1 ПОСТАНОВКА ЗАДАЧИ

Создать параметризованный класс бинарного дерева. С методами - добавить элемент в дерево, прохождение по дереву в нисходящем и в восходящем порядке. Осуществить поиск по дереву.

2 ДИАГРАММА КЛАССОВ

Диаграмма классов представлена на рисунке 2.1.

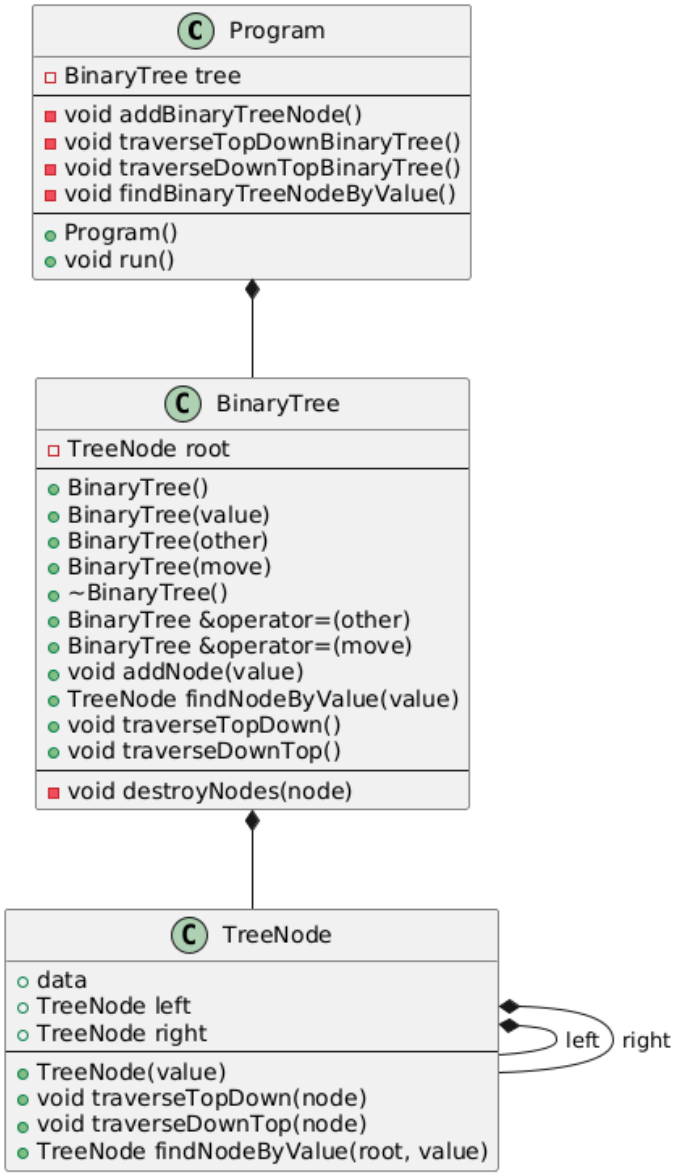

Рисунок 2.1 – Диаграмма классов

3 ЛИСТИНГ КОДА

Файл main.cpp

```
#include "program.h"

int main() {
    choiceBinaryTreeType();

    return 0;
}
```

Файл tree_node.h

```
#pragma once

#include <iostream>

template <typename T>

struct TreeNode {
    T data;
    TreeNode* left = nullptr;
    TreeNode* right = nullptr;

    explicit TreeNode(T value);
    void traverseTopDown(TreeNode<T>* node) const;
    void traverseDownTop(TreeNode<T>* node) const;
    TreeNode* findNodeByValue(TreeNode<T>* root, T value) const;
};

template <typename T>
TreeNode<T>::TreeNode(T value) : data(std::move(value)) {}

template <typename T>
void TreeNode<T>::traverseTopDown(TreeNode<T>* node) const {
    if (node == nullptr) {
        return;
    }

    std::cout << " -> " << node->data;

    traverseTopDown(node->left);
    traverseTopDown(node->right);
}

template <typename T>
void TreeNode<T>::traverseDownTop(TreeNode<T>* node) const {
    if (node == nullptr) {
        return;
    }

    traverseDownTop(node->left);
    traverseDownTop(node->right);

    std::cout << " -> " << node->data;
}

template <typename T>
TreeNode<T>* TreeNode<T>::findNodeByValue(TreeNode<T>* root, T value) const {
    if (root == nullptr) return nullptr;
```

```
    if (root->data == value) return root;

    if (auto* node = findNodeByValue(root->left, value)) {
        return node;
    }
    if (auto* node = findNodeByValue(root->right, value)) {
        return node;
    }

    return nullptr;
}
```

