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 *Москвин Артём Артурович, группа М80-208Б-20*

Преподаватель *Дорохов Евгений Павлович*

### 

### 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 ( колонка фигура 1),** согласно вариантам задания. Классы должны удовлетворять следующим правилам:

Требования к классу фигуры аналогичны требованиям из лаб.работы 1.

Классы фигур должны содержать набор следующих методов:  
  
 Перегруженный оператор ввода координат вершин фигуры из потока std::istream (>>). Он должен заменить конструктор, принимающий координаты вершин из стандартного потока.

Перегруженный оператор вывода в поток std::ostream (<<), заменяющий метод Print из лабораторной работы 1.

Оператор копирования (=)

Оператор сравнения с такими же фигурами (==)

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

Класс-контейнер должен содержать набор следующих методов:

TODO: по поводу методов в личку

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.

**Дневник отладки**

Во время выполнения лабораторной работы программа была несколько раз отлажена, так как плохо работала функция удаления из дерева. После нескольких отладок программа стала работать исправно.

**Недочёты**  
Недочётов не было обнаружено.

**Выводы**

Лабораторная работа №4 - это модернизация последних лабораторных 2 семестра. Если на 1 курсе я реализовывал бинарное дерево при помощи структур на языке СИ, то сейчас я реализовал бинарное дерево при помощи ООП на языке С++. Лабораторная прошла успешно, я повторил старый материал и узнал, усвоил много нового.

**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 "hexagon.h"

#include "TBinaryTree.h"

#include "TBinaryTreeItem.h"

// 3.0 4.0 5.0 3.0 6.0 0.0 4.0 -4.0 1.0 -4.0 -2.0 -1.0 ; area = 39.5

// 3.0 0.0 5.0 2.0 4.0 4.0 2.0 5.0 0.0 4.0 0.0 2.0 ; area = 16

// 3.0 0.0 5.0 2.0 4.0 4.0 2.0 5.0 -1.0 3.0 -1.0 1.0; area = 20

int main () {

    //lab1

    Hexagon a (std::cin);

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

    Hexagon b (std::cin);

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

    Hexagon c (std::cin);

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

    //lab2

    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->counter << 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;

}

hexagon.cpp  
  
#include "hexagon.h"

#include <cmath>

    Hexagon::Hexagon() {}

    Hexagon::Hexagon(std::istream &is)

  {

      is >> a;

      is >> b;

      is >> c;

      is >> d;

      is >> e;

      is >> f;

      std:: cout << "Hexagon that you wanted to create has been created" << std:: endl;

  }

  void Hexagon::Print(std::ostream &os) {

      os << "Hexagon: ";

      os << a << " " << b << " " << c << " " << d << " " << e << f << std:: endl;

  }

   size\_t Hexagon::VertexesNumber() {

       size\_t number = 6;

       return number;

   }

   double Hexagon::Area() {

   double q = abs(a.X() \* b.Y() + b.X() \* c.Y() + c.X() \* d.Y() + d.X() \* e.Y() + e.X() \* f.Y() + f.X() \* a.Y()  - b.X() \* a.Y() - c.X() \* b.Y() - d.X() \* c.Y() - e.X() \* d.Y() - f.X() \* e.Y() - a.X() \* f.Y());

   double s = q / 2;

   this->area = s;

   return s;

   }

   double Hexagon:: GetArea() {

       return area;

   }

    Hexagon::~Hexagon() {

          std:: cout << "Hexagon has been deleted" << std:: endl;

      }

    bool operator == (Hexagon& p1, Hexagon& p2){

        if(p1.a == p2.a && p1.b == p2.b && p1.c == p2.c && p1.d == p2.d && p1.e == p2.e && p1.f == p2.f) {

            return true;

        }

        return false;

    }

    std::ostream& operator << (std::ostream& os, Hexagon& p){

    os << "Hexagon: ";

    os << p.a << p.b << p.c << p.d << p.e << p.f;

    os << std::endl;

    return os;

}

hexagon.h  
  
#ifndef HEXAGON\_H

#define HEXAGON\_H

#include "figure.h"

#include <iostream>

class Hexagon : public Figure {

    public:

    Hexagon(std::istream &InputStream);

    Hexagon();

    double GetArea();

    size\_t VertexesNumber();

    double Area();

    void Print(std::ostream &OutputStream);

    friend bool operator == (Hexagon& p1, Hexagon& p2);

    friend std::ostream& operator << (std::ostream& os, Hexagon& p);

    virtual ~Hexagon();

    double area;

    private:

    Point a;

    Point b;

    Point c;

    Point d;

    Point e;

    Point f;

};

#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);

}

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 Hexagon;

  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

TBinaryTree.cpp

#include "TBinaryTree.h"

TBinaryTree::TBinaryTree () {

    root = NULL;

}

TBinaryTreeItem\* copy (TBinaryTreeItem\* root) {

    if (!root) {

        return NULL;

    }

    TBinaryTreeItem\* root\_copy = new TBinaryTreeItem (root->hexagon);

    root\_copy->left = copy (root->left);

    root\_copy->right = copy (root->right);

    return root\_copy;

}

TBinaryTree::TBinaryTree (const TBinaryTree &other) {

    root = copy(other.root);

}

void Print (std::ostream& os, TBinaryTreeItem\* node){

    if (!node){

        return;

    }

    if( node->left){

        os << node->hexagon.GetArea() << ": [";

        Print (os, node->left);

        if (node->right){

            if (node->right){

                os << ", ";

