Цель работы

Целью лабораторной работы является:

- [?] Закрепление навыков работы с классами;

- [?] Знакомство с умными указателями.

Задание

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 No1;

- [?] Требования к классу контейнера аналогичны требованиям из лабораторной работы No2;

- [?] Класс-контейнер должен содержать объекты используя std::shared_ptr<...>.

Нельзя использовать:

- [?] Стандартные контейнеры std;

- [?] Шаблоны (template);

- [?] Объекты «по-значению».

Программа должна позволять:

- [?] Вводить произвольное количество фигур и добавлять их в контейнер;

- [?] Распечатывать содержимое контейнера;

- Удалять фигуры из контейнера.

Листинг

Figure.h

```
//
// Created by Илья Рожков on 12.09.2021.
//

#ifndef LAB1_FIGURE_H
#define LAB1_FIGURE_H
#include "iostream"
#include <utility>
#include <math.h>
#include <cmath>


class Figure {

public:
virtual void Print() const = 0;
virtual size_t VertexesNumber() const = 0;
virtual double Area() const = 0;
```

```
};

#endif //LAB1_FIGURE_H
```

GeronFormula.cpp

```
//
// Created by Илья Рожков on 16.09.2021.
//
#include "GeronFormula.h"
#include<cmath>


double GeronFormula(double a, double b, double c) {
double p, s;
p = (a + b + c) / 2;
s = sqrt(p * (p - a) * (p - b) * (p - c));
return s;
}
```

```
double getDistance(const std::pair<double, double> &x, const std::pair<double,
return sqrt(pow((x.first - y.first), 2) + pow((x.second - y.second), 2));
}

double GeronFormulaFromCordinates(const Cordinate &a, const Cordinate &b,
double x = getDistance(a, b);
double y = getDistance(b, c);
double z = getDistance(c, a);
return GeronFormula(x, y, z);
}

double AreaOfMultigone(const std::vector<Cordinate> &cordinates) {
double s = 0;
for (int i = 0; i < cordinates.size(); i += 3)
s += GeronFormulaFromCordinates(cordinates[i], cordinates[(i + 1) %
return s;
}
```

GeronFormula.h

```
//
// Created by Илья Рожков on 16.09.2021.
//
```

```
#ifndef LAB1_GERONFORMULA_H
#define LAB1_GERONFORMULA_H

#include <utility>
#include <vector>

typedef std::pair<double, double> Cordinate;


double GeronFormula(double a, double b, double c);
double getDistance(const std::pair<double, double>& x , const std::pair<double,
double GeronFormulaFromCordinates(const Cordinate& a, const Cordinate&
double AreaOfMultigone(const std::vector<Cordinate>& cordinates);


#endif //LAB1_GERONFORMULA_H
```

```
//
 // Created by Илья Рожков on 16.09.2021.
 //

 #include "Hexagon.h"

 Hexagon::Hexagon() {
 for (int i = 0; i < 6; i++) {
 Cordinate elemt = std::make_pair(0, 0);
 _cordinates.push_back(elemt);
 }

 }
```

```
Hexagon::Hexagon(const std::vector<Cordinate> &cordinates) :
if (_cordinates.size() != 6) {
throw "wrong size";
}

}

size_t Hexagon::VertexesNumber() const {
return 6;
}

double Hexagon::Area() const {
return AreaOfMultigone(_cordinates);
}

void Hexagon::Print() const {
for (int i = 0; i < _cordinates.size(); i++)
std::cout << _cordinates[i].first << ' ' << _cordinates[i].second << std::endl;

}

std::ostream &operator<<(std::ostream &out, const Hexagon &r) {
for (int i = 0; i < r._cordinates.size(); i++)
out << r._cordinates[i].first << ' ' << r._cordinates[i].second << std::endl;
return out;
}

std::istream &operator>>(std::istream &in, Hexagon &r) {
for (int i = 0; i < 6; i++)
in >> r._cordinates[i].first >> r._cordinates[i].second;
```

```
    return in;
}

Hexagon::~Hexagon() {

}
```