Файл binary_tree.h

```
#pragma once

#include <typeinfo>

#include "tree_node.h"
#include "utils.h"

template <typename T>
TreeNode<T> *copyNodes(const TreeNode<T> *node);

template <typename T>
TreeNode<T> *createNode(T value);

template <typename T>

class BinaryTree {
    TreeNode<T> *root;

    friend TreeNode<T> *copyNodes<T>(const TreeNode<T> *other);
    void destroyNodes(TreeNode<T> *node);

   public:
    BinaryTree();
    explicit BinaryTree(T value);
    BinaryTree(const BinaryTree &other);
    BinaryTree(BinaryTree &&move) noexcept;
    ~BinaryTree();

    BinaryTree &operator=(const BinaryTree &other);
    BinaryTree &operator=(BinaryTree &&move) noexcept;

    friend TreeNode<T> *createNode<T>(T value);
    void addNode(T value);
    TreeNode<T> *findNodeByValue(T value) const;
    void traverseTopDown() const;
    void traverseDownTop() const;
};

template <typename T>
BinaryTree<T>::BinaryTree() : root(nullptr) {}

template <typename T>
BinaryTree<T>::BinaryTree(T value) : root(createNode(value)) {}

template <typename T>
BinaryTree<T>::BinaryTree(const BinaryTree &other)
    : root(copyNodes(other.root)) {}

template <typename T>
```

```
BinaryTree<T>::BinaryTree(BinaryTree &&move) noexcept : root(move.root) {
    move.root = nullptr;
}

template <typename T>
BinaryTree<T> &BinaryTree<T>::operator=(const BinaryTree &other) {
    if (this != &other) {
        destroyNodes(root);
        root = copyNodes(other.root);
    }

    return *this;
}

template <typename T>
BinaryTree<T>::~BinaryTree() {
    destroyNodes(root);
    root = nullptr;
}

template <typename T>
BinaryTree<T> &BinaryTree<T>::operator=(BinaryTree &&move) noexcept {
    if (this != &move) {
        destroyNodes(root);
        root = move.root;
        move.root = nullptr;
    }

    return *this;
}

template <typename T>
TreeNode<T> *copyNodes(const TreeNode<T> *node) {
    if (node == nullptr) {
        return nullptr;
    }

    auto *newNode = new TreeNode<T>(node->data);

    newNode->left = copyNodes(node->left);
    newNode->right = copyNodes(node->right);

    return newNode;
}

template <typename T>
void BinaryTree<T>::destroyNodes(TreeNode<T> *node) {
    if (node == nullptr) {
        return;
    }

    destroyNodes(node->left);
    destroyNodes(node->right);

    delete node;
}

template <typename T>
TreeNode<T> *createNode(T value) {
    auto *node = new TreeNode<T>(value);

    node->left = nullptr;
    node->right = nullptr;
```

```
    return node;
}

template <typename T>
void BinaryTree<T>::addNode(T value) {
    if (root == nullptr) {
        root = createNode(value);
        return;
    }

    TreeNode<T> *curr = root;
    TreeNode<T> *prev = nullptr;

    while (curr != nullptr) {
        prev = curr;
        if (value < curr->data) {
            curr = curr->left;
        } else {
            curr = curr->right;
        }
    }

    if (value < prev->data) {
        prev->left = createNode(value);
    } else {
        prev->right = createNode(value);
    }
}

template <typename T>
TreeNode<T> *BinaryTree<T>::findNodeByValue(T value) const {
    return root->findNodeByValue(root, value);
}

template <typename T>
void BinaryTree<T>::traverseTopDown() const {
    root->traverseTopDown(root);
}

template <typename T>
void BinaryTree<T>::traverseDownTop() const {
    root->traverseDownTop(root);
}
```

