследующегося от figures
6. point.cpp: реализация класса точки
7. tqueue.cpp: реализация класса очереди
8. rectangle.cpp: реализация класса прямоугольника
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

```
//
// Created by Temi4 on 13.09.2021.
//

#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(Rectangle &&rectangle);

  bool pop();

  Rectangle 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(Rectangle(a->GetRectangle())));
    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->GetRectangle().Area();
    item = item->GetNext();
  }
  for (int i = (int) q.n - 1; i >= 0; --i){
    os << "=> " << s[i] << " ";
  }
  return os;
}

bool TQueue::push(Rectangle &&rectangle) {
  auto *tail = new TQueue_item(rectangle);
  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;
}

Rectangle TQueue::top() {
  if (first) {
    return first->GetRectangle();
  }
}

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

```
//
// Created by Temi4 on 13.09.2021.
//

#ifndef INC_2_LAB_QUQUE_ITEM_H
#define INC_2_LAB_QUQUE_ITEM_H

#include "rectangle.h"

class TQueue_item {
public:
  TQueue_item(const Rectangle &rectangle);

  TQueue_item(const TQueue_item &other);

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

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

  Rectangle GetRectangle();

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

  virtual ~TQueue_item();

private:
  Rectangle rectangle;
  TQueue_item *next;
};

#endif // INC_2_LAB_QUQUE_ITEM_H
```

tqueueitem.cpp

```cpp
//
// Created by Temi4 on 13.09.2021.
//

#include "tqueueitem.h"

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

TQueue_item::TQueue_item(const TQueue_item &other) {
  this->rectangle = other.rectangle;
  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;
}

Rectangle TQueue_item::GetRectangle() { return this->rectangle; }

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.rectangle;
  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 Rectangle; // 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_);
}
```

rectangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H

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

using namespace std;

class Rectangle {
public:
  Rectangle();

  Rectangle(vector<Point>);

  Rectangle(const Rectangle &other);

  virtual ~Rectangle();

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

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

  Rectangle &operator++();

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

  Rectangle &operator=(const Rectangle &right);

  size_t VertexNumbers();

  double Area();

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

#endif
```

rectangle.cpp

```cpp
#include "rectangle.h"

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

Rectangle::Rectangle() : a(), b(), c(), d() {
  cout << "Default rectangle is created" << endl;
}

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

Rectangle::Rectangle(vector<Point> v) : a(v[0]), b(v[1]), c(v[2]), d(v[3]) {
  cout << "Rectangle with vertices " << a << ", " << b << ", " << c << ", " << d
       << " was created" << endl;
}

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

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

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

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

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

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

size_t Rectangle::VertexNumbers() { return 4; }
```

main.cpp

```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), Point(1, 0)};
  /*Rectangle r(v);
  ++r;
  Rectangle r1;
  cin >> r1;
  Rectangle r2 = r1 + r;
  TQueue_item q1(r1);
  cout << q1 << endl; */
  q->push(Rectangle(v));
  q->push(Rectangle());
  cout << *q << endl;
  q->Clear();
  cout << q->size() << endl;
  return 0;
}
```