

# Лабораторная работа №4

## Цель работы

Целью лабораторной работы является:

- Знакомство с шаблонами классов;
- Построение шаблонов динамических структур данных.

## Задание

Необходимо спроектировать и запрограммировать на языке C++ **шаблон класса-контейнера** первого уровня, содержащий **одну фигуру (колонка фигура 1)**, согласно вариантам задания. Классы должны удовлетворять следующим правилам:

- Требования к классам фигуры аналогичны требованиям из лабораторной работы No1;
- Требования к классу контейнера аналогичны требованиям из лабораторной работы No2;
- Шаблон класса-контейнера должен содержать объекты используя `std::shared_ptr<...>`.  
Нельзя использовать:
- Стандартные контейнеры `std`.

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

# G e r o n F o r m u l a. c p p

//	
	// 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 GeronFormulaFromCoordinates(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> &coordinates) {
double s = 0;
for (int i = 0; i < coordinates.size(); i += 3)
s += GeronFormulaFromCoordinates(coordinates[i], coordinates[(i + 1) %
return s;
}
```

**G  
er  
o  
n  
F  
or  
m  
ul  
a.  
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);
	_coordinates.push_back(elemt);
	}
	}

```

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

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

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

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

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

std::istream &operator>>(std::istream &in, Hexagon &r) {
for (int i = 0; i < 6; i++)
in >> r._coordinates[i].first >> r._coordinates[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<Coordinate> _coordinates;
	};
	#endif //LAB1_HEXAGON_H

**tb  
in  
ar  
yt  
re  
e.  
c  
p  
p**

//	
	// Created by Илья Рожков on 30.09.2021.
//	
	#include "tbinarytree.h"
	#include "stdexcept"



```
template <class Poligon>
```

```
TBinaryTree<Poligon>::TBinaryTree() {
```

```
    t_root = nullptr;
```

```
}
```

```
template <class Poligon>
```

```
void TBinaryTree<Poligon>::Push(const Poligon& octagon) {
```

```
    SPTR(TreeElem<Poligon>) curr = t_root;
```

```
    SPTR(TreeElem<Poligon>) OctSptr(new TreeElem<Poligon>(octagon));
```

```
    if (!curr)
```

```
    {
```

```
        t_root = OctSptr;
```

```
    }
```

```
    while (curr)
```

```
    {
```

```
        if (curr->get_poligon() == octagon)
```

```
        {
```

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

```
            return;
```

```
        }
```

```
        if (octagon.Area() < curr->get_poligon().Area())
```

```
        if (curr->get_left() == nullptr)
```

```
        {
```

```
            curr->set_left(OctSptr);
```

```
            return;
```

```
        }
```

```
        if (octagon.Area() >= curr->get_poligon().Area())
```

```
        if (curr->get_right() == nullptr && !(curr->get_poligon() == octagon))
```

```
        {
```

```

curr->set_right(OctSptr);

return;

}

if (curr->get_poligon().Area() > octagon.Area())

curr = curr->get_left();

else

curr = curr->get_right();

}

}

```

```

template <class Poligon>

const Poligon& TBinaryTree<Poligon>::GetItemNotLess(double area) {

SPTR(TreeElem<Poligon>) curr = t_root;

while (curr)

{

if (area == curr->get_poligon().Area())

return curr->get_poligon();

if (area < curr->get_poligon().Area())

{

curr = curr->get_left();

continue;

}

if (area >= curr->get_poligon().Area())

{

curr = curr->get_right();

continue;

}

}

throw std::out_of_range("out of range");

}

```

```
template <class Poligon>
```

```
size_t TBinaryTree<Poligon>::Count(const Poligon& octagon) {
```

```
size_t count = 0;
```

```
SPTR(TreeElem<Poligon>) curr = t_root;
```

```
while (curr)
```

```
{
```

```
if (curr->get_poligon() == octagon)
```

```
count = curr->get_count_fig();
```

```
if (octagon.Area() < curr->get_poligon().Area())
```

```
{
```

```
curr = curr->get_left();
```

```
continue;
```

```
}
```

```
if (octagon.Area() >= curr->get_poligon().Area())
```

```
{
```

```
curr = curr->get_right();
```

```
continue;
```

```
}
```

```
}
```

```
return count;
```

```
}
```

```
template <class Poligon>
```

```
void Pop_List(SPTR(TreeElem<Poligon>) curr, SPTR(TreeElem<Poligon>)
```

```
parent->get_left() == curr)
```

```
parent->set_left(nullptr);
```

```
else
```

```
parent->set_right(nullptr);
```

```

    }

template <class Poligon>
void Pop_Part_of_Branch(SPTR(TreeElem<Poligon>) curr,
    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;
        }
    }
}

template <class Poligon>

