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

ЛАБОРАТОРНАЯ РАБОТА №4

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: Недосекин Максим Александрович, группа М8О-207Б-20

Преподаватель: Дорохов Евгений Павлович, каф. 806

Задание:

Спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий одну фигуру. Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 2.
- Классы фигур должны содержать набор следующих методов:
  - Перегруженный оператор ввода координат вершин фигуры из потока `std::istream` (>>)
  - Перегруженный оператор вывода в поток `std::ostream` (<<)
  - Оператор копирования (=)
  - Оператор сравнения с такими же фигурами (==)
- Класс-контейнер должен содержать объекты фигур “по значению” (не по ссылке).
- Класс-контейнер должен содержать набор следующих методов:
  - `Length()` – возвращает количество элементов в контейнере
  - `Empty()` – для пустого контейнера возвращает 1, иначе – 0
  - `First()` – возвращает первый (левый) элемент списка
  - `Last()` – возвращает последний (правый) элемент списка
  - `InsertFirst(elem)` – добавляет элемент в начало списка
  - `RemoveFirst()` – удаляет элемент из начала списка
  - `InsertLast(elem)` – добавляет элемент в конец списка
  - `RemoveLast()` – удаляет элемент из конца списка
  - `Insert(elem, pos)` – вставляет элемент на позицию `pos`
  - `Remove(pos)` – удаляет элемент, находящийся на позиции `pos`
  - `Clear()` – удаляет все элементы из списка
  - `operator<<` – выводит список поэлементно в поток вывода (слева направо)

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 №18:

- Фигура: Трапеция (Trapezoid)
- Контейнер: Бинарное дерево (Binary Tree)

Описание программы:

Исходный код разделён на 10 файлов:

- `figure.h` – описание класса фигуры
- `point.h` – описание класса точки
- `point.cpp` – реализация класса точки
- `trapezoid.h` – описание класса трапеции
- `trapezoid.cpp` – реализация класса трапеции
- `TBinaryTreeItem.h` – описание элемента бинарного дерева
- `TBinaryTreeItem.cpp` – реализация элемента бинарного дерева
- `TBinaryTree.h` – описание бинарного дерева
- `TBinaryTree.cpp` – реализация бинарного дерева
- `main.cpp` – основная программа

Дневник отладки:

В отладке не нуждался

Вывод:

Я познакомился на практике с реализацией пользовательских контейнеров, а также закрепил полученные ранее навыки работы с классами. Была написана программа с собственной реализацией контейнера в виде бинарного дерева с содержанием объектов фигур «по назначению», которая может вводить/выводить содержимое контейнера и удалить их при надобности.

Исходный код:

figure.h:

```
#ifndef FIGURE_H
#define FIGURE_H

#include "point.h"

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

#endif
```

Main.cpp:

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


int main () {
   Trapezoid a (std::cin);
   std:: cout << "The area of your figure is : " << a.Area() << std:: endl;

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

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


   TBinaryTree 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;
}
```

Point.h:

```
#ifndef POINT_H
#define POINT_H

#include <iostream>

class Point {
public:
  Point();
  Point(std::istream &is);
  Point(double x, double y);
  double dist(Point &other);
  friend bool operator == (Point& p1, Point& p2);
  friend class Pentagon;
  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::dist(Point& other) {
    double dx = (other.x - x);
    double dy = (other.y - y);
    return std::sqrt(dx*dx + dy*dy);
}

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);
}
```

TBinaryTree.cpp:

```
#include "TBinaryTree.h"

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

TBinaryTreeItem* copy (TBinaryTreeItem* root) {
    if (!root) {
        return nullptr;
    }
    TBinaryTreeItem* root_copy = new TBinaryTreeItem (root->GetTrapezoid());
    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, TBinaryTreeItem* node){
    if (!node){
        return;
    }
    if(node->GetLeft()){
        os << node->GetTrapezoid().GetArea() << ": [";
        Print (os, node->GetLeft());
        if (node->GetRight()){
            if (node->GetRight()){
                os << ", ";
                Print (os, node->GetRight());
            }
        }
        os << "]";
    } else if (node->GetRight()) {
        os << node->GetTrapezoid().GetArea() << ": [";
        Print (os, node->GetRight());
        if (node->GetLeft()){
            if (node->GetLeft()){
                os << ", ";
```

```
            Print (os, node->GetLeft());
        }
    }
    os << "]";
  }
  else {
    os << node->GetTrapezoid().GetArea();
  }
}

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

void TBinaryTree::Push (Trapezoid &trapezoid) {
  if (root == nullptr) {
  root = new TBinaryTreeItem(trapezoid);
  }
  else if (root->GetTrapezoid() == trapezoid) {
    root->IncreaseCounter();
  }
  else {
    TBinaryTreeItem* parent = root;
    TBinaryTreeItem* current;
    bool childInLeft = true;
    if (trapezoid.GetArea() < parent->GetTrapezoid().GetArea()) {
      current = root->GetLeft();
    }
    else if (trapezoid.GetArea() > parent->GetTrapezoid().GetArea()) {
      current = root->GetRight();
      childInLeft = false;
    }
    while (current != NULL) {
      if (current->GetTrapezoid() == trapezoid) {
        current->IncreaseCounter();
      }
      else {
        if (trapezoid.GetArea() < current->GetTrapezoid().GetArea()) {
        parent = current;
        current = parent->GetLeft();
        childInLeft = true;
        }
        else if (trapezoid.GetArea() > current->GetTrapezoid().GetArea()) {
        parent = current;
        current = parent->GetRight();
        childInLeft = false;
      }
    }
  }
    current = new TBinaryTreeItem(trapezoid);
    if (childInLeft == true) {
      parent->SetLeft(current);
    }
```

