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 *Михеева Кристина Олеговна, группа М8О-207Б-20*
Преподаватель *Дорохов Евгений Павлович*

Условие:

Вариант 17: Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня (TBinaryTree), содержащий **треугольник** класса фигуры, согласно вариантам задания. Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Требования к классу контейнера аналогичны требованиям из лабораторной работы 2.
- Класс-контейнер должен содержать объекты используя std:shared_ptr<…>.
- Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

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 фигуру из контейнера.

Описание программы:

Исходный код лежит в 10 файлах:

1. main.cpp: основная программа, взаимодействие с пользователем посредством команд из меню.
2. figure.h: описание абстрактного класса фигуры.
3. point.h: описание класса точки.
4. point.cpp: реализация класса точки.
5. triangle.h: описание класса треугольника, наследующего от figure.
6. triangle.cpp: реализация класса треугольника, наследующего от figure.
7. TbinaryTree.cpp: реализация контейнера (бинарное дерево).
8. TBinaryTree.h: описание контейнера (бинарное дерево).

9. TbinaryTreeItem.cpp: реализация элемента бинарного дерева.

10.iTbinaryTreeItem.h: описание элемента бинарного дерева.

Дневник отладки:

В данной лабораторной возникли проблемы с утечкой памяти, но при дальнейшем выполнении работы, все утечки были устранены.

Выводы:

В данной лабораторной мы снова закрепили навыки с работами классами, а также познакомились с умными указателями, которые являются очень полезными, так как они предназначены для управления динамически выделенной памятью и обеспечения освобождения (удаления) выделенной памяти.

Исходный код:

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H
#include <memory>
#include "point.h"

class Figure {
    public:
    virtual size_t VertexesNumber() = 0;
    virtual double Area() = 0;
    virtual void Print (std:: ostream &os) = 0;
    virtual ~Figure() {};
};

#endif
```

point.h

```
   #ifndef POINT_H
   #define POINT_H

   #include <iostream>

   class Point {
   public:
     Point();
     Point(std::istream &is);
     Point(double x, double y);
     friend bool operator == (Point& p1, Point& p2);
```

```
  friend class Triangle;
  double getX();
  double getY();
  friend std::istream& operator>>(std::istream& is, Point& p);
  friend std::ostream& operator<<(std::ostream& os, Point& p);

private:
  double x;
  double y;
};

#endif
```

point.cpp

```
#include "point.h"

#include <cmath>

Point::Point() : x(0.0), y(0.0) {}

Point::Point(double x, double y) : x(x), y(y) {}

Point::Point(std::istream &is) {
  is >> x >> y;
}


double Point::getX() {
  return x;
};
double Point::getY() {
  return y;
};

std::istream& operator>>(std::istream& is, Point& p) {
  is >> p.x >> p.y;
  return is;
}

std::ostream& operator<<(std::ostream& os, Point& p) {
  os << "(" << p.x << ", " << p.y << ")";
  return os;
}

bool operator == (Point &p1, Point& p2) {
  return (p1.x == p2.x && p1.y == p2.y);
}
```

triangle.h

```
#include "figure.h"
#include <iostream>
```

```
#ifndef TRIANGLE_H
#define TRIANGLE_H
class Triangle : public Figure {
    public:
    Triangle(std:: istream &is);
    Triangle();
    size_t VertexesNumber();
    double Area();
    double GetArea();
    void Print (std:: ostream &os);
    virtual ~Triangle();
    friend bool operator == (Triangle& t1, Triangle& t2);
    friend std::ostream& operator << (std::ostream& os, Triangle& t);
    double area;

    private:
    Point a;
    Point b;
    Point c;
};
#endif
```

triangle.cpp

```
#include "triangle.h"
#include <cmath>
Triangle::Triangle() {}

Triangle::Triangle(std:: istream &is)
{
    is >> a >> b >> c;
    std:: cout << "The triangle was created" << std:: endl;
}

size_t Triangle::VertexesNumber()
{
    return 3;
}

double Triangle::Area() {
    double Square = 0.5 * abs(a.getX() * b.getY() + b.getX() * c.getY() +
c.getX() * a.getY() - a.getY() * b.getX() - b.getY() * c.getX() - c.getY() *
a.getX());
    this->area = Square;
    return Square;
}

void Triangle::Print(std:: ostream &os)
{
    std:: cout << "Triangle: " << a << " " << b << " " << c << std:: endl;
```

