**Федеральное государственное автономное образовательное учреждение высшего образования**

**"Московский Авиационный Институт (Национальный Исследовательский Университет"**

Институт “Информационные технологии и прикладная математика”

**Лабораторная работа № 3**

по курсу «ООП»

Тема: Основы метапрограммирования

Студент: Николаева Елизавета

Группа: 80-201

Преподаватель: Чернышов Л.Н.

Дата: 15.01.2021

Оценка:

Москва, 2021

1. Постановка задачи

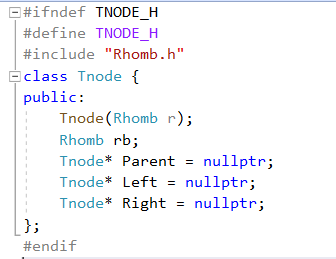
Вариант 15

Бинарное-Дерево

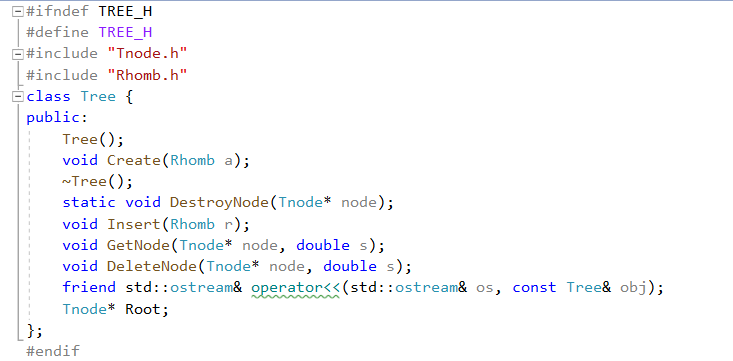
2. Описание программы

Репозиторий: <https://github.com/esnikolaeva/oop/tree/main/oop_exercise_04>

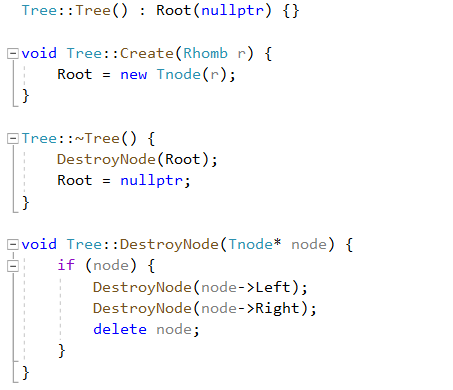
Лист дерева реализован с использованием указателя на родителя, правого и левого потомка + данных (фигуры), хранящихся в нем:



Дерево состоит из листьев и имеет собственные функции, которые будут описаны чуть ниже.



Функции конструкторов и деструкторов реализованы под этим текстом, Create создает дерево, записывая в его корень данные, DestroyNode удаляет все листья, а ~Tree дополнительно удаляет данные корня.



Функция Insert добавляет элемент в дерево следующим способом: сортирует слева направо по увеличению площади фигуры.

Printer печатает дерево с помощью перегруженных операторов вывода, а Printler печатает площади всех элементов дерева.

GetNode печатает первый входящий слева элемент по его площади, minimum и maximum ищут наименьший и наибольший по площади элемент на промежутке, search ищет элемент по площади (надо для удаления элемента).

Класс ромб, для него перегружены все требуемые операторы для удобства пользователя.

Так же в программе реализовано меню.

3. Тестирование

**Тест 01:**

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

1

Введите корень дерева :

Введите длину вертикальной диагонали ромба: 2

Введите длину горизонтальной диагонали ромба: 4

Введите координаты центра ромба в виде x y: 0 0

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

4

Вертикальная диагональ = 2

Горизонтальная диагональ = 4

Центр = (0 ; 0)

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

2

Введите добавляемый ромб:

Введите длину вертикальной диагонали ромба: 5

Введите длину горизонтальной диагонали ромба: 5

Введите координаты центра ромба в виде x y: 3 2

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

4

Вертикальная диагональ = 2

Горизонтальная диагональ = 4

Центр = (0 ; 0)

Вертикальная диагональ = 5

Горизонтальная диагональ = 5

Центр = (3 ; 2)

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

2

Введите добавляемый ромб:

Введите длину вертикальной диагонали ромба: 1

Введите длину горизонтальной диагонали ромба: 1

Введите координаты центра ромба в виде x y: 7 9

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

4

Вертикальная диагональ = 1

Горизонтальная диагональ = 1

Центр = (7 ; 9)

Вертикальная диагональ = 2

Горизонтальная диагональ = 4

Центр = (0 ; 0)

Вертикальная диагональ = 5

Горизонтальная диагональ = 5

Центр = (3 ; 2)

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

5

0.5

4

12.5

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

6

Введите площадь запрашиваемого элемента: 12.5

Вертикальная диагональ = 5

Горизонтальная диагональ = 5

Центр = (3 ; 2)

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

3

Введите площадь удаляемого элемента: 12.5

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

4

Вертикальная диагональ = 1

Горизонтальная диагональ = 1

Центр = (7 ; 9)

Вертикальная диагональ = 2

Горизонтальная диагональ = 4

Центр = (0 ; 0)

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

5

0.5

4

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

3

Введите площадь удаляемого элемента: 4

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

4

Вертикальная диагональ = 1

Горизонтальная диагональ = 1

Центр = (7 ; 9)

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

7

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

8

**Тест 02:**

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

2

Дерево не было создано

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

3

Дерево не было создано

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

4

Дерево не было создано

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

5

Дерево не было создано

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

7

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

1

Введите корень дерева :

Введите длину вертикальной диагонали ромба: 0

Введите длину горизонтальной диагонали ромба: 7

Введите координаты центра ромба в виде x y: 3 3

Введены некорректные параметры

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

4

Дерево не было создано

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

1

Введите корень дерева :

Введите длину вертикальной диагонали ромба: 2 3

Введите длину горизонтальной диагонали ромба: Введите координаты центра ромба в виде x y: 0 0

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

4

Вертикальная диагональ = 2

Горизонтальная диагональ = 3

Центр = (0 ; 0)

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

7

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

4

Дерево не было создано

1. Создать дерево

2. Добавить элемент в дерево

3. Удалить элемент из дерева

4. Распечатать дерево

5. Распечатать площади элементов дерева

6. Вывести элемент из дерева по его площади

7. Удалить дерево

8. Выход

8

4. Листинг

**Tnode.h**

#ifndef TNODE\_H

#define TNODE\_H

#include "Rhomb.h"

class Tnode {

public:

Tnode(Rhomb r);

Rhomb rb;

Tnode\* Parent = nullptr;

Tnode\* Left = nullptr;

Tnode\* Right = nullptr;

};

#endif

Rhomb.h

-----------------------

#ifndef RHOMB\_H

#define RHOMB\_H

#include <iostream>

#include <cstdlib>

class Point {

public:

double x, y;

Point& operator=(const Point& right);

};

class Rhomb {

public:

double Square();

friend std::ostream& operator<<(std::ostream& os, const Rhomb& obj);

friend std::istream& operator>>(std::istream& is, Rhomb& obj);

Rhomb& operator=(const Rhomb& right);

double diag1;

double diag2;

Point center;

};

#endif

**Tree.h**

#ifndef TREE\_H

#define TREE\_H

#include "Tnode.h"

#include "Rhomb.h"

class Tree {

public:

Tree();

void Create(Rhomb a);

~Tree();

static void DestroyNode(Tnode\* node);

void Insert(Rhomb r);

void GetNode(Tnode\* node, double s);

void DeleteNode(Tnode\* node, double s);

friend std::ostream& operator<<(std::ostream& os, const Tree& obj);

Tnode\* Root;

};

#endif

**main.cpp**

#include <iostream>

#include "Tree.h"

#include "Tnode.h"

#include "Rhomb.h"

using namespace std;

/\*Point\*/

Point& Point::operator=(const Point& right) {

x = right.x;

y = right.y;

return \*this;

}

/\*Rhomb\*/

double Rhomb::Square() {

return (diag1 \* diag2) / 2;

}

ostream& operator<<(ostream& os, const Rhomb& obj) {

os << "Вертикальная диагональ = " << obj.diag1 << '\n'

<< "Горизонтальная диагональ = " << obj.diag2 << '\n'

<< "Центр = (" << obj.center.x << " ; " << obj.center.y << ")" << '\n';

return os;

