МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ  
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСТИТЕТ)

**ЛАБОРАТОРНАЯ РАБОТА №5**

по курсу “Объектно-ориентированное программирование”

I семестр, 2021/22 учебный год

Студент: *Попов Андрей Викторович, группа М8О-207Б-20*

Преподаватель: *Дорохов Евгений Павлович, каф. 806*

**Задание:** Дополнить класс-контейнер из лабораторной работы №4 умными указателями.

**Вариант №20:**

* + Фигура: Трапеция(Trapezoid)
  + Контейнер: Н-дерево (TNaryTree)

**Описание программы:**

Исходный код разделён на 9 файлов:

1. point.h – описание класса точки
2. point.cpp – реализация класса точки
3. trapezoid.h – описание класса трапеции
4. trapezoid.cpp – реализация класса трапеции
5. TNaryTree\_item.h – описание элемента н-дерева
6. TNaryTree\_item.cpp – реализация элемента н-дерева
7. TNaryTree.h – описание н-дерева
8. TNaryTree.cpp – реализация н-дерева
9. main.cpp – основная программа

**Дневник отладки:**

При замене обычных указателей на умные, ошибок не возникло.

**Вывод:**  
 В процессе выполнения работы я на практике познакомился с умными указателями, изменил реализацию нескольких классов данных(фигуры), и для каждого из них - функции, заменив применение обычных указателей умными.