Файл car.h

```
#pragma once

#include "cargo_carrier.h"

class Car : public CargoCarrier {
  public:
    const char *getSound() const override;

    Car();
};
```

Файл menus.h

```
#pragma once
```

```
void showTask();
void showBinaryTreeTypeMenu();
void showMenu();
```

Файл menus.cpp

```
#include "menus.h"

#include <iostream>

void showTask() {
    std::cout << "\t\t\t\tTASK" << std::endl;
    std::cout << "Create a parameterized class of a binary tree." << std::endl;
    std::cout << "Implement methods to:" << std::endl;
    std::cout << "- Add an element to the tree." << std::endl;
    std::cout << "- Traverse the tree in top-down (pre-order) order."
              << std::endl;
    std::cout << "- Traverse the tree in down-top (post-order) order."
              << std::endl;
    std::cout << "- Search for an element in the tree." << std::endl;
}

void showBinaryTreeTypeMenu() {
    std::cout << "\nSelect binary tree type:\n";
    std::cout << "1. int\n";
    std::cout << "2. long\n";
    std::cout << "3. float\n";
    std::cout << "4. double\n";
    std::cout << "5. char\n";
    std::cout << "6. string\n";
}

void showMenu() {
    std::cout << "\n\t\t\t\tMENU" << std::endl;
    std::cout << "1. Add an element to the tree." << std::endl;
    std::cout << "2. Traverse tree (top-down)." << std::endl;
    std::cout << "3. Traverse tree (down-top)." << std::endl;
    std::cout << "4. Search for an element." << std::endl;
    std::cout << "5. Exit program." << std::endl;
}
```

Файл consts.h

```
#pragma once

inline constexpr const char *kWhiteColor = "\o{33}[0m";
inline constexpr const char *kRedColor = "\o{33}[31m";
inline constexpr const char *kGreenColor = "\o{33}[32m";
```

Файл program.h

```
#pragma once

#include <iostream>

#include "binary_tree.h"
#include "consts.h"
#include "menus.h"
#include "utils.h"
```

```
template <typename T>

class Program {
    BinaryTree<T> tree;

    void addBinaryTreeNode();
    void traverseTopDownBinaryTree();
    void traverseDownTopBinaryTree();
    void findBinaryTreeNodeByValue();

   public:
    Program();
    void run();
};

template <typename T>
Program<T>::Program() {
    std::cout
        << "\nPlease create binary tree.\nEnter the binary tree node data: ";

    T value = getValue<T>();

    tree.addNode(value);
    std::cout << kGreenColor << "\nYou successfully created binary tree!"
              << kWhiteColor << std::endl;
}

template <typename T>
void Program<T>::addBinaryTreeNode() {
    std::cout << "\nPlease enter the value which you want to add to the binary
"
                 "tree: ";
    T value = getValue<T>();

    if (tree.findNodeByValue(value) != nullptr) {
        std::cout << kRedColor
                  << "\nError, this value already exists in the binary tree."
                  << kWhiteColor << std::endl;
        return;
    }

    tree.addNode(value);

    std::cout << kGreenColor << "\nYou successfully added node to binary tree!"
              << kWhiteColor << std::endl;
}

template <typename T>
void Program<T>::traverseTopDownBinaryTree() {
    std ::cout << "\nBinary tree traverse from top to down:";

    tree.traverseTopDown();

    std::cout << kGreenColor
              << "\nYou have successfully traversed the tree from top to down!"
              << kWhiteColor << std::endl;
}

template <typename T>
void Program<T>::traverseDownTopBinaryTree() {
    std ::cout << "\nBinary tree traverse from down to top:";

    tree.traverseDownTop();
```

```
    std::cout << kGreenColor
              << "\nYou have successfully traversed the tree from down to top!"
              << kWhiteColor << std::endl;
}

template <typename T>
void Program<T>::findBinaryTreeNodeByValue() {
    std::cout << "\nPlease enter the value which you want to find: ";
    T value = getValue<T>();

    if (tree.findNodeByValue(value) != nullptr) {
        std::cout << kGreenColor
                  << "\nYou have successfully finded node by value!"
                  << kWhiteColor << std::endl;
    } else {
        std::cout << kRedColor << "\nNode with this value was not found!"
                  << kWhiteColor << std::endl;
    }
}

template <typename T>
void Program<T>::run() {
    showMenu();

    int opt = 0;

    while (true) {
        std::cout << "\nPlease enter menu option: ";

        opt = getValue<int>();

        switch (opt) {
            case 1:
                addBinaryTreeNode();
                break;
            case 2:
                traverseTopDownBinaryTree();
                break;
            case 3:
                traverseDownTopBinaryTree();
                break;
            case 4:
                findBinaryTreeNodeByValue();
                break;
            case 5:
                std::cout << kGreenColor
                          << "\nYou have successfully exited the program."
                          << kWhiteColor << std::endl;
                return;
            default:
                std::cout << kRedColor
                          << "\nError, you picked is an incorrect menu option.
"
                             "Please try again."
                          << kWhiteColor << std::endl;
        }
    }
}
```

