МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №2

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

Студент Хашимов Амир Азизович группа М8О-207Б-20

Преподаватель Дорохов Евгений Павлович

### **Цель работы**

Целью лабораторной работы является:

· Закрепление навыков работы с классами.

· Создание простых динамических структур данных.

· Работа с объектами, передаваемыми «по значению».

### **Задание**

Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий **одну фигуру (Rectangle),** согласно вариантам задания. Классы должны удовлетворять следующим правилам:

* Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
* 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 (==)
* Класс-контейнер должен соджержать объекты фигур “по значению” (не по ссылке).

Нельзя использовать:

· Стандартные контейнеры std.

· Шаблоны (template).

· Различные варианты умных указателей (shared\_ptr, weak\_ptr).

Программа должна позволять:

· Вводить произвольное количество фигур и добавлять их в контейнер.

· Распечатывать содержимое контейнера.

· Удалять фигуры из контейнера.

Описание программы

* Структура Point (point.h) с double полями x и y, а также методами ToString(), DistanceTo(Point endPoint) и перегрузкой оператора присваивания
* Родительский класс Figure (figure.h + figure.cpp) с реализованными виртуальными публичными методами VertexesNumber() [возвращает apixes.size()], Print(std::ostream& os) [выводит с помощью Point.ToString()], чисто виртуальным методом Area() и protected полями std::string figureName и std::vector<Point>apixes
  + Класс-наследник Rectangle (rectangle.h и rectangle.cpp), переопределяющий метод Area() [возвращает произведение двух смежных сторон] (в конструкторе кроме ввода определяется figureName = “Rectangle”). Также переопределен конструктор копирования и операторы == и <<
* Класс QueueNode (queueNode.h и queueNode.cpp), содержащий указатели head, tail, а также значение Data
* Класс Queue (queue.h и queue.cpp) с реализацией основных методов очереди

Исходный код:

// figure.cpp

#include "figure.h"

size\_t Figure::VertexesNumber()

{

return apixes.size();

}

void Figure::Print(std::ostream& os)

{

std::string output = figureName + ": ";

for (auto& apix : apixes)

{

output += apix.ToString();

}

os << output + "**\n**";

}

// figure.h

#pragma once

#include <ostream>

#include "point.h"

#include <vector>

class Figure

{

protected:

std::vector<Point> apixes;

std::string figureName;

public:

virtual size\_t VertexesNumber();

virtual double Area() = 0;

virtual void Print(std::ostream& os);

};

//point.h

#pragma once

#ifndef POINT\_H\_INCLUDED

#define POINT\_H\_INCLUDED

#include <string>

#include <cmath>

struct Point {

double x, y;

Point(double x, double y)

{

this->x = x;

this->y = y;

}

Point(const Point& other)

{

this->x = other.x;

this->y = other.y;

}

std::string ToString()

{

return "(" + std::to\_string(x) + ", " + std::to\_string(y) + ")";

}

double DistanceTo(Point endPoint)

{

return sqrt((x - endPoint.x) \* (x - endPoint.x)

+ (y - endPoint.y) \* (y - endPoint.y));

}

};

#endif

//rectangle.cpp

#include "rectangle.h"

#include "point.h"

#include <iostream>

Rectangle::Rectangle(const Rectangle& other)

{

figureName = "Rectangle";

for (int i = 0; i < 4; i++)

{

apixes.push\_back(other.apixes[i]);

}

}

double Rectangle::Area()

{

return apixes[0].DistanceTo(apixes[1])

\* apixes[1].DistanceTo(apixes[2]);

}

Rectangle::Rectangle(std::istream& inputStream)

{

figureName = "Rectangle";

for (int i = 0; i < 4; i++)

{

Point inputPoint(0, 0);

inputStream >> inputPoint.x >> inputPoint.y;

apixes.push\_back(inputPoint);

}

}

bool Rectangle::operator==(const Rectangle& other) const

{

for (int i = 0; i < 4; i++)

{

if (apixes[i].x != other.apixes[i].x && apixes[i].y != other.apixes[i].y)

return false;

}

return true;

}

std::ostream& operator<<(std::ostream& os, const Rectangle& rect)

{

((Figure&)rect).Print(os);

return os;

}

//rectangle.h

#pragma once

#ifndef RECTANGLE\_H\_INCLUDED

#define RECTANGLE\_H\_INCLUDED

#include "figure.h"

class Rectangle :

public Figure

{

public:

friend std::ostream& operator<<(std::ostream& os, const Rectangle& rect);

Rectangle(const Rectangle&);

double Area() override;

Rectangle(std::istream&);

bool operator==(const Rectangle& other) const;

friend std::ostream& operator<<(std::ostream& os, const Rectangle& rect);

};

#endif

// queueNode.cpp

#include "queueNode.h"

QueueNode::QueueNode(item& value)

{

prev = NULL;

next = NULL;

Data = &value;

}

QueueNode::QueueNode()

{

prev = NULL;

next = NULL;

Data = NULL;

}

// queueNode.h

#pragma once

#ifndef QUEUE\_NODE\_H\_INCLUDED

#define QUEUE\_NODE\_H\_INCLUDED

#include "rectangle.h"

typedef Rectangle item;

class QueueNode

{

public:

QueueNode(item&);

QueueNode();

item\* Data;

QueueNode\* prev;

QueueNode\* next;

};

#endif

// queue.h

#pragma once

#ifndef QUEUE\_H\_INCLUDED

#define QUEUE\_H\_INCLUDED

#include "queueNode.h"

class Queue

{

private:

size\_t \_size;

public:

Queue();

Queue(const Queue&);

~Queue();

size\_t size();

bool empty();

item\* top();

void push(item&);

item\* pop();

friend std::ostream& operator<<(std::ostream& os, const Queue& dt);

QueueNode\* head;

QueueNode\* tail;

};

#endif

// queue.cpp

#include "queue.h"

#include <iostream>

Queue::Queue()

{

head = new QueueNode();

tail = new QueueNode();

head->next = tail;

tail->prev = head;

\_size = 0;

}

Queue::Queue(const Queue& other): Queue()

{

QueueNode\* current = other.head;

while (current->next != other.tail)

{

current = current->next;

push(\*current->Data);

}

}

Queue::~Queue()

{

std::cout << "I\'m Destructor\n";

}

size\_t Queue::size()

{

return \_size;

}

bool Queue::empty()

{

return \_size == 0;

}

item\* Queue::top()

{

if (empty())

throw;

return head->next->Data;

}

void Queue::push(item& value)

{

QueueNode\* prelast = tail->prev;

QueueNode\* additional = new QueueNode(value);

tail->prev = additional;

prelast->next = additional;

additional->next = tail;

additional->prev = prelast;

\_size++;

}

item\* Queue::pop()

{

if (empty())

throw;

QueueNode\* deleted = head->next;

head->next = deleted->next;

if (deleted->next != NULL)

deleted->next->prev = head;

\_size--;

return deleted->Data;

}

std::ostream& operator<<(std::ostream& os, const Queue& dt)

{

QueueNode\* current = dt.head;

while (current->next != dt.tail)

{

current = current->next;

current->Data->Print(os);

}

return os;

}