**Исходный код:**

**figure.h**

**#ifndef FIGURE\_H**

**#define FIGURE\_H**

**#include "point.h"**

**class Figure {**

**public:**

**virtual size\_t VertexesNumber() = 0;**

**virtual double Area() = 0;**

**virtual void Print(std::ostream& os) = 0;**

**~Figure() {**

**};**

**};**

**#endif**

**main.cpp**

**#include <iostream>**

**#include "TNaryTree\_item.h"**

**#include "point.h"**

**#include "trapezoid.h"**

**#include "figure.h"**

**#include "TNaryTree.h"**

**int main(){**

**TNaryTree t(5);**

**t.Update(Trapezoid(Point(0, 0), Point(1, 0),Point(1, 1), Point(0, 1)), "");**

**t.Update(Trapezoid(Point(0, 0), Point(4, 0),Point(4, 1), Point(0, 1)), "b");**

**t.Update(Trapezoid(Point(0, 0), Point(4, 0),Point(4, 1), Point(0, 1)), "bb");**

**t.Update(Trapezoid(Point(0, 0), Point(4, 0),Point(4, 1), Point(0, 1)), "bbc");**

**t.Update(Trapezoid(Point(0, 0), Point(4, 0),Point(4, 1), Point(0, 1)), "c");**

**std::cout << t.size() << "\n";**

**std::cout << t.Area("") << "\n";**

**std::cout << t.size() << "\n";**

**TNaryTree q(t);**

**std::cout << q.size() << " " << q.Area("") << "\n";**

**std::cout << t;**

**t.Clear("");**

**std::cout << t.Area("") << "\n";**

**}**

**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);**

**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\_H**

**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;**

**}**

**trapezoid.h**

**#ifndef TRAPEZOID\_H**

**#define TRAPEZOID\_H**

**#include "figure.h"**

**class Trapezoid: Figure {**

**public:**

**size\_t VertexesNumber();**

**double Area();**

**void Print(std::ostream& os);**

**Trapezoid();**

**Trapezoid(Point a\_, Point b\_, Point c\_, Point d\_);**

**Trapezoid(std::istream& is);**

**private:**

**Point a;**

**Point b;**

**Point c;**

**Point d;**

**};**

**#endif**

**trapezoid.cpp**

**#include <cmath>**

**#include "point.h"**

**#include "trapezoid.h"**

**double Trapezoid::Area(){**

**double la = a.dist(d);**

**double lb = b.dist(c);**

**double lc = c.dist(d);**

**double ld = a.dist(b);**

**if (la > lb) {**

**double t = la;**

**la = lb;**

**lb = t;**

**}**

**double numenator = (lb - la) \* (lb - la) + lc \* lc - ld \* ld;**

**double denumenator = 2 \* (lb - la);**

**if (denumenator == 0) {**

**return (la \* lc);**

**}**

**double h = sqrt(lc \* lc - ((numenator \* numenator) / (denumenator \* denumenator)));**

**return ((la + lb) / 2 \* h);**

**}**

**void Trapezoid::Print(std::ostream& os){**

**os << a << " " << b << " " << c << " " << d << "\n";**

**}**

**size\_t Trapezoid::VertexesNumber(){**

**return (size\_t)(4);**

**}**

**Trapezoid::Trapezoid() : a(Point()), b(Point()), c(Point()), d(Point()){**

**}**

**Trapezoid::Trapezoid(Point a\_, Point b\_, Point c\_, Point d\_):**

**a(a\_), b(b\_), c(c\_), d(d\_){**

**}**

**Trapezoid::Trapezoid(std::istream& is){**

**is >> a >> b >> c >> d;**

**}**

**TNaryTree.h**

**#ifndef TNARYTREE\_H**

**#define TNARYTREE\_H**

**#include "TNaryTree\_item.h"**

**#include "point.h"**

**#include "trapezoid.h"**

**#include "figure.h"**

**class TNaryTree {**

**public:**

**// Инициализация дерева с указанием размера**

**TNaryTree(int n);**

**// Полное копирование дерева**

**TNaryTree(const TNaryTree& other);**

**// Добавление или обновление вершины в дереве согласно заданному пути.**

**// Путь задается строкой вида: "cbccbccc",**

**// где 'c' - старший ребенок, 'b' - младший брат**

**// последний символ строки - вершина, которую нужно добавить или обновить.**

**// Пустой путь "" означает добавление/обновление корня дерева.**

**// Если какой-то вершины в tree\_path не существует,**

**// то функция должна бросить исключение std::invalid\_argument**

**// Если вершину нельзя добавить из за переполнения,**

**// то функция должна бросить исключение std::out\_of\_range**

**void Update(Trapezoid &&polygon, std::string &&tree\_path = "");**

**// Удаление поддерева**

**void Clear(std::string &&tree\_path = "");**

**// Проверка наличия в дереве вершин**

**bool Empty();**

**// Подсчет суммарной площади поддерева**

**double Area(std::string &&tree\_path);**

**int size();**

**// Вывод дерева в формате вложенных списков, где каждый вложенный список является:**

**// "S0: [S1: [S3, S4: [S5, S6]], S2]", где Si - площадь фигуры**

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

**virtual ~TNaryTree();**

**private:**

**int curr\_number;**

**int max\_number;**

**Item\* root;**

**};**

**#endif**

**TNaryTree.cpp**

**#include "TNaryTree.h"**

**#include <string>**

**#include <stdexcept>**

**TNaryTree::TNaryTree(int n) {**

**max\_number = n;**

**curr\_number = 0;**

**root = nullptr;**

**};**

**bool TNaryTree::Empty(){**

**return curr\_number ? 