Hexagon.h

```
//
// Created by Илья Рожков on 16.09.2021.
//

#ifndef LAB1_HEXAGON_H
#define LAB1_HEXAGON_H
#include "Figure.h"
#include "GeronFormula.h"



class Hexagon : public Figure {
public:
Hexagon();
~Hexagon();
Hexagon(const std::vector<Cordinate>& cordinates);
size_t VertexesNumber() const override;
double Area() const override;
```

```
    void Print() const override;
    friend std::ostream& operator<<(std::ostream &out, const Hexagon& r);
    friend std::istream& operator>> (std::istream &in, Hexagon& r);


protected:
    std::vector<Cordinate> _cordinates;

};


#endif //LAB1_HEXAGON_H
```

tbinarytree.cpp

```
//
// Created by Илья Рожков on 30.09.2021.
//

#include "tbinarytree.h"
#include "stdexcept"

```

```
TBinaryTree::TBinaryTree() {
t_root = nullptr;
}

void TBinaryTree::Push(const Pentagon& octagon) {
SPTR(TreeElem) curr = t_root;
SPTR(TreeElem) OctSptr(new TreeElem(octagon));

if (!curr)
{
t_root = OctSptr;
}
while (curr)
{
if (curr->get_octagon() == octagon)
{
curr->set_count_fig(curr->get_count_fig() + 1);
return;
}
if (octagon.Area() < curr->get_octagon().Area())
if (curr->get_left() == nullptr)
{
curr->set_left(OctSptr);
return;
}
if (octagon.Area() >= curr->get_octagon().Area())
if (curr->get_right() == nullptr && !(curr->get_octagon() == octagon))
{
curr->set_right(OctSptr);
return;
```

```
}
if (curr->get_octagon().Area() > octagon.Area())
curr = curr->get_left();
else
curr = curr->get_right();
}
}

const Pentagon& TBinaryTree::GetItemNotLess(double area) {
SPTR(TreeElem) curr = t_root;
while (curr)
{
if (area == curr->get_octagon().Area())
return curr->get_octagon();
if (area < curr->get_octagon().Area())
{
curr = curr->get_left();
continue;
}
if (area >= curr->get_octagon().Area())
{
curr = curr->get_right();
continue;
}
}
throw std::out_of_range("out of range");
}

size_t TBinaryTree::Count(const Pentagon& octagon) {
size_t count = 0;
```

```
    SPTR(TreeElem) curr = t_root;

    while (curr)
    {
    if (curr->get_octagon() == octagon)
    count = curr->get_count_fig();
    if (octagon.Area() < curr->get_octagon().Area())
    {
    curr = curr->get_left();
    continue;
    }
    if (octagon.Area() >= curr->get_octagon().Area())
    {
    curr = curr->get_right();
    continue;
    }
    }
    return count;
    }

    void Pop_List(SPTR(TreeElem) curr, SPTR(TreeElem) parent);
    void Pop_Part_of_Branch(SPTR(TreeElem) curr, SPTR(TreeElem) parent);
    void Pop_Root_of_Subtree(SPTR(TreeElem) curr, SPTR(TreeElem) parent);
    void TBinaryTree::Pop(const Pentagon& octagon) {

    SPTR(TreeElem) curr = t_root;
    SPTR(TreeElem) parent = nullptr;

    while (curr && curr->get_octagon() != octagon)
    {
```

```
        parent = curr;
        if (curr->get_octagon().Area() > octagon.Area())
        curr = curr->get_left();
        else
        curr = curr->get_right();
        }

        if (curr == nullptr)
        return;

        curr->set_count_fig(curr->get_count_fig() - 1);

        if(curr->get_count_fig() <= 0)
        {
        if (curr->get_left() == nullptr && curr->get_right() == nullptr)
        {
        Pop_List(curr, parent);
        return;
        }
        if (curr->get_left() == nullptr || curr->get_right() == nullptr)
        {
        Pop_Part_of_Branch(curr, parent);
        return;
        }
        if (curr->get_left() != nullptr && curr->get_right() != nullptr)
        {
        Pop_Root_of_Subtree(curr, parent);
        return;
        }
        }
```

```
}

void Pop_List(SPTR(TreeElem) curr, SPTR(TreeElem) parent) {
if (parent->get_left() == curr)
parent->set_left(nullptr);
else
parent->set_right(nullptr);
}

void Pop_Part_of_Branch(SPTR(TreeElem) curr, SPTR(TreeElem) parent) {
if (parent) {
if (curr->get_left()) {
if (parent->get_left() == curr)
parent->set_left(curr->get_left());

if (parent->get_right() == curr)
parent->set_right(curr->get_left());

curr->set_right(nullptr);
curr->set_left(nullptr);
return;
}

if (curr->get_left() == nullptr) {
if (parent && parent->get_left() == curr)
parent->set_left(curr->get_right());

if (parent && parent->get_right() == curr)
parent->set_right(curr->get_right());
```

```
curr->set_right(nullptr);
curr->set_left(nullptr);
return;
}
}
}

void Pop_Root_of_Subtree(SPTR(TreeElem) curr, SPTR(TreeElem) parent) {
SPTR(TreeElem) replace = curr->get_left();
SPTR(TreeElem) rep_parent = curr;
while (replace->get_right())
{
rep_parent = replace;
replace = replace->get_right();
}

curr->set_octagon(replace->get_octagon());
curr->set_count_fig(replace->get_count_fig());

if (rep_parent->get_left() == replace)
rep_parent->set_left(nullptr);
else
rep_parent->set_right(nullptr);
return;
}

bool TBinaryTree::Empty() {
return t_root == nullptr ? true : false;
}