```

```

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

curr->set_poligon(replace->get_poligon());
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;
}

```

```

template <class Poligon>
void TBinaryTree<Poligon>::Pop(const Poligon& octagon) {
    SPTR(TreeElem<Poligon>) curr = t_root;
    SPTR(TreeElem<Poligon>) parent = nullptr;

    while (curr && curr->get_poligon() != octagon)
    {
        parent = curr;
        if (curr->get_poligon().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;
```

```
}
```

```
}
```

```
}
```

```
template <class Poligon>
```

```
bool TBinaryTree<Poligon>::Empty() {
```

```
return t_root == nullptr ? true : false;
```

```
}
```

```
template <class Poligon>
```

```
void Tree_out (std::ostream& os, SPTR(TreeElem<Poligon>) curr) {
```

```
if (curr)
```

```
{
```

```
if(curr->get_poligon().Area() >= 0)
```

```
os << curr->get_count_fig() << "*" << curr->get_poligon().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 << "]";
```

```
}
```

```
}
```

```
}
```

```
template <class A>
```

```
std::ostream& operator<<(std::ostream& os, const TBinaryTree<A>& tree) {
```

```
SPTR(TreeElem<A>) curr = tree.t_root;
```

```
Tree_out(os, curr);
```

```
return os;
```

```
}
```

```
template <class Poligon>
```

```

void recursive_clear(SPTR(TreeElem<Poligon>) 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);
    }
}

template <class Poligon>
void TBinaryTree<Poligon>::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;
}

template <class Poligon>
TBinaryTree<Poligon>::~TBinaryTree() {
}

#include "pentagon.h"

template class TBinaryTree<Pentagon>;

template std::ostream& operator<<(std::ostream& os, const

```



# Н о д е. с р р

//	
	// Created by Илья Рожков on 30.09.2021.
	//
	#include "Node.h"
	#include <memory>
	template <class Poligon>
	TreeElem<Poligon>::TreeElem() {
	polig = nullptr;
	count_fig = 0;
	t_left = nullptr;
	t_right = nullptr;
	}
	template <class Poligon>
	TreeElem<Poligon>::TreeElem(const Poligon poligon) {
	polig = MakeSPTR(Poligon)(poligon);
	count_fig = 1;
	t_left = nullptr;
	t_right = nullptr;
	}

	template <class Poligon>
	const Poligon& TreeElem<Poligon>::get_poligon() const{
	return *polig;
	}
	template <class Poligon>
	int TreeElem<Poligon>::get_count_fig() const{
	return count_fig;
	}
	template <class Poligon>
	SPTR(TreeElem<Poligon>) TreeElem<Poligon>::get_left() const{
	return t_left;
	}
	template <class Poligon>
	SPTR(TreeElem<Poligon>) TreeElem<Poligon>::get_right() const{
	return t_right;
	}
	template <class Poligon>
	void TreeElem<Poligon>::set_poligon(const Poligon& poligon){
	polig = MakeSPTR(Poligon)(poligon);
	}
	template <class Poligon>
	void TreeElem<Poligon>::set_count_fig(const int count) {
	count_fig = count;
	}
	template <class Poligon>
	void TreeElem<Poligon>::set_left(SPTR(TreeElem<Poligon>) to_left) {
	t_left = to_left;
	}
	template <class Poligon>

	void TreeElem<Poligon>::set_right(SPTR(TreeElem<Poligon>) to_right) {
	t_right = to_right;
	}
	template <class Poligon>
	TreeElem<Poligon>::~~TreeElem() {
	}
	#include "Pentagon.h"
	template class TreeElem<Pentagon>;