Московский Авиационный Институт

(Национальный Исследовательский Университет)

Лабораторная работа №5 по курсу

«Объектно-ориентированное программирование»

Студент: Шаларь Игорь Павлович
Группа: М8О-208Б-20
Вариант: 23
Преподаватель: Дорохов Евгений Павлович
Оценка: _____
Дата: _____

Цель:

-Закрепление навыков работы с классами;
-Знакомство с умными указателями.

Задание:

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 лабораторной работы №1;
- Требования к классу контейнера аналогичны требованиям из лабораторной работы №2;
- Класс-контейнер должен содержать объекты используя std::shared_ptr<…>.

Нельзя использовать:

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

Программа должна позволять:

- Вводить произвольное количество фигур и добавлять их в контейнер;
- Распечатывать содержимое контейнера;
- Удалять фигуры из контейнера.

Вариант 23: hexagon tnarytree

Описание программы:

figure.h - описание родительского класса для класса-фигуры.

hexagon.h, tnarytree.h - заголовочный файл описывающий классы.

hexagon.cpp, tnarytree.cpp - реализация.

main.cpp - основной файл, взаимодействие с пользователем.

Пример работы:

```
USAGE:
Add element:
a [x1] [y1] [x2] [y2] [x3] [y3] [x4] [y4] [x5] [y5] [x6] [y6]
[route]
example:
a 0 2 1 1 1 -1 0 -2 -1 -1 -1 1
cbb
Print: p
Delete:
d
[path]
Area:
s
[path]
Stop: q
```

```
Enter tree size:
100
a 0 2 1 1 1 -1 0 -2 -1 -1 -1 1

added
a 0 0 0 0 0 0 0 0 0 0 0 0
c
added
p
6: [0]
a 0 2 1 1 1 -1 0 -2 -1 -1 -1 1
cb
added
p
6: [0, 6]
s

12
d
c
deleted
p
6: [6]
q
```

Дневник отладки:

Были проблемы с использованием умных указателей при удалении вершин. Ошибки были исправлены.

Выводы:

Познакомился с умными указателями и научился применять их на практике. Их оказалось довольно удобно использовать для моей структуры данных, так как при удалении можно не обходить всю структуру, а перевесить максимум один-два указателя и удалить указатель на нужную вершину. Тогда всё поддерево удалится автоматически благодаря умным

указателям. Также работать с памятью стало легче, так как память теперь освобождается автоматически.

Исходный код:

CMakeLists.txt:

```
cmake_minimum_required(VERSION 3.20)

set(CMAKE_CXX_STANDARD 14)

add_executable(lab2 main.cpp hexagon.cpp tnarytree.cpp)
```

figure.h:

```
#ifndef FIGURE_H
#define FIGURE_H

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

#endif
```

hexagon.h:

```
#ifndef HEXAGON_H
#define HEXAGON_H
```

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


class Hexagon : public Figure{

public:
    friend std::istream &operator >> (std::istream &is, Hexagon &p);
    friend std::ostream &operator << (std::ostream &os, Hexagon &p);
    Hexagon &operator = (const Hexagon &right);
    bool operator == (const Hexagon &right);
    size_t VerticesNumber();
    double Area();

private:
    std::pair <double, double> p[6];
};


#endif
```

tnarytree.h:

```
#ifndef TNARYTREE_H
#define TNARYTREE_H

#include "hexagon.h"

using namespace std;
```

```
class node{

public:
    shared_ptr <node> brt, cld;
    shared_ptr <Hexagon> val;

    node(){
        cld = NULL;
        brt = NULL;
        val = NULL;
    }

};

class TNaryTree {
public:
    // Инициализация дерева с указанием размера
    TNaryTree(int);

    // 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.
    void Update(Hexagon &now, 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);

    // 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);

    ~TNaryTree();

private:
    shared_ptr <node> root;
    shared_ptr <node> now;
    int size;
    int max_size;
    int get_size();
    void check();
    void decrease();
    void node_delete(shared_ptr <node>, TNaryTree &, shared_ptr <node>);
    void change_val(Hexagon &);
};

#endif
```

hexagon.cpp:

```
#include"hexagon.h"
#include<iostream>
#include<math.h>

using namespace std;

std::istream &operator >> (std::istream &is, Hexagon &h){
    for (int i = 0; i < 6; i++) is >> h.p[i].first >> h.p[i].second;
    return is;
}

std::ostream &operator << (std::ostream &os, Hexagon &h){
    os << "Hexagon:";
    for (int i = 0; i < 6; i++) os << " (" << h.p[i].first << ", " << h.p[i].second << ")";
    return os;
}

double hex_distance(pair <double, double> a, pair<double, double> b){
    return sqrt(pow(a.first - b.first, 2) + pow(a.second - b.second, 2));
}

double hex_S_triangle (double a, double b, double c){
    double p = (a + b + c) / 2;
    return sqrt(p * (p - a) * (p - b) * (p - c));
}

size_t Hexagon::VerticesNumber(){
```

```
    return 6;
}

double Hexagon::Area(){
    double res = 0, a, b, c;
    for (int i = 1; i < 5; i++){
        a = hex_distance(this->p[0], this->p[i]);
        b = hex_distance(this->p[i], this->p[i + 1]);
        c = hex_distance(this->p[0], this->p[i + 1]);
        res += hex_S_triangle(a, b, c);
    }
    return res;
}

Hexagon & Hexagon::operator = (const Hexagon &right){
    if (this == &right) return *this;
    for (int i = 0; i < 6; i++) this->p[i] = right.p[i];
    return *this;
}

double find_eps(){
    double eps = 1;
    while (eps + 1 != 1){
        eps /= 2;
    }
    return eps;
}

double eps = find_eps();
```

```
bool Hexagon::operator == (const Hexagon &right){
    bool p = 1;
    for (int i = 0; i < 6; i++){
        if (abs(this->p[i].first - right.p[i].first) >= eps || abs(this->p[i].second - right.p[i].second) >= eps)
            p = 0;
    }
    return p;
}
```

tnarytree.cpp:

```
#include"hexagon.h"
#include"tnarytree.h"
#include<iostream>
#include <stdexcept>

using namespace std;

void TNaryTree::check(){
    if (size >= max_size){
        throw out_of_range("Nodes limit exceeded.");
        exit(-1);
    }
}

TNaryTree::TNaryTree(int n){
    max_size = n;
    root = NULL;
    size = 0;
}
```

```
void assign (shared_ptr <node> left,  Hexagon & right){
    if (left->val == NULL){
        Hexagon * ptr = new Hexagon;
        * ptr = right;
        shared_ptr <Hexagon> temp(ptr);
        left->val.swap(temp);
        return;
    }
    * left->val = right;
}

shared_ptr <node> create_node(shared_ptr <node> other){
    node * temp = new node;
    shared_ptr <node> ptr(temp);
    ptr->val = other->val;
    return ptr;
}

void TNaryTree::decrease(){
    size--;
}

int TNaryTree::get_size(){
    return size;
}

void node_copy(shared_ptr <node> now, const shared_ptr <node> other, char p){
    if (other == NULL) return;
    if (p == 'c') {
```

```
        now->cld = create_node(other);
        now = now->cld;
    }
    else{
        now->brt = create_node(other);
        now = now->brt;
    }
    node_copy (now, other->cld, 'c');
    node_copy (now, other->brt, 'b');
}

// Полное копирование дерева

TNaryTree::TNaryTree(const TNaryTree& other){
    if (other.root == NULL) return;
    size = other.size;
    max_size = other.max_size;
    root = create_node(other.root);
    node_copy (root, other.root->cld, 'c');
}

// 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.
void TNaryTree::Update(Hexagon &h, std::string &tree_path){
    if (tree_path == ""){
        if (root == NULL){
            check();
```

```
            node * ptr = new node;
            shared_ptr <node> temp(ptr);
            assign(temp, h);
            root = temp;
            size++;
        }
        else assign(root, h);
        return;
    }
    if (root == NULL || tree_path[0] == 'b'){
        throw invalid_argument("Invalid tree_path.");
        exit(-1);
    }
    now = root;
    for (int i = 0; i < tree_path.size() - 1; i++){
        if (tree_path[i] == 'c') now = now->cld;
        else now = now->brt;
        if (now == NULL){
            throw invalid_argument("Invalid tree_path.");
            exit(-1);
        }
    }
    if (tree_path[tree_path.size() - 1] == 'c'){
        if (now->cld == NULL){
            check();
            node * ptr = new node;
            shared_ptr <node> temp(ptr);
            assign(temp, h);
            now->cld = temp;
            size++;
```

```
        }
        else assign(now->cld, h);
    }
    else{
        if (now->brt == NULL){
            check();
            node * ptr = new node;
            shared_ptr <node> temp(ptr);
            assign(temp, h);
            now->brt = temp;
            size++;
        }
        else assign(now->brt, h);
    }
}

void TNaryTree::node_delete(shared_ptr <node> now1, TNaryTree &t, shared_ptr <node> rt){
    if (now1==NULL) return;
    t.decrease();
    node_delete(now1->cld, t, rt);
    if (now1 != rt) node_delete(now1->brt, t, rt);
}