}

istream& operator>>(istream& is, Rhomb& obj) {

cout << "Введите длину вертикальной диагонали ромба: ";

is >> obj.diag1;

cout << "Введите длину горизонтальной диагонали ромба: ";

is >> obj.diag2;

cout << "Введите координаты центра ромба в виде x y: ";

is >> obj.center.x >> obj.center.y;

return is;

}

Rhomb& Rhomb::operator=(const Rhomb& right) {

diag1 = right.diag1;

diag2 = right.diag2;

center = right.center;

return \*this;

}

bool operator==(const Rhomb& a, const Rhomb& b) {

return ((a.diag1 == b.diag1) &&

(a.diag2 == b.diag2) &&

(a.center.x == b.center.x) &&

(a.center.y == b.center.y));

}

/\*Tnode\*/

Tnode::Tnode(Rhomb r) : rb(r) {}

/\*Tree\*/

Tree::Tree() : Root(nullptr) {}

void Tree::Create(Rhomb r) {

Root = new Tnode(r);

}

Tree::~Tree() {

DestroyNode(Root);

Root = nullptr;

}

void Tree::DestroyNode(Tnode\* node) {

if (node) {

DestroyNode(node->Left);

DestroyNode(node->Right);

delete node;

}

}

void Tree::Insert(Rhomb r) {

Tnode\* root2 = Root;

Tnode\* root3 = nullptr;

Tnode\* tmp = new Tnode(r);

while (root2 != nullptr) {

root3 = root2;

if (r.Square() < (root2->rb.Square())) {

root2 = root2->Left;

}

else {

root2 = root2->Right;

}

}

tmp->Parent = root3;

tmp->Left = nullptr;

tmp->Right = nullptr;

if (r.Square() < (root3->rb.Square())) {

root3->Left = tmp;

}

else {

root3->Right = tmp;

}

}

ostream& Printer(ostream& os, Tnode\* Root) {

if (Root == nullptr) {

return os;

}

Printer(os, Root->Left);

os << Root->rb << '\n';

Printer(os, Root->Right);

}

ostream& operator<<(ostream& os, Tree& T) {

return Printer(os, T.Root);

}

void Printler(Tnode\* Root) {

if (Root == nullptr) {

return;

}

Printler(Root->Left);

cout << Root->rb.Square() << '\n';

Printler(Root->Right);

}

void Tree::GetNode(Tnode\* node, double s) {

if (node != nullptr) {

if (s == node->rb.Square()) {

cout << node->rb << '\n';

return;

}

else if (s > node->rb.Square()) {

GetNode(node->Right, s);

}

else if (s < node->rb.Square()) {

GetNode(node->Left, s);

}

}

else {

cout << "Элемента с такой площадью нет" << '\n';

return;

}

}

Tnode\* minimum(Tnode\* node) {

if (node->Left == nullptr) {

return node;

}

else {

return minimum(node->Left);

}

}

Tnode\* maximum(Tnode\* node) {

if (node->Right == nullptr) {

return node;

}

else {

return maximum(node->Right);

}

}

Tnode\* search(Tnode\* root, double value) {

if ((root == NULL) || (root->rb.Square() == value)) {

return root;

}

if (value < root->rb.Square()) {

return search(root->Left, value);

}

else {

return search(root->Right, value);

}

}

void Tree::DeleteNode(Tnode\* root, double value) {

Tnode \*l = nullptr, \*m = nullptr;

l = search(root, value);

if (Root->rb.Square() == value) {

if ((Root->Left == nullptr) && (Root->Right == nullptr)) {

Root = nullptr;

return;

}

else if ((Root->Left == nullptr) && (Root->Right != nullptr)) {

m = minimum(Root->Right);

if (m != Root->Right) {

Tnode\* p = m->Parent;

p->Left = nullptr;

Root->rb = m->rb;

delete(m);

return;

}

else {

Tnode\* p = Root;

Root = Root->Right;

delete(p);

return;

}

}

else if ((Root->Left != nullptr) && (Root->Right == nullptr)) {

m = maximum(Root->Left);

if (m != Root->Left) {

Tnode\* p = m->Parent;

p->Right = nullptr;

Root->rb = m->rb;

delete(m);

return;

}

else {

Tnode\* p = Root;

Root = Root->Left;

delete(p);

return;

