МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

# ЛАБОРАТОРНАЯ РАБОТА №5 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++ 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.

Нельзя использовать:

- Стандартные контейнеры std.
- Шаблоны (template).
- Объекты «по-значению»

## 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, наследующегося от figure.
6. triangle.cpp: реализация класса треугольника, наследующегося 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 шаблонам. Главная идея — создание функций без указания точного типов некоторых или всех переменных. Для этого мы определяем функцию, указывая тип параметра шаблона, который используется вместо любого типа данных. После того, как мы создали функцию с типом параметра шаблона, мы фактически создали «трафарет функции».

## Листинг:

**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 X();
  double Y();
  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::X() {
  return x;
};
double Point::Y() {
  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**

```
#ifndef TRIANGLE_H
#define TRIANGLE_H

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


class Triangle : public Figure {
    public:
    Triangle(std::istream &InputStream);
    Triangle();
    double GetArea();
    size_t VertexesNumber();
    double Area();
    void Print(std::ostream &OutputStream);
    friend bool operator == (Triangle& p1, Triangle& p2);
    friend std::ostream& operator << (std::ostream& os, Triangle& p);
    virtual ~Triangle();
    double area;

    private:
    Point a;
    Point b;
    Point c;

};
#endif
```

**triangle.cpp**

```
#include "triangle.h"
#include <cmath>

    Triangle::Triangle() {}

    Triangle::Triangle(std::istream &InputStream)
  {
      InputStream >> a;
      InputStream >> b;
      InputStream >> c;
      std:: cout << "Triangle that you wanted to create has been created" <<
std:: endl;
  }

  void Triangle::Print(std::ostream &OutputStream) {
      OutputStream << "Triangle: ";
```

```
      OutputStream << a << " " << b << " " << c << std:: endl;

  }

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

   double Triangle::Area() {
    double s = 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 = s;
   return s;
   }

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

    Triangle::~Triangle() {
          std:: cout << "My friend, your triangle has been deleted" << std::
endl;
      }

    bool operator == (Triangle& p1, Triangle& p2){
        if(p1.a == p2.a && p1.b == p2.b && p1.c == p2.c) {
            return true;
        }
        return false;
    }

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

**TBinaryTree.h**

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

template <class T>
```

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

**TBinaryTree.cpp**

```
#include "TBinaryTreeItem.h"

template <class T>
TBinaryTreeItem<T>::TBinaryTreeItem(const T &triangle) {
    this->triangle = triangle;
    this->left = this->right = NULL;
    this->counter = 1;
}

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

template <class T>
T& TBinaryTreeItem<T>::GetTriangle() {
    return this->triangle;
}

template <class T>
void TBinaryTreeItem<T>::SetTriangle(const T& triangle{
    this->triangle = triangle;
}

template <class T>
```

```
std::shared_ptr<TBinaryTreeItem<T>> TBinaryTreeItem<T>::GetLeft(){
    return this->left;
}

template <class T>
std::shared_ptr<TBinaryTreeItem<T>> TBinaryTreeItem<T>::GetRight(){
    return this->right;
}

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

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

template <class T>
void TBinaryTreeItem<T>::IncreaseCounter() {
    if (this != NULL){
        counter++;
    }
}

template <class T>
void TBinaryTreeItem<T>::DecreaseCounter() {
    if (this != NULL){
        counter--;
    }
}

template <class T>
int TBinaryTreeItem<T>::ReturnCounter() {
    return this->counter;
}

template <class T>
TBinaryTreeItem<T>::~TBinaryTreeItem() {
    std::cout << "Destructor TBinaryTreeItem was called\n";
}

template <class T>
std::ostream &operator<<(std::ostream &os, TBinaryTreeItem<T> &obj)
{
    os << "Item: " << obj.GetTriangle() << std::endl;
    return os;
}
```

```
#include "triangle.h"
template class TBinaryTreeItem<Triangle>;
template std::ostream& operator<<(std::ostream& os, TBinaryTreeItem<Triangle>
&obj);
```

**TBinaryTreeItem.h**

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

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

template<class A>
friend std::ostream &operator<<(std::ostream &os, const TBinaryTreeItem<A>
&obj);

private:
T triangle;
std::shared_ptr<TBinaryTreeItem<T>> left;
std::shared_ptr<TBinaryTreeItem<T>> right;
int counter;
```

```
};
#endif
```

**TBinaryTreeItem.cpp**

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

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

template<class A>
friend std::ostream &operator<<(std::ostream &os, const TBinaryTreeItem<A>
&obj);

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

**main.cpp**

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

```
#include "TBinaryTreeItem.h"
int main () {

    Triangle a (std::cin);
    std:: cout << "The area of your figure is : " << a.Area() << std:: endl;

    Triangle b (std::cin);
    std:: cout << "The area of your figure is : " << b.Area() << std:: endl;

    Triangle c (std::cin);
    std:: cout << "The area of your figure is : " << c.Area() << std:: endl;

    //lab4
    TBinaryTree<Triangle> tree;
    std:: cout << "Is tree empty? " << tree.Empty() << std:: endl;
    std:: cout << "And now, is tree empty? " << tree.Empty() << std:: endl;
    tree.Push(a);
    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;
    return 0;

}
```

4