```
        else {
            parent->SetRight(current);
        }
    }
}

TBinaryTreeItem* FMRST(TBinaryTreeItem* root) {
    if (root->GetLeft() == nullptr) {
        return root;
    }
    return FMRST(root->GetLeft());
}

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

void RecursiveCount(double minArea, double maxArea, TBinaryTreeItem* current, int& ans) {
    if (current != nullptr) {
```

```
        RecursiveCount(minArea, maxArea, current->GetLeft(), ans);
        RecursiveCount(minArea, maxArea, current->GetRight(), ans);
        if (minArea <= current->GetTrapezoid().GetArea() && current->GetTrapezoid().GetArea() < maxArea)
{
            ans += current->ReturnCounter();
        }
    }
}

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

Trapezoid& TBinaryTree::GetItemNotLess(double area, TBinaryTreeItem* root) {
    if (root->GetTrapezoid().GetArea() >= area) {
        return root->GetTrapezoid();
    }
    else {
        return GetItemNotLess(area, root->GetRight());
    }
}

void RecursiveClear(TBinaryTreeItem* current){
    if (current!= nullptr){
        RecursiveClear(current->GetLeft());
        RecursiveClear(current->GetRight());
            delete current;
            current = nullptr;
    }
}

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

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

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

TBinaryTree.h:

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

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

TBinaryTreeItem.cpp:

```
#include "TBinaryTreeItem.h"


TBinaryTreeItem::TBinaryTreeItem(const Trapezoid &trapezoid) {
    this->trapezoid = trapezoid;
    this->left = this->right = nullptr;
    this->counter = 1;
}

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

Trapezoid& TBinaryTreeItem::GetTrapezoid() {
    return this->trapezoid;
}

void TBinaryTreeItem::SetTrapezoid(const Trapezoid& trapezoid){
    this->trapezoid = trapezoid;
}
TBinaryTreeItem* TBinaryTreeItem::GetLeft(){
    return this->left;
}
TBinaryTreeItem* TBinaryTreeItem::GetRight(){
    return this->right;
}


void TBinaryTreeItem::SetLeft(TBinaryTreeItem* item) {
    if (this != nullptr){
        this->left = item;
    }
}
```

```
void TBinaryTreeItem::SetRight(TBinaryTreeItem* item) {
    if (this != nullptr){
        this->right = item;
    }
}

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

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

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

TBinaryTreeItem.h:

```
#include "TBinaryTreeItem.h"


TBinaryTreeItem::TBinaryTreeItem(const Trapezoid &trapezoid) {
    this->trapezoid = trapezoid;
    this->left = this->right = nullptr;
    this->counter = 1;
}

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

Trapezoid& TBinaryTreeItem::GetTrapezoid() {
    return this->trapezoid;
}

void TBinaryTreeItem::SetTrapezoid(const Trapezoid& trapezoid){
    this->trapezoid = trapezoid;
}
TBinaryTreeItem* TBinaryTreeItem::GetLeft(){
    return this->left;
}
TBinaryTreeItem* TBinaryTreeItem::GetRight(){
    return this->right;
```

```
}


void TBinaryTreeItem::SetLeft(TBinaryTreeItem* item) {
    if (this != nullptr){
        this->left = item;
    }
}

void TBinaryTreeItem::SetRight(TBinaryTreeItem* item) {
    if (this != nullptr){
        this->right = item;
    }
}

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

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

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

Trapezoid.cpp:

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

    Trapezoid::Trapezoid() {}

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

  void Trapezoid::Print(std::ostream &OutputStream) {
      OutputStream << "Trapezoid: ";
      OutputStream << a << " " << b << " " << c << " " << d << std:: endl;

  }
```

```
  size_t Trapezoid::VertexesNumber() {
     size_t number = 4;
     return number;
  }

  double Trapezoid::Area() {
   double k = (a.Y() - d.Y()) / (a.X() - d.X());
   double m = a.Y() - k * a.X();
   double h = abs(b.Y() - k * b.X() - m) / sqrt(1 + k * k);
   return 0.5 * (a.dist(d) + b.dist(c)) * h;
  }

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

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

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

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

Trapezoid.h:

```
#ifndef TRAPEZOID_H
#define TRAPEZOID_H

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


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

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

Результат работы:

```
1 1 2 2 3 3 4 4
Trapezoid that you wanted to create has been created
The area of your figure is : 0
0 0 0 1 1 1 1 0
Trapezoid that you wanted to create has been created
The area of your figure is : 1
0 0 1 1 2 1 3 0
Trapezoid that you wanted to create has been created
The area of your figure is : 2
Is tree empty? 1
And now, is tree empty? 1
The number of figures with area in [minArea, maxArea] is: 3
The result of searching the same-figure-counter is: 1
The result of function named GetItemNotLess is: Trapezoid: (1, 1)(2, 2)(3, 3)(4, 4)

0: [7.90505e-323: [0]]

My friend, your trapezoid has been deleted
7.90505e-323: [0]

My friend, your trapezoid has been deleted
My friend, your trapezoid has been deleted
Your tree has been deleted
My friend, your trapezoid has been deleted
My friend, your trapezoid has been deleted
My friend, your trapezoid has been deleted

Process finished with exit code 0
```