**Московский авиационный институт**

**(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Дисциплина: «Объектно-ориентированное программирование»

**Лабораторная работа № 4**

Тема: Основы метапрограммирования

Студент: Лисин Роман Сергеевич

Группа: 80-206

Преподаватель: Чернышов Л.Н.

Дата:

Оценка:

Москва, 2021

1. **Постановка задачи**

Необходимо спроектировать и запрограммировать на языке C++ N-дерево, содержащее треугольники.

Классы должны удовлетворять следующим правилам:

· Требования к классу фигуры аналогичны требованиям из лабораторной работы 3

· Классы фигур должны иметь переопределенный оператор вывода в поток std::ostream (<<). Оператор должен распечатывать параметры фигуры (тип фигуры, длины сторон, радиус и т.д).

· Классы фигур должны иметь переопределенный оператор ввода фигуры из потока std::istream (>>). Оператор должен вводить основные параметры фигуры (длины сторон, радиус и т.д).

· Классы фигур должны иметь операторы копирования (=).

· Классы фигур должны иметь операторы сравнения с такими же фигурами (==).

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

· Класс-контейнер должен иметь метод по добавлению фигуры в контейнер.

· Класс-контейнер должен иметь методы по получению фигуры из контейнера (определяется структурой контейнера).

· Класс-контейнер должен иметь метод по удалению фигуры из контейнера (определяется структурой контейнера).

· Класс-контейнер должен иметь перегруженный оператор по выводу контейнера в поток std::ostream (<<).

· Класс-контейнер должен иметь деструктор, удаляющий все элементы контейнера.

· Классы должны быть расположены в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

1. **Описание программы**

Программа сама запрашивает входные данные, формирует дерево и показывает свой функционал.

1. **Набор тестов**

4

1 2 3

6

4 5 6

7 8 9

10 11 12

13 14 15

16 17 18

19 20 21

1

3

1 2 3

6

4 5 6

7 8 9

10 11 12

13 14 15

16 17 18

19 20 21

3

1. **Результаты выполнения тестов**

Введите количество детей у узла в N-дереве: 3

Введите треугольник-корень N-дерева: 1 2 3

Введите количество остальных треугольников в N-дереве: 6

4 5 6

7 8 9

10 11 12

13 14 15

16 17 18

19 20 21

Печать дерева:

\\_\_a=1, b=2, c=3

\\_\_a=4, b=5, c=6

\\_\_a=13, b=14, c=15

\\_\_a=16, b=17, c=18

\\_\_a=19, b=20, c=21

\\_\_a=7, b=8, c=9

\\_\_a=10, b=11, c=12

Какой элемент дерева удалить? 3

Печать дерева:

\\_\_a=1, b=2, c=3

\\_\_a=4, b=5, c=6

\\_\_a=13, b=14, c=15

\\_\_a=16, b=17, c=18

\\_\_a=19, b=20, c=21

\\_\_a=10, b=11, c=12

Посмотрим какой у дерева корень: a=1, b=2, c=3

1. **Листинг программы**

Ntree.h

#ifndef NTREE\_H

#define NTREE\_H

#include "Triangle.h"

#include <vector>

using namespace std;

struct node {

Triangle value; // data

vector<node\*> children; // child nodes

node(Triangle v, int max\_children\_size) {

value = v;

children.resize(max\_children\_size);

}

};

class Ntree {

public:

Ntree();

Ntree(Triangle value, int N);

bool insert(Triangle value);

bool find(int pos);

bool remove(int pos);

friend ostream& operator<<(ostream& os, const Ntree& obj);

node\* get\_root();

~Ntree();

private:

node\* root;

int N;

};

Ntree.cpp

#include "Ntree.h"

#include <queue>

#include <vector>

#include <iostream>

using namespace std;

Ntree::Ntree() {}

Ntree::Ntree(Triangle value, int N) {

this->root = new node(value, N);

if (!this->root) {

throw "N-tree wasn't created";

}

this->N = N;

}

bool Ntree::insert(Triangle value) {

queue<node\*> nodes\_to\_visit;

node\* current\_node;

nodes\_to\_visit.push(root);

while (!nodes\_to\_visit.empty()) {

current\_node = nodes\_to\_visit.front();

nodes\_to\_visit.pop();

for (int i = 0; i < N; i++) {

if (!current\_node->children[i]) {

current\_node->children[i] = new node(value, N);

if (!current\_node->children[i]) {

return false;

}

else {

return true;

}

}

else {

nodes\_to\_visit.push(current\_node->children[i]);

}

}

}

}

bool Ntree::find(int pos) {

queue<node\*> nodes\_to\_visit;

node\* current\_node;

nodes\_to\_visit.push(root);

if (root == NULL) {

throw "В дереве нет этого элемента";

return false;

}

while (!nodes\_to\_visit.empty()) {

current\_node = nodes\_to\_visit.front();

if (!current\_node) {

throw "В дереве нет этого элемента";

return false;

}

nodes\_to\_visit.pop();

if (pos == 1) {

cout << current\_node->value << '\n';

return true;

}

for (int i = 0; i < N; i++) {

if (current\_node->children[i]) {

nodes\_to\_visit.push(current\_node->children[i]);

}

}

pos--;

}

}

bool Ntree::remove(int pos) {

queue<node\*> nodes\_to\_visit;

node\* current\_node;

nodes\_to\_visit.push(root);

while (!nodes\_to\_visit.empty()) {

current\_node = nodes\_to\_visit.front();

nodes\_to\_visit.pop();

if (pos == 1) {

delete current\_node;

root = nullptr;

return true;

