-7

Специальность: 09.02.07 Информационные системы и программирование

Выполнил:

Студент группы Y2333

Сааль С. Ф.

ОТЧЕТ

о лабораторной работе № 7

по теме: Контейнеры

по дисциплине: Разработка программных модулей

Санкт-Петербург 2020

Проверил:

Антонов. М.Б.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

УНИВЕРСИТЕТ ИТМО

Факультет среднего профессионального образования

1. Цель и задачи

Цель: реализация контейнерного класса.

1. Задание

Создать шаблон класса BinaryTree. Предусмотреть генерацию и обработку исключений для возможных ошибочных ситуаций. Показать использование в клиенте main() созданного класса, включая ситуации, приводящие к генерации исключений. Показать инстанцирование шаблона для классов int, double, std::string.

1. КОд программы

Файл main.cpp:

/\*\*

\* Точка входа

\* @file main.com

\* @author Сааль Степан

\*/

#include "TypeMenu.h"

/\*\*

\* Вывод меню выбора типа

\* @return

\*/

int main() {

TypeMenu::show();

return 0;

}

Файл BinaryTree.h:

/\*\*

\* Бинарное дерево

\* @file BinaryTree.h

\* @author Сааль Степан

\*/

#ifndef LAB7\_BINARYTREE\_H

#define LAB7\_BINARYTREE\_H

#include <stdexcept>

#include <string>

using namespace std;

/\*\*

\* Узел дерева

\* @tparam A

\*/

template<typename A>

struct node {

A value;

node \*left = nullptr;

node \*right = nullptr;

};

/\*\*

\* Бинарное дерево

\* @tparam T

\*/

template<typename T>

class BinaryTree {

public:

/\*\*

\* Создание пустого дерева

\*/

BinaryTree() {

root = nullptr;

}

/\*\*

\* Удаление всего дерева

\*/

~BinaryTree() {

deleteTree();

}

/\*\*

\* Вставка значения

\* @param value

\*/

void insert(T value) {

if (root == nullptr) {

root = new node<T>;

root->value = value;

} else {

insert(value, root);

}

}

/\*\*

\* Удаление дерева

\*/

void deleteTree() {

deleteNode(root);

root = nullptr;

}

/\*\*

\* Поиск значения в дереве

\* @param value

\* @return

\*/

node<T> \*search(T value) {

node<T> \*leaf = search(value, root);

return leaf;

}

/\*\*

\* Печать дерева на экран

\*/

void print() {

if (isEmpty()) {

throw runtime\_error("Tree is empty");

}

print(root);

cout << endl;

}

/\*\*

\* Печать части дерева на экран

\* @param value

\*/

void print(T value) {

node<T> \*leaf = search(value, root);

print(leaf);

cout << endl;

}

/\*\*

\* Проверка на пустоту

\* @return

\*/

bool isEmpty() {

return root == nullptr;

}

/\*\*

\* Кол-во элементов в дереве

\* @return

\*/

int count() {

int count = 0;

countElements(root, count);

return count;

}

private:

/\*\* Вершина дерева \*/

node<T> \*root;

/\*\*

\* Удаление ветви дерева

\* @param leaf

\*/

void deleteNode(node<T> \*leaf) {

if (leaf != nullptr) {

deleteNode(leaf->left);

deleteNode(leaf->right);

delete leaf;

}

}

/\*\*

\* Вставка значения

\* @param value

\* @param leaf

\*/

void insert(T value, node<T> \*leaf) {

if (value < leaf->value) {

insertLeft(value, leaf);

} else {

insertRight(value, leaf);

}

}

/\*\*

\* Вставка значения в левую ветвь

\* @param value

\* @param leaf

\*/

void insertLeft(T value, node<T> \*leaf) {

if (leaf->left != nullptr) {

insert(value, leaf->left);

} else {

leaf->left = new node<T>;

leaf->left->value = value;

}

}

/\*\*

\* Вставка значения в правую ветвь

\* @param value

\* @param leaf

\*/

void insertRight(T value, node<T> \*leaf) {

if (leaf->right != nullptr) {

insert(value, leaf->right);

} else {

leaf->right = new node<T>;

leaf->right->value = value;

}

}

/\*\*

\* Поиск значения

\* @param value

\* @param leaf

\* @return

\*/

node<T> \*search(T value, node<T> \*leaf) {

if (leaf != nullptr) {

if (value == leaf->value) {

return leaf;