```
}

Triangle::~Triangle() {
    std:: cout << "Trianle was deleted" << std:: endl;
}

double Triangle:: GetArea() {
       return area;
   }

bool operator == (Triangle& t1, Triangle& t2){
        if(t1.a == t2.a && t1.b == t2.b && t1.c == t2.c){
            return true;
        }
        return false;
    }

std::ostream& operator << (std::ostream& os, Triangle& t){
    os << "Triangle: ";
    os << t.a << t.b << t.c;
    os << std::endl;
    return os;
}
```

TBinaryTree.h

```
#ifndef TBINARYTREE_H
#define TBINARYTREE_H
#include "TBinaryTreeItem.h"

class TBinaryTree {
public:
TBinaryTree();
TBinaryTree(const TBinaryTree &other);
void Push(Triangle &triangle);
std::shared_ptr<TBinaryTreeItem> Pop(std::shared_ptr<TBinaryTreeItem> root,
Triangle &triangle);
Triangle& GetItemNotLess(double area, std::shared_ptr<TBinaryTreeItem> root);
void Clear();
bool Empty();
int Count(double minArea, double maxArea);
friend std::ostream& operator<<(std::ostream& os, TBinaryTree& tree);
virtual ~TBinaryTree();
std::shared_ptr<TBinaryTreeItem> root;
```

```
};
#endif
```

TBinaryTree.cpp

```
#include "TBinaryTree.h"

TBinaryTree::TBinaryTree () {
    root = NULL;
}

std::shared_ptr<TBinaryTreeItem> copy (std::shared_ptr<TBinaryTreeItem> root)
{
    if (!root) {
        return NULL;
    }
    std::shared_ptr<TBinaryTreeItem> root_copy(new TBinaryTreeItem
(root->GetTriangle()));
    root_copy->SetLeft(copy(root->GetLeft()));
    root_copy->SetRight(copy(root->GetRight()));
    return root_copy;
}

TBinaryTree::TBinaryTree (const TBinaryTree &other) {
    root = copy(other.root);
}

void Print (std::ostream& os, std::shared_ptr<TBinaryTreeItem> node){
    if (!node){
        return;
    }
    if(node->GetLeft()){
        os << node->GetTriangle().GetArea() << ": [";
        Print (os, node->GetLeft());
        if (node->GetRight()){
            if (node->GetRight()){
                os << ", ";
                Print (os, node->GetRight());
            }
        }
        os << "]";
    } else if (node->GetRight()) {
       os << node->GetTriangle().GetArea() << ": [";
        Print (os, node->GetRight());
        if (node->GetLeft()){
            if (node->GetLeft()){
                os << ", ";
                Print (os, node->GetLeft());
            }
        }
        os << "]";
    }
    else {
        os << node->GetTriangle().GetArea();
```

```
    }
}

std::ostream& operator<< (std::ostream& os, TBinaryTree& tree){
    Print(os, tree.root);
    os << "\n";
}

