## **Лабораторная работа №4**

**Студент**: Сазонов Вадим Кириллович , м8О-201Б-20

### **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** (одна из фигур ЛР3 на выбор). Вариант структуры данных для контейнера выбрать из документа “**Варианты структур данных**” **(контейнер 1-го уровня)** согласно своему номеру из **Варианты ЛР4.**.

Классы должны удовлетворять следующим правилам:

· Требования к классу фигуры аналогичны требованиям из лабораторной работы 3

· Классы фигур должны иметь переопределенный оператор вывода в поток std::ostream (<<). Оператор должен распечатывать параметры фигуры (тип фигуры, длины сторон, радиус и т.д).

· Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream (>>). Оператор должен вводить основные параметры фигуры (длины сторон, радиус и т.д).

· Классы фигур должны иметь операторы копирования (=).

· Классы фигур должны иметь операторы сравнения с такими же фигурами (==).

· Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке).

· Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.

· Класс-контейнер должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).

· Класс-контейнер должен иметь метод по удалению фигуры из контейнера (определяется структурой контейнера).

· Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<).

· Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.

· Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

**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.

Вариант 16:

Бинарное- Дерево

**Код программы:**

**Main.cpp**

#include <iostream>

#include "tree.h"

#include "fiveanglefig.h"

#define SIZE 3

int main()

{

Fiveanglefigure a[SIZE];

TBinaryTree tree;

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

{

std::cin >> a[i];

tree.Push(a[i]);

}

std::cout << tree << std::endl;

std::cout << tree.Count(a[0]) << std::endl;

tree.Pop(a[1]);

std::cout << tree << std::endl;

tree.Clear();

if (tree.Empty())

std::cout << "Дерево пустое" << std::endl;

return 0;

}

**Fiveanglefig.h**

#pragma once

#include "figure.h"

#include <iostream>

class Fiveanglefigure : public Figure

{

public:

Fiveanglefigure(float a, float b, float length):x1(a),y1(b),side(length){};

Fiveanglefigure():x1(0),y1(0),side(0){};

float centerx();

float centery();

float area() const;

void print();

friend bool operator==(const Fiveanglefigure& fig\_1,const Fiveanglefigure& fig\_2);

friend bool operator!=(const Fiveanglefigure& fig\_1,const Fiveanglefigure& fig\_2);

friend std::istream& operator>> (std::istream &in, Fiveanglefigure &fig){

std::cout<<"Введите сторону:";

in >> fig.side;

std::cout<<"Введите координаты нижнего угла:";

in >> fig.x1 >>fig.y1;

return in;

}

private:

float x1, y1;

float side;

};

**fiveanglefig.cpp**

#include "fiveanglefig.h"

#include <math.h>

#include <iostream>

float Fiveanglefigure::centerx()

{

return x1 + side / 2;

}

float Fiveanglefigure::centery()

{

return y1 + side;

}

float Fiveanglefigure::area() const

{

return 5 \* side \* side / 4 / tan(36 \* 3.14 / 180);

}

void Fiveanglefigure::print()

{

std::cout << x1 << " " << y1 << std::endl;

}

bool operator==(const Fiveanglefigure& fig\_1,const Fiveanglefigure& fig\_2){

if (fig\_1.side == fig\_2.side && fig\_1.x1 ==fig\_2.x1 && fig\_1.y1 == fig\_2.y1) return true;

return false;

}

bool operator!=(const Fiveanglefigure& fig\_1,const Fiveanglefigure& fig\_2){

if (fig\_1.side == fig\_2.side && fig\_1.x1 ==fig\_2.x1 && fig\_1.y1 == fig\_2.y1) return false;

return true;

}

**Figure.h**

#pragma once

class Figure

{

public:

virtual float centerx() = 0;

virtual float centery() = 0;

virtual float area() const = 0;

virtual void print() = 0;

};

**Tree.h**

#ifndef TBINARYTREE\_H

#define TBINARYTREE\_H

#include <iostream>

#include "treeelem.h"

#include "fiveanglefig.h"

class TBinaryTree {

public:

TBinaryTree();

void Push(const Fiveanglefigure& octagon);

size\_t Count(const Fiveanglefigure& octagon);

void Pop(const Fiveanglefigure& octagon);

bool Empty();

friend std::ostream& operator<<(std::ostream& os, const TBinaryTree& tree);

void Clear();

virtual ~TBinaryTree();

private:

TreeElem\* t\_root;

};

#endif

**Tree.cpp**

#include "tree.h"

#include <stdexcept>

TBinaryTree::TBinaryTree() {

t\_root = nullptr;

}

void TBinaryTree::Push(const Fiveanglefigure& fig) {

TreeElem\* curr = t\_root;

if (curr == nullptr)

t\_root = new TreeElem(fig);

while (curr)

{

if (curr->get\_fig() == fig)

{

curr->set\_count\_fig(curr->get\_count\_fig() + 1);

return;

}

if (fig.area() < curr->get\_fig().area())

if (curr->get\_left() == nullptr)