                Print (os, node->right);

            }

        }

        os << "]";

    } else if (node->right) {

       os << node->hexagon.GetArea() << ": [";

        Print (os, node->right);

        if (node->left){

            if (node->left){

                os << ", ";

                Print (os, node->left);

            }

        }

        os << "]";

    }

    else {

        os << node->hexagon.GetArea();

    }

}

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

    Print(os, tree.root);

    os << "\n";

}

void TBinaryTree::Push (Hexagon &hexagon) {

    if (root == NULL) {

    root = new TBinaryTreeItem(hexagon);

    }

    else if (root->hexagon == hexagon) {

        root->counter++;

    }

    else {

        TBinaryTreeItem\* parent = root;

        TBinaryTreeItem\* current;

        bool childInLeft = true;

        if (hexagon.GetArea() < parent->hexagon.GetArea()) {

            current = root->left;

        }

        else if (hexagon.GetArea() > parent->hexagon.GetArea()) {

            current = root->right;

            childInLeft = false;

        }

        while (current != NULL) {

            if (current->hexagon == hexagon) {

                current->counter++;

            }

            else {

            if (hexagon.GetArea() < current->hexagon.GetArea()) {

                parent = current;

                current = parent->left;

                childInLeft = true;

                }

            else if (hexagon.GetArea() > current->hexagon.GetArea()) {

                parent = current;

                current = parent->right;

                childInLeft = false;

            }

        }

    }

        current = new TBinaryTreeItem(hexagon);

        if (childInLeft == true) {

            parent->left = current;

        }

        else {

            parent->right = current;

        }

    }

}

TBinaryTreeItem\* FMRST(TBinaryTreeItem\* root) {

    if (root->left == NULL) {

        return root;

    }

    return FMRST(root->left);

}

TBinaryTreeItem\* TBinaryTree:: Pop(TBinaryTreeItem\* root, Hexagon &hexagon) {

    if (root == NULL) {

        return root;

    }

    else if (hexagon.GetArea() < root->hexagon.GetArea()) {

        root->left = Pop(root->left, hexagon);

    }

    else if (hexagon.GetArea() > root->hexagon.GetArea()) {

        root->right = Pop(root->right, hexagon);

    }

    else {

        //first case of deleting - we are deleting a list

        if (root->left == NULL && root->right == NULL) {

            delete root;

            root = NULL;

            return root;

        }

        //second case of deleting - we are deleting a verex with only one child

        else if (root->left == NULL && root->right != NULL) {

            TBinaryTreeItem\* pointer = root;

            root = root->right;

            delete pointer;

            return root;

        }

        else if (root->right == NULL && root->left != NULL) {

            TBinaryTreeItem\* pointer = root;

            root = root->left;

            delete pointer;

            return root;

        }

        //third case of deleting

        else {

            TBinaryTreeItem\* pointer = FMRST(root->right);

            root->hexagon.area = pointer->hexagon.GetArea();

            root->right = Pop(root->right, pointer->hexagon);

        }

    }

}

void RecursiveCount(double minArea, double maxArea, TBinaryTreeItem\* current, int& ans) {

    if (current != NULL) {

        RecursiveCount(minArea, maxArea, current->left, ans);

        RecursiveCount(minArea, maxArea, current->right, ans);

        if (minArea <= current->hexagon.GetArea() && current->hexagon.GetArea() < maxArea) {

            ans += current->counter;

        }

    }

}

int TBinaryTree::Count(double minArea, double maxArea) {

    int ans = 0;

    RecursiveCount(minArea, maxArea, root, ans);

    return ans;

}

Hexagon& TBinaryTree::GetItemNotLess(double area, TBinaryTreeItem\* root) {

    if (root->hexagon.GetArea() >= area) {

        return root->hexagon;

    }

    else {

        GetItemNotLess(area, root->right);

    }

}

void RecursiveClear(TBinaryTreeItem\* current){

    if (current!= NULL){

        RecursiveClear(current->left);

        RecursiveClear(current->right);

            delete current;

            current = NULL;

    }

}

void TBinaryTree::Clear(){

    RecursiveClear(root);

    delete 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;

}

TBinaryTree.h

#ifndef TBINARYTREE\_H

#define TBINARYTREE\_H

#include "TBinaryTreeItem.h"

class TBinaryTree {

public:

TBinaryTree();

TBinaryTree(const TBinaryTree &other);

void Push(Hexagon &hexagon);

TBinaryTreeItem\* Pop(TBinaryTreeItem\* root, Hexagon &hexagon);

Hexagon& 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 Hexagon &hexagon) {

    this->hexagon = hexagon;

    this->left = this->right = NULL;

    this->counter = 1;

}

TBinaryTreeItem::TBinaryTreeItem(const TBinaryTreeItem &other) {

    this->hexagon = other.hexagon;

    this->left = other.left;

    this->right = other.right;

    this->counter = other.counter;

}

TBinaryTreeItem::~TBinaryTreeItem() {}

TBinaryTreeItem.h  
  
#ifndef TBINARYTREE\_ITEM\_H

#define TBINARYTREE\_ITEM\_H

#include "hexagon.h"

class TBinaryTreeItem {

public:

TBinaryTreeItem(const Hexagon& hexagon);

TBinaryTreeItem(const TBinaryTreeItem& other);

virtual ~TBinaryTreeItem();

Hexagon hexagon;

TBinaryTreeItem \*left;

TBinaryTreeItem \*right;

int counter;

};

#endif