void TBinaryTree::Push (Triangle &triangle) {
    if (root == NULL) {
    std::shared_ptr<TBinaryTreeItem> help(new TBinaryTreeItem(triangle));
    root = help;
    }
    else if (root->GetTriangle() == triangle) {
        root->IncreaseCounter();
    }
    else {
        std::shared_ptr<TBinaryTreeItem> parent = root;
        std::shared_ptr<TBinaryTreeItem> current;
        bool childInLeft = true;
        if (triangle.GetArea() < parent->GetTriangle().GetArea()) {
            current = root->GetLeft();
        }
        else if (triangle.GetArea() > parent->GetTriangle().GetArea()) {
            current = root->GetRight();
            childInLeft = false;
        }
        while (current != NULL) {
            if (current->GetTriangle() == triangle) {
                current->IncreaseCounter();
            }
            else {
            if (triangle.GetArea() < current->GetTriangle().GetArea()) {
                parent = current;
                current = parent->GetLeft();
                childInLeft = true;
                }
            else if (triangle.GetArea() > current->GetTriangle().GetArea()) {
                parent = current;
                current = parent->GetRight();
                childInLeft = false;
            }
        }
    }
        std::shared_ptr <TBinaryTreeItem> item (new
TBinaryTreeItem(triangle));
        current = item;
        if (childInLeft == true) {
            parent->SetLeft(current);
        }
        else {
            parent->SetRight(current);
        }
    }
```

```
}

 std::shared_ptr<TBinaryTreeItem> FMRST (std::shared_ptr<TBinaryTreeItem>
root) {
    if (root->GetLeft() == NULL) {
        return root;
    }
    return FMRST(root->GetLeft());
}

std::shared_ptr <TBinaryTreeItem> TBinaryTree:: Pop(std::shared_ptr
<TBinaryTreeItem> root, Triangle &triangle) {
    if (root == NULL) {
        return root;
    }
    else if (triangle.GetArea() < root->GetTriangle().GetArea()) {
        root->SetLeft(Pop(root->GetLeft(), triangle));
    }
    else if (triangle.GetArea() > root->GetTriangle().GetArea()) {
        root->SetRight(Pop(root->GetRight(), triangle));
    }
    else {
        //first case of deleting - we are deleting a list
        if (root->GetLeft() == NULL && root->GetRight() == NULL) {
            root = NULL;
            return root;
        }
        //second case of deleting - we are deleting a verex with only one
child
        else if (root->GetLeft() == NULL && root->GetRight() != NULL) {
            std::shared_ptr <TBinaryTreeItem> pointer = root;
            root = root->GetRight();
            return root;
        }
        else if (root->GetRight() == NULL && root->GetLeft() != NULL) {
            std::shared_ptr <TBinaryTreeItem> pointer = root;
            root = root->GetLeft();
            return root;
        }
        //third case of deleting
        else {
            std::shared_ptr <TBinaryTreeItem> pointer =
FMRST(root->GetRight());
            root->GetTriangle().area = pointer->GetTriangle().GetArea();
            root->SetRight(Pop(root->GetRight(), pointer->GetTriangle()));
        }
    }
}

void RecursiveCount(double minArea, double maxArea, std::shared_ptr
<TBinaryTreeItem> current, int& ans) {
    if (current != NULL) {
        RecursiveCount(minArea, maxArea, current->GetLeft(), ans);
        RecursiveCount(minArea, maxArea, current->GetRight(), ans);
```

```
        if (minArea <= current->GetTriangle().GetArea() &&
current->GetTriangle().GetArea() < maxArea) {
            ans += current->ReturnCounter();
        }
    }
}

int TBinaryTree::Count(double minArea, double maxArea) {
    int ans = 0;
    RecursiveCount(minArea, maxArea, root, ans);
    return ans;
}

Triangle& TBinaryTree::GetItemNotLess(double area, std::shared_ptr
<TBinaryTreeItem> root) {
    if (root->GetTriangle().GetArea() >= area) {
        return root->GetTriangle();
    }
    else {
        GetItemNotLess(area, root->GetRight());
    }
}

void RecursiveClear(std::shared_ptr <TBinaryTreeItem> current){
    if (current!= NULL){
        RecursiveClear(current->GetLeft());
        RecursiveClear(current->GetRight());
            current = NULL;
    }
}

void TBinaryTree::Clear(){
    RecursiveClear(root);
    root = NULL;
}

bool TBinaryTree::Empty() {
     if (root == NULL) {
         return true;
     }
     return false;
}

TBinaryTree::~TBinaryTree() {
    Clear();
    std:: cout << "Your tree has been deleted" << std:: endl;
}
```

TBinaryTreeItem.h