{

curr->set\_left(new TreeElem(fig));

return;

}

if (fig.area() >= curr->get\_fig().area())

if (curr->get\_right() == nullptr && !(curr->get\_fig() == fig))

{

curr->set\_right(new TreeElem(fig));

return;

}

if (curr->get\_fig().area() > fig.area())

curr = curr->get\_left();

else

curr = curr->get\_right();

}

}

size\_t TBinaryTree::Count(const Fiveanglefigure& octagon) {

size\_t count = 0;

TreeElem\* curr = t\_root;

while (curr)

{

if (curr->get\_fig() == octagon)

count = curr->get\_count\_fig();

if (octagon.area() < curr->get\_fig().area())

{

curr = curr->get\_left();

continue;

}

if (octagon.area() >= curr->get\_fig().area())

{

curr = curr->get\_right();

continue;

}

}

return count;

}

void Pop\_List(TreeElem\* curr, TreeElem\* parent);

void Pop\_Part\_of\_Branch(TreeElem\* curr, TreeElem\* parent);

void Pop\_Root\_of\_Subtree(TreeElem\* curr, TreeElem\* parent);

void TBinaryTree::Pop(const Fiveanglefigure& octagon) {

TreeElem\* curr = t\_root;

TreeElem\* parent = nullptr;

while (curr && curr->get\_fig() != octagon)

{

parent = curr;

if (curr->get\_fig().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(TreeElem\* curr, TreeElem\* parent) {

if (parent->get\_left() == curr)

parent->set\_left(nullptr);

else

parent->set\_right(nullptr);

delete(curr);

}

void Pop\_Part\_of\_Branch(TreeElem\* curr, 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);

delete(curr);

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);

delete(curr);

return;

}

}

}

void Pop\_Root\_of\_Subtree(TreeElem\* curr, TreeElem\* parent) {

TreeElem\* replace = curr->get\_left();

TreeElem\* rep\_parent = curr;

while (replace->get\_right())

{

rep\_parent = replace;

replace = replace->get\_right();

}

curr->set\_fig(replace->get\_fig());

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);

delete(replace);

return;

}

bool TBinaryTree::Empty() {

return t\_root == nullptr ? true : false;

}

void Tree\_out (std::ostream& os, TreeElem\* curr);

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

TreeElem\* curr = tree.t\_root;

Tree\_out(os, curr);

return os;

}

void Tree\_out (std::ostream& os, TreeElem\* curr) {

if (curr)

{

if(curr->get\_fig().area() >= 0)

os << curr->get\_count\_fig() << "\*" << curr->get\_fig().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(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);

delete t\_root;

t\_root = nullptr;

}

void recursive\_clear(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);

delete curr;

}

}

TBinaryTree::~TBinaryTree() {

}

**Treeelem.h**

#ifndef TREEELEM\_H

#define TREEELEM\_H

#include <memory>

#include "fiveanglefig.h"

class TreeElem{

public:

TreeElem();

TreeElem(const Fiveanglefigure octagon);

const Fiveanglefigure& get\_fig() const;

int get\_count\_fig() const;

TreeElem\* get\_left() const;

TreeElem\* get\_right() const;

void set\_fig(const Fiveanglefigure& octagon);

void set\_count\_fig(const int count);

void set\_left(TreeElem\* to\_left);

void set\_right(TreeElem\* to\_right);

virtual ~TreeElem();

private:

Fiveanglefigure fig;

int count\_fig;

TreeElem\* t\_left;

TreeElem\* t\_right;

};

#endif

**Treeelem.cpp**

#include <iostream>

#include <memory>

#include "treeelem.h"

TreeElem::TreeElem() {

fig;

count\_fig = 0;

t\_left = nullptr;

t\_right = nullptr;

}

TreeElem::TreeElem(const Fiveanglefigure figure) {

fig = figure;

count\_fig = 1;

t\_left = nullptr;

t\_right = nullptr;

}

const Fiveanglefigure& TreeElem::get\_fig() const{

return fig;

}

int TreeElem::get\_count\_fig() const{

return count\_fig;

}

TreeElem\* TreeElem::get\_left() const{

return t\_left;

}

TreeElem\* TreeElem::get\_right() const{

return t\_right;

}

void TreeElem::set\_fig(const Fiveanglefigure& figure){

fig = figure;

}

void TreeElem::set\_count\_fig(const int count) {

count\_fig = count;

}

void TreeElem::set\_left(TreeElem\* to\_left) {

t\_left = to\_left;

}

void TreeElem::set\_right(TreeElem\* to\_right) {

t\_right = to\_right;

}

TreeElem::~TreeElem() {

}

**Результат работы программы:**

**Введите сторону:1**

**Введите координаты нижнего угла:2 2**

**Введите сторону:3**

**Введите координаты нижнего угла:1 1**

**Введите сторону:4**

**Введите координаты нижнего угла:5 5**

**1\*1.72163: [1\*15.4947: [1\*27.5461]]**

**1**

**1\*1.72163: [1\*27.5461]**

**Дерево пустое**