// Удаление поддерева
void TNaryTree::Clear(std::string &tree_path){
    now = NULL;
    if (tree_path == ""){
        node_delete(root, * this, root);
        root = NULL;
        return;
```

```
}
if (root == NULL || tree_path[0] == 'b'){
    throw invalid_argument("Invalid tree_path.");
    exit(-1);
}
now = root;
for (int i = 0; i < tree_path.size() - 1; i++){
    if (tree_path[i] == 'c'){
        now = now->cld;
    }
    else now = now->brt;
    if (now == NULL){
        throw invalid_argument("Invalid tree_path.");
        exit(-1);
    }
}
if (tree_path[tree_path.size() - 1] == 'c'){
    if (now->cld == NULL){
        throw invalid_argument("Invalid tree_path.");
        exit(-1);
    }
    node_delete (now->cld, * this, now->cld);
    now->cld = now->cld->brt;
}
else{
    if (now->brt == NULL){
        throw invalid_argument("Invalid tree_path.");
        exit(-1);
    }
    node_delete (now->brt, * this, now->brt);
```

```
        now->brt = now->brt->brt;
    }
}

// Проверка наличия в дереве вершин
bool TNaryTree::Empty(){
    return root == NULL;
}

double node_square(shared_ptr <node> now){
    if (now == NULL) return 0;
    double res = now->val->Area();
    res += node_square(now->cld);
    res += node_square(now->brt);
    return res;
}

// Подсчет суммарной площади поддерева
double TNaryTree::Area(std::string &tree_path){
    if (tree_path == "") return node_square(root);
    if (root == NULL || tree_path[0] == 'b'){
        throw invalid_argument("Invalid tree_path.");
        exit(-1);
    }
    now = root;
    for (int i = 0; i < tree_path.size() - 1; i++){
        if (tree_path[i] == 'c') now = now->cld;
        else now = now->brt;
        if (now == NULL){
            throw invalid_argument("Invalid tree_path.");
```

```
            exit(-1);
        }
    }
    if (tree_path[tree_path.size() - 1] == 'c'){
        if (now->cld == NULL){
            throw invalid_argument("Invalid tree_path.");
            exit(-1);
        }
        return node_square (now->cld);
    }
    else{
        if (now->brt == NULL){
            throw invalid_argument("Invalid tree_path.");
            exit(-1);
        }
        return node_square (now->brt);
    }
    return 0;
}

// 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, shared_ptr <node> v, bool p){
    os << v->val->Area();
    if (v->cld != NULL){
        os << ": [";
        print(os, v->cld, 1);
    }
    if (v->brt != NULL){
        os << ", ";
```

```
        print(os, v->brt, 1);
    }
    else if (p) os << "]";
}
std::ostream& operator<<(std::ostream& os, const TNaryTree& tree){
    if (tree.root == NULL) return os;
    print(os, tree.root, 0);
    return os;
}

TNaryTree::~TNaryTree(){
    string s = "";
    Clear(s);
}
```

main.cpp:

```
#include<iostream>
#include"hexagon.h"
#include"tnarytree.h"

using namespace std;

int main(){
    cout << endl << "USAGE:" << endl;
    cout << "Add element:" << endl;
    cout << "a [x1] [y1] [x2] [y2] [x3] [y3] [x4] [y4] [x5] [y5] [x6] [y6]" << endl << "[route]" << endl;
    cout << "example:" << endl << "a 0 2 1 1 1 -1 0 -2 -1 -1 -1 1" << endl << "cbb" << endl;
```

```
cout << "Print: p" << endl;
cout << "Delete:" << endl << "d" << endl << "[path]" << endl;
cout << "Area:" << endl << "s" << endl << "[path]" << endl;
cout << "Stop: q" << endl;
cout << "_____" << endl;
int n;
cout << "Enter tree size:" << endl;
cin >> n;
TNaryTree * temp = new TNaryTree(n);
shared_ptr <TNaryTree> t(temp);
char ch;
string s;
Hexagon h;
while (1){
    cin >> ch;
    switch (ch){
        case 'a': {
            cin >> h;
            getline(cin, s);
            getline(cin, s);
            t->Update(h, s);
            cout << "added" << endl;
            break;
        }
        case 'p': {
            cout << * t << endl;
            break;
        }
        case 'd': {
            getline(cin, s);
```

```
            getline(cin, s);
            t->Clear(s);
            cout << "deleted "  << endl;
            break;
        }
        case 's': {
            getline(cin, s);
            getline(cin, s);
            cout << t->Area(s) << endl;
            break;
        }
        case 'q': {
            s = "";
            t->Clear(s);
            return 0;
        }
    }
  }
  return 0;
}
```