```
#ifndef TBINARYTREE_ITEM_H
#define TBINARYTREE_ITEM_H
#include "triangle.h"

class TBinaryTreeItem {
public:
TBinaryTreeItem(const Triangle& triangle);
TBinaryTreeItem(const TBinaryTreeItem& other);
Triangle& GetTriangle();
void SetTriangle(Triangle& triangle);
std::shared_ptr<TBinaryTreeItem> GetLeft();
std::shared_ptr<TBinaryTreeItem> GetRight();
void SetLeft(std::shared_ptr<TBinaryTreeItem> item);
void SetRight(std::shared_ptr<TBinaryTreeItem> item);
void SetTriangle(const Triangle& triangle);
void IncreaseCounter();
void DecreaseCounter();
int ReturnCounter();
virtual ~TBinaryTreeItem();

private:
Triangle triangle;
std::shared_ptr<TBinaryTreeItem> left;
std::shared_ptr<TBinaryTreeItem> right;
int counter;
};
#endif
```

TBinaryTreeItem.cpp

```
#include "TBinaryTreeItem.h"


TBinaryTreeItem::TBinaryTreeItem(const Triangle &triangle) {
    this->triangle = triangle;
    this->left = this->right = NULL;
    this->counter = 1;
}

TBinaryTreeItem::TBinaryTreeItem(const TBinaryTreeItem &other) {
    this->triangle = other.triangle;
    this->left = other.left;
    this->right = other.right;
    this->counter = other.counter;
}

Triangle& TBinaryTreeItem::GetTriangle() {
    return this->triangle;
}

void TBinaryTreeItem::SetTriangle(const Triangle& triangle){
    this->triangle = triangle;
}
std::shared_ptr<TBinaryTreeItem> TBinaryTreeItem::GetLeft(){
    return this->left;
}
std::shared_ptr<TBinaryTreeItem> TBinaryTreeItem::GetRight(){
    return this->right;
}


void TBinaryTreeItem::SetLeft(std::shared_ptr<TBinaryTreeItem> item) {
    if (this != NULL){
        this->left = item;
    }
}

void TBinaryTreeItem::SetRight(std::shared_ptr<TBinaryTreeItem> item) {
    if (this != NULL){
        this->right = item;
    }
}

void TBinaryTreeItem::IncreaseCounter() {
    if (this != NULL){
        counter++;
    }
}
void TBinaryTreeItem::DecreaseCounter() {
    if (this != NULL){
        counter--;
    }
}
```

```
int TBinaryTreeItem::ReturnCounter() {
    return this->counter;
}

TBinaryTreeItem::~TBinaryTreeItem() {
}
```

main.cpp

```
#include "triangle.h"
#include <iostream>
#include "TBinaryTree.h"
#include "TBinaryTreeItem.h"

int main () {
    Triangle a (std:: cin);
    std:: cout << "Area of a triangle:" << " " << a.Area() << std:: endl;

    Triangle b (std:: cin);
    std:: cout << "Area of a triangle:" << " " << b.Area() << std:: endl;

    Triangle c (std:: cin);
    std:: cout << "Area of a triangle:" << " " << c.Area() << std:: endl;



    TBinaryTree tree;
    std:: cout << "Is tree empty? " << tree.Empty() << std:: endl;
    tree.Push(a);
    std:: cout << "And now, is tree empty? " << tree.Empty() << std:: endl;
    tree.Push(b);
    tree.Push(c);
    std:: cout << "The number of figures with area in [minArea, maxArea] is: "
<< tree.Count(0, 100000) << std:: endl;
    std:: cout << "The result of searching the same-figure-counter is: " <<
tree.root->ReturnCounter() << std:: endl;
    std:: cout << "The result of function named GetItemNotLess is: " <<
tree.GetItemNotLess(0, tree.root) << std:: endl;
    std:: cout << tree << std:: endl;
    tree.root = tree.Pop(tree.root, a);
    std:: cout << tree << std:: endl;
    system("pause");
}
```