0 : 1;**

**}**

**void TNaryTree::Update(Trapezoid &&polygon, std::string &&tree\_path){**

**if(tree\_path != "" && curr\_number == 0){**

**throw std::invalid\_argument("Error, there is not a root value\n");**

**return;**

**} else if(tree\_path == "" && curr\_number == 0){**

**Item\* q = new Item(polygon);**

**root = q;**

**curr\_number++;**

**} else if(curr\_number + 1 > max\_number){**

**throw std::out\_of\_range("Current number of elements equals maximal number of elements in tree\n");**

**return;**

**} else {**

**Item\* tmp = root;**

**for(int i = 0; i < tree\_path.length() - 1; i++){**

**if(tree\_path[i] == 'b'){**

**if(tmp->bro == nullptr){**

**throw std::invalid\_argument("Path does not exist\n");**

**return;**

**}**

**tmp = tmp->bro;**

**} else if(tree\_path[i] == 'c'){**

**if(tmp->son == nullptr){**

**throw std::invalid\_argument("Path does not exist\n");**

**return;**

**}**

**tmp = tmp->son;**

**} else {**

**throw std::invalid\_argument("Error in path\n");**

**return;**

**}**

**}**

**Item\* item = new Item(polygon);**

**if(tree\_path.back() == 'b'){**

**tmp->bro = item;**

**this->curr\_number++;**

**} else if(tree\_path.back() == 'c'){**

**tmp->son = item;**

**this->curr\_number++;**

**} else {**

**throw std::invalid\_argument("Error in path\n");**

**return;**

**}**

**}**

**}**

**Item\* copy(Item\* root){**

**if(!root){**

**return nullptr;**

**}**

**Item \*root\_copy = new Item(root);**

**root\_copy->bro = copy(root->bro);**

**root\_copy->son = copy(root->son);**

**return root\_copy;**

**}**

**TNaryTree::TNaryTree(const TNaryTree& other){**

**curr\_number = 0;**

**max\_number = other.max\_number;**

**root = copy(other.root);**

**curr\_number = other.curr\_number;**

**;}**

**int TNaryTree::size(){**

**return curr\_number;**

**}**

**int clear(Item\* node) {**

**if (!node) {**

**return 0;**

**}**

**int temp\_res = clear(node->bro) + clear(node->son) + 1;**

**delete node;**

**return temp\_res;**

**}**

**void TNaryTree::Clear(std::string &&tree\_path){**

**Item\* prev\_tmp = nullptr;**

**Item\* tmp = root;**

**if (tree\_path.empty()) {**

**clear(root);**

**curr\_number = 0;**

**root = nullptr;**

**return;**

**}**

**for(int i = 0; i < tree\_path.length(); i++){**

**if(tree\_path[i] == 'b'){**

**if(tmp->bro == nullptr){**

**throw std::invalid\_argument("Path does not exist\n");**

**return;**

**}**

**prev\_tmp = tmp;**

**tmp = tmp->bro;**

**} else if(tree\_path[i] == 'c'){**

**if(tmp->son == nullptr){**

**throw std::invalid\_argument("Path does not exist\n");**

**return;**

**}**

**prev\_tmp = tmp;**

**tmp = tmp->son;**

**} else {**

**throw std::invalid\_argument("Error in path\n");**

**return;**

**}**

**}**

**if (tmp == prev\_tmp->son) {**

**prev\_tmp->son = nullptr;**

**} else {**

**prev\_tmp->bro = nullptr;**

**}**

**curr\_number -= clear(tmp);**

**}**

**double area(Item\* node){**

**if(!node){**

**return 0;**

**}**

**return node->Area() + area(node->bro) + area(node->son);**

**}**

**double TNaryTree::Area(std::string &&tree\_path){**

**Item\* tmp = root;**

**for(int i = 0; i < tree\_path.length(); i++){**

**if(tree\_path[i] == 'b'){**

**if(tmp->bro == nullptr){**

**throw std::invalid\_argument("Path does not exist\n");**

**return -1;**

**}**

**tmp = tmp->bro;**

**} else if(tree\_path[i] == 'c'){**

**if(tmp->son == nullptr){**

**throw std::invalid\_argument("Path does not exist\n");**

**return -1;**

**}**

**tmp = tmp->son;**

**} else {**

**throw std::invalid\_argument("Error in path\n");**

**return -1;**

**}**

**}**

**return area(tmp);**

**}**

**// Вывод дерева в формате вложенных списков, где каждый вложенный список является:**

**// "S0: [S1: [S3, S4: [S5, S6]], S2]", где Si - площадь фигуры**

**void print(std::ostream& os, Item\* node){**

**if(!node){**

**return;**

**}**

**if(node->son){**

**//os << <<node->pentagon.GetArea() << : ]" <<**

**os << node->Area() << ": [";**

**print(os, node->son);**

**if(node->bro){**

**if(node->bro){**

**os << ", ";**

**print(os, node->bro);**

**}**

**}**

**os << "]";**

**} else if (node->bro) {**

**os << node->Area() << ": [";**

**print(os, node->bro);**

**if(node->son){**

**if(node->son){**

**os << ", ";**

**print(os, node->son);**

**}**

**}**

**os << "]";**

**}**

**else {**

**os << node->Area();**

**}**

**}**

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

**print(os, tree.root);**

**os << "\n";**

**}**

**TNaryTree::~TNaryTree(){**

**this->Clear();**

**};**

**TNaryTree\_item.h**

**#ifndef TNARYTREE\_ITEM\_H**

**#define TNARYTREE\_ITEM\_H**

**#include "trapezoid.h"**

**class Item{**

**public:**

**Item\* bro = nullptr;**

**Item\* son = nullptr;**

**Item(Point a\_, Point b\_, Point c\_, Point d\_);**

**Item(Trapezoid a);**

**Item(Item\* a);**

**Item();**

**void Print(std::ostream &os);**

**double Area();**

**~Item();**

**private:**

**Trapezoid data;**

**};**

**#endif**

**TNaryTree\_item.cpp**

**#include "TNaryTree\_item.h"**

**Item::Item(Point a\_, Point b\_, Point c\_, Point d\_){**

**data = Trapezoid(a\_, b\_, c\_, d\_);**

**}**

**Item::Item(){**

**data = Trapezoid();**

**}**

**Item::Item(Trapezoid a){**

**data = a;**

**}**

**Item::Item(Item\* a){**

**bro = a->bro;**

**son = a->son;**

**data = a->data;**

**}**

**void Item::Print(std::ostream &os){**

**os << data.Area();**

**}**

**double Item::Area(){**

**return data.Area();**

**}**

**Item::~Item(){};**

**Результат работы:**

5

17

5

5 17

1: [4, 4: [4: [4]]]

0