Файл utils.h

```
#pragma once
```

```
#include <iostream>

#include "consts.h"

template <class T>

T getValue() {
    T value;
    int sym = 0;

    while (true) {
        if (std::cin.peek() != '\n' && std::cin.peek() != ' ' &&
            (std::cin >> value).good()) {
            sym = std::cin.peek();
            if ((char)sym == '\n' || (char)sym == EOF) {
                std::cin.get();
                return value;
            }
        }

        std::cin.clear();
        while (std::cin.get() != '\n' && !std::cin.eof());
        std::cout << kRedColor << "\nError, invalid input. Please try again: "
                  << kWhiteColor;
    }
}

void choiceBinaryTreeType();
```

Файл utils.cpp

```
#include "utils.h"

#include "program.h"

void choiceBinaryTreeType() {
    int choice = 0;

    system("clear");
    showTask();
    showBinaryTreeTypeMenu();

    while (true) {
        std::cout << "\nPlease choose a binary tree type: ";

        choice = getValue<int>();

        switch (choice) {
            case 1: {
                Program<int> program;
                program.run();
                return;
            }
            case 2: {
                Program<long> program;
                program.run();
                return;
            }
            case 3: {
                Program<float> program;
                program.run();
                return;
```

```
            }
            case 4: {
                Program<double> program;
                program.run();
                return;
            }
            case 5: {
                Program<char> program;
                program.run();
                return;
            }
            case 6: {
                Program<std::string> program;
                program.run();
                return;
            }
            default:
                std::cout << kRedColor
                          << "\nError, invalid choice. Please try again."
                          << kWhiteColor << std::endl;
        }
    }
}
```

4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

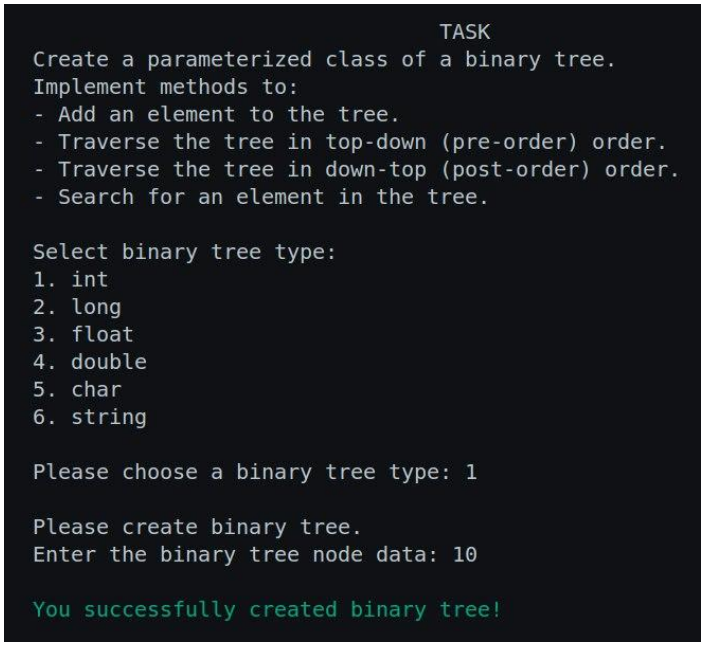

Рисунок 4.1