}

}

else if ((Root->Left != nullptr) && (Root->Right != nullptr)) {

m = minimum(Root->Right);

if (m != Root->Right) {

Tnode\* p = m->Parent;

p->Left = nullptr;

Root->rb = m->rb;

delete(m);

return;

}

else {

Tnode\* p = Root;

Root = Root->Right;

Root->Left = p->Left;

delete(p);

return;

}

}

}

else if ((l->Left == nullptr) && (l->Right == nullptr)) {

m = l->Parent;

if (l == m->Left) {

m->Left = nullptr;

}

else {

m->Right = nullptr;

}

delete(l);

}

else if ((l->Left == nullptr) && (l->Right != nullptr)) {

m = l->Parent;

if (l == m->Left) {

m->Left = l->Right;

}

else {

m->Right = l->Right;

}

delete(l);

}

else if ((l->Left != nullptr) && (l->Right == nullptr)) {

m = l->Parent;

if (l == m->Left) {

m->Left = l->Left;

}

else {

m->Right = l->Left;

}

delete(l);

}

else if ((l->Left != nullptr) && (l->Right != nullptr)) {

m = minimum(l->Right);

l->rb = m->rb;

if ((m->Right != nullptr) && (m->Parent != l)) {

m->Parent->Left = m->Right;

}

if ((m->Right == nullptr) && (m->Parent != l)) {

m->Parent->Left = nullptr;

}

if ((m->Right != nullptr) && (m->Parent == l)) {

l->Right = m->Right;

}

if ((m->Right == nullptr) && (m->Parent == l)) {

l->Right = nullptr;

}

delete(m);

}

}

int main() {

setlocale(LC\_ALL, "Russian");

cout << "Программа позволяет взаимодействовать с деревом ромбов, представлена 4 файлами: Tnode.h, Rhomb.h, Tree.h, lab4\_oop.cpp ." << '\n';

int menu, f = 1;

Rhomb a;

Tree T;

while (f) {

cout << "1. Создать дерево" << '\n';

cout << "2. Добавить элемент в дерево" << '\n';

cout << "3. Удалить элемент из дерева" << '\n';

cout << "4. Распечатать дерево" << '\n';

cout << "5. Распечатать площади элементов дерева" << '\n';

cout << "6. Вывести элемент из дерева по его площади" << '\n';

cout << "7. Удалить дерево" << '\n';

cout << "8. Выход" << '\n';

cin >> menu;

switch (menu) {

case 1: {

if (T.Root != nullptr) {

cout << "Дерево уже было создано" << '\n';

break;

}

cout << "Введите корень дерева :" << '\n';

cin >> a;

if (a.Square() == 0) {

cout << "Введены некорректные параметры" << '\n';

break;

}

T.Create(a);

break;

}

case 2: {

if (T.Root == nullptr) {

cout << "Дерево не было создано" << '\n';

break;

}

cout << "Введите добавляемый ромб:" << '\n';

cin >> a;

if (a.Square() == 0) {

cout << "Введены некорректные параметры" << '\n';

break;

}

T.Insert(a);

break;

}

case 3: {

if (T.Root == nullptr) {

cout << "Дерево не было создано" << '\n';

break;

}

double as;

cout << "Введите площадь удаляемого элемента: ";

cin >> as;

T.DeleteNode(T.Root, as);

break;

}

case 4: {

if (T.Root == nullptr) {

cout << "Дерево не было создано" << '\n';

break;

}

cout << T;

break;

}

case 5: {

if (T.Root == nullptr) {

cout << "Дерево не было создано" << '\n';

break;

}

cout << '\n';

Printler(T.Root);

break;

}

case 6: {

double as;

cout << "Введите площадь запрашиваемого элемента: ";

cin >> as;

T.GetNode(T.Root, as);

break;

}

case 7: {

T.~Tree();

break;

}

case 8: {

f = 0;

break;

}

default: {

cout << "wrong command" << '\n';

break;

}

}

}

return 0;

}

5. Список литературы

Справочник по языку [Электронный ресурс]. URL: <https://ravesli.com/uroki-cpp/> (дата обращения 15.01.2021).

Справочник по языку [Электронный ресурс]. URL:

<https://metanit.com/cpp/tutorial/> (дата обращения 15.01.2021).