}

for (int i = 0; i < N; i++) {

if (current\_node->children[i]) {

pos--;

if (pos == 1) {

delete current\_node->children[i];

current\_node->children[i] = NULL;

return true;

}

nodes\_to\_visit.push(current\_node->children[i]);

}

}

}

return false;

}

void PrintTree(node\* current\_node, ostream& os, int N) {

static int level = 0;

level++;

if (current\_node) {

os << "\\\_\_" << current\_node->value << '\n';

for (int j = 0; (j < N); j++) {

if (current\_node->children[j]) {

for (int i = 0; i < level; i++) {

cout << " ";

}

PrintTree(current\_node->children[j], os, N);

}

}

}

level--;

}

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

PrintTree(obj.root, os, obj.N);

return os;

}

Ntree::~Ntree() {

queue<node\*> nodes\_to\_visit;

node\* current\_node;

nodes\_to\_visit.push(root);

while (!nodes\_to\_visit.empty()) {

current\_node = nodes\_to\_visit.front();

nodes\_to\_visit.pop();

for (int i = 0; i < N; i++) {

if (current\_node && current\_node->children[i]) {

nodes\_to\_visit.push(current\_node->children[i]);

}

}

delete current\_node;

}

}

node\* Ntree::get\_root() {

return root;

}

N-tree\_oop\_exercise\_04.cpp

#include <iostream>

#include <clocale>

#include "Ntree.h"

/\*Лабораторная работа 4, подготовил Лисин Роман, группа М8О-206Б-20

Вариант 21: N-дерево с треугольниками

\*/

int main() {

setlocale(LC\_ALL, "rus");

int N, triangles;

cout << "Введите количество детей у узла в N-дереве: ";

if (!(cin >> N)) {

cout << "Ошибка ввода N" << '\n';

abort();

}

Triangle root;

cout << "Введите треугольник-корень N-дерева: ";

if (!(cin >> root)) {

cout << "Ошибка ввода треугольника" << '\n';

abort();

}

cout << "Введите количество остальных треугольников в N-дереве: ";

if (!(cin >> triangles)) {

cout << "Ошибка ввода количество треугольников" << '\n';

abort();

}

Ntree t = {root, N};

if (triangles < 1) {

cout << "Треугольников должно быть больше одного" << '\n';

abort();

}

for (int i = 0; i < triangles; i++) {

Triangle tr;

if (!(cin >> tr)) {

cout << "Ошибка ввода треугольника" << '\n';

abort();

}

if (tr.Square() )

t.insert(tr);

}

cout << "Печать дерева: " << '\n';

cout << t;

cout << "Какой элемент дерева удалить? ";

int pos;

if (!(cin >> pos) || pos < 1 || pos > triangles) {

cout << "Ошибка ввода позиции для удаления" << '\n';

abort();

}

t.remove(pos);

cout << "Печать дерева: " << '\n';

if (t.get\_root()) {

cout << t;

}

cout << "Посмотрим какой у дерева корень: ";

if (pos == 1) {

cout << "Дерева больше нет" << '\n';

}

else {

t.find(1);

}

return 0;

}

Triangle.h

#ifndef TRIANGLE\_H

#define TRIANGLE\_H

#include <cstdlib>

#include <iostream>

class Triangle {

public:

Triangle();

Triangle(size\_t i, size\_t j, size\_t k);

Triangle(const Triangle& orig);

Triangle& operator++();

double Square();

friend Triangle operator+(const Triangle& left, const Triangle& right);

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

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

Triangle& operator=(const Triangle& right);

virtual ~Triangle();

private:

size\_t side\_a;

size\_t side\_b;

size\_t side\_c;

};

#endif

Triangle.cpp

#include "Triangle.h"

#include <iostream>

#include <cmath>

Triangle::Triangle() : Triangle(0, 0, 0) {

}

Triangle::Triangle(size\_t i, size\_t j, size\_t k) : side\_a(i), side\_b(j), side\_c(k) {

}

Triangle::Triangle(const Triangle& orig) {

side\_a = orig.side\_a;

side\_b = orig.side\_b;

side\_c = orig.side\_c;

}

double Triangle::Square() {

double p = double(side\_a + side\_b + side\_c) / 2.0;

return sqrt(p \* (p - double(side\_a)) \* (p - double(side\_b)) \* (p - double(side\_c)));

}

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

if (this == &right) return \*this;

side\_a = right.side\_a;

side\_b = right.side\_b;

side\_c = right.side\_c;

return \*this;

}

Triangle& Triangle::operator++() {

side\_a++;

side\_b++;

side\_c++;

return \*this;

}

Triangle operator+(const Triangle& left, const Triangle& right) {

return Triangle(left.side\_a + right.side\_a, left.side\_b + right.side\_b, left.side\_c + right.side\_c);

}

Triangle::~Triangle() {

}

std::ostream& operator<<(std::ostream& os, const Triangle& obj) {

os << "a=" << obj.side\_a << ", b=" << obj.side\_b << ", c=" << obj.side\_c << std::endl;

return os;

}

std::istream& operator>>(std::istream& is, Triangle& obj) {

is >> obj.side\_a;

is >> obj.side\_b;

is >> obj.side\_c;

return is;

}

**ЛИТЕРАТУРА**

1. N-tree [Электронный ресурс].

URL: https://en.wikipedia.org/wiki/M-ary\_tree (дата обращения: 05.11.2021)