} else if (value < leaf->value) {

return search(value, leaf->left);

} else {

return search(value, leaf->right);

}

} else {

throw runtime\_error("Element does not exists");

}

}

/\*\*

\* Вывод на экран

\* @param leaf

\*/

void print(node<T> \*leaf) {

if (leaf != nullptr) {

print(leaf->left);

cout << leaf->value << ", ";

print(leaf->right);

}

}

/\*\*

\* Подсчёт элементов

\* @param leaf

\* @param count

\*/

void countElements(node<T> \*leaf, int &count) {

if (leaf != nullptr) {

count++;

countElements(leaf->left, count);

countElements(leaf->right, count);

}

}

};

#endif //LAB7\_BINARYTREE\_H

Файл Menu.h:

/\*\*

\* Меню для работы с деревом

\* @file Menu.h

\* @author Сааль Степан

\*/

#ifndef LAB7\_MENU\_H

#define LAB7\_MENU\_H

#include <iostream>

#include <string>

#include "BinaryTree.h"

using namespace std;

/\*\*

\* Меню для работы с деревом

\* @tparam T

\*/

template<typename T>

class Menu {

public:

/\*\*

\* Создание дерева

\*/

Menu() {

tree = new BinaryTree<T>();

}

/\*\*

\* Удаление дерева

\*/

~Menu() {

tree->deleteTree();

delete tree;

}

/\*\*

\* Показать меню

\*/

void show() {

mainMenu();

}

private:

/\*\* Дерево \*/

BinaryTree<T> \*tree;

/\*\*

\* Главное меню

\*/

void mainMenu() {

cout << "1. Insert value to tree" << endl;

cout << "2. Search value in tree" << endl;

cout << "3. Print whole tree" << endl;

cout << "4. Print node with value" << endl;

cout << "5. Get count of elements" << endl;

cout << "6. Check if tree is empty" << endl;

cout << "7. Delete tree" << endl;

cout << "0. Exit" << endl;

mainMenuAction();

}

/\*\*

\* Действия главного меню

\*/

void mainMenuAction() {

int action;

cin >> action;

switch (action) {

case 1: insertAction(); break;

case 2: searchAction(); break;

case 3: printAction(true); break;

case 4: printAction(false); break;

case 5: countAction(); break;

case 6: emptyAction(); break;

case 7: deleteAction(); break;

case 0: cout << "Bye!" << endl; return;

default: cout << "Wrong action, try again:" << endl;

}

cout << endl;

mainMenu();

}

/\*\*

\* Вставка данных

\*/

void insertAction() {

cout << "Input value:" << endl;

T value;

cin >> value;

tree->insert(value);

cout << "Done!" << endl;

}

/\*\*

\* Поиск значения

\*/

void searchAction() {

cout << "Input value to searching:" << endl;

T value;

cin >> value;

try {

tree->search(value);

cout << "Element exists!" << endl;

} catch (const exception& e) {

cout << e.what() << endl;

}

}

/\*\*

\* Вывод на экран дерева

\* @param whole Требуется ли выводить всё дерево или часть

\*/

void printAction(bool whole) {

try {

if (whole) {

tree->print();

} else {

T value;

cout << "Input value:" << endl;

cin >> value;

tree->print(value);

}

} catch (const exception& e) {

cout << e.what() << endl;

}

}

/\*\*

\* Вывод кол-ва элементов

\*/

void countAction() {

cout << tree->count() << endl;

}

/\*\*

\* Проверка на пустоту

\*/

void emptyAction() {

cout << (tree->isEmpty() ? "Empty" : "Not empty") << endl;

}

/\*\*

\* Очистка дерева

\*/

void deleteAction() {

tree->deleteTree();

cout << "Done!" << endl;

}

};

#endif //LAB7\_MENU\_H

Файл TypeMenu.h:

/\*\*

\* Меню выбора типа

\* @file TypeMenu.h

\* @author Сааль Степан

\*/

#ifndef LAB7\_TYPEMENU\_H

#define LAB7\_TYPEMENU\_H

#include <iostream>

#include <string>

#include "Menu.h"

using namespace std;

/\*\*

\* Меню выбора типа

\*/

class TypeMenu {

public:

/\*\*

\* Показать меню

\*/

static void show() {

int type;

cout << "Choose type:" << endl;

cout << "1. int" << endl;

cout << "2. double" << endl;

cout << "3. string" << endl;

cin >> type;

showMenuForType(type);

}

private:

/\*\*

\* Создание меню работы с деревом в зависимости от типа

\* @param type

\*/

static void showMenuForType(int type) {

if (type == 1) {

auto \*menu = new Menu<int>;

menu->show();

} else if (type == 2) {

auto \*menu = new Menu<double>;

menu->show();

} else if (type == 3) {

auto \*menu = new Menu<string>;

menu->show();

} else {

cout << "Wrong number, try again:" << endl;

show();

}

}

};

#endif //LAB7\_TYPEMENU\_H

В качестве результата работы программы представлены скриншоты результата выполнения программы на рисунках 1-10:

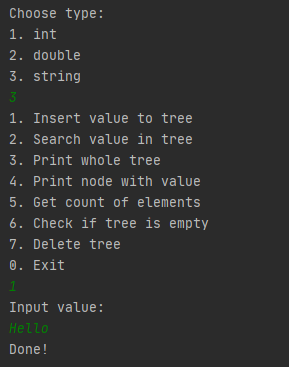


Рисунок 1

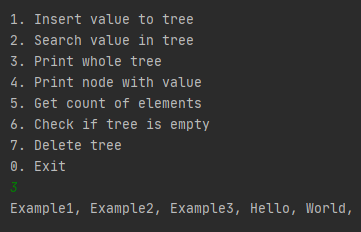


Рисунок 2

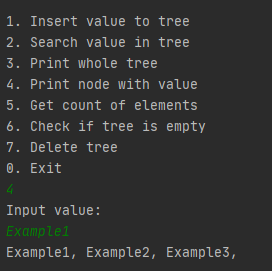


Рисунок 3

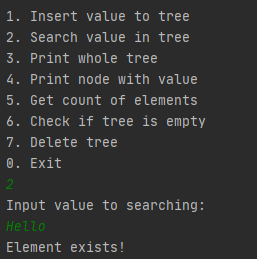


Рисунок 4

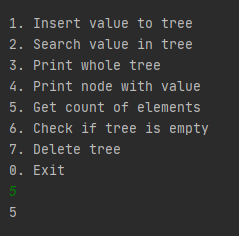


Рисунок 5

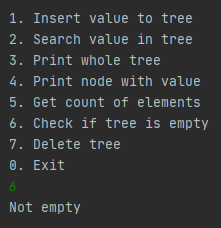


Рисунок 6

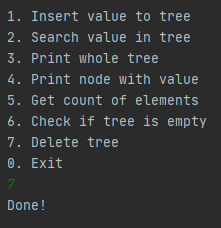


Рисунок 7

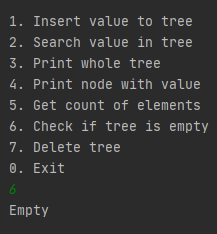


Рисунок 8

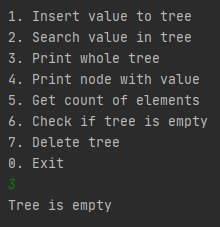


Рисунок 9

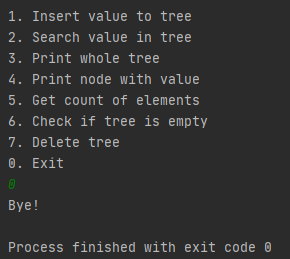


Рисунок 10

1. Диаграммы классов

Диаграмма классов представлена на рисунке 11.

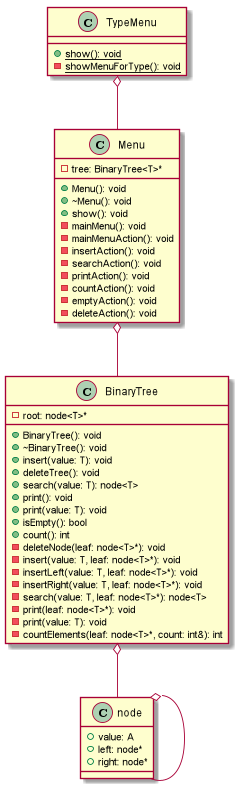


Рисунок 11

# ВЫВОД

В процессе выполнения лабораторной работы был изучен способ шаблонизирования классов.