```

```
void Tree_out (std::ostream& os, SPTR(TreeElem) curr);
std::ostream& operator<<(std::ostream& os, const TBinaryTree& tree) {
SPTR(TreeElem) curr = tree.t_root;
Tree_out(os, curr);
return os;
}

void Tree_out (std::ostream& os, SPTR(TreeElem) curr) {
if (curr)
{
if(curr->get_octagon().Area() >= 0)
os << curr->get_count_fig() << "*" << curr->get_octagon().Area();
if(curr->get_left() || curr->get_right())
{
os << ": [";
if (curr->get_left())
Tree_out(os, curr->get_left());
if(curr->get_left() && curr->get_right())
os << ", ";
if (curr->get_right())
Tree_out(os, curr->get_right());
os << "]";
}
}
}

void recursive_clear(SPTR(TreeElem) curr);
void TBinaryTree::Clear() {
if (t_root->get_left())
recursive_clear(t_root->get_left());
```

```
    t_root->set_left(nullptr);
    if (t_root->get_right())
    recursive_clear(t_root->get_right());
    t_root->set_right(nullptr);
    t_root = nullptr;
    }

    void recursive_clear(SPTR(TreeElem) curr){
    if(curr)
    {
    if (curr->get_left())
    recursive_clear(curr->get_left());
    curr->set_left(nullptr);
    if (curr->get_right())
    recursive_clear(curr->get_right());
    curr->set_right(nullptr);
    }
    }

    TBinaryTree::~TBinaryTree() {
    }
```

Node.cpp

```
#include "Node.h"
```

```
#include <memory>


TreeElem::TreeElem() {
octi = nullptr;
count_fig = 0;
t_left = nullptr;
t_right = nullptr;
}

TreeElem::TreeElem(const Pentagon octagon) {
octi = MakeSPTR(Pentagon)(octagon);
count_fig = 1;
t_left = nullptr;
t_right = nullptr;
}

const Pentagon& TreeElem::get_octagon() const{
return *octi;
}
int TreeElem::get_count_fig() const{
return count_fig;
}
SPTR(TreeElem) TreeElem::get_left() const{
return t_left;
}
SPTR(TreeElem) TreeElem::get_right() const{
return t_right;
}
```

```
void TreeElem::set_octagon(const Pentagon& octagon){
octi = MakeSPTR(Pentagon)(octagon);
}
void TreeElem::set_count_fig(const int count) {
count_fig = count;
}
void TreeElem::set_left(SPTR(TreeElem) to_left) {
t_left = to_left;
}
void TreeElem::set_right(SPTR(TreeElem) to_right) {
t_right = to_right;
}

TreeElem::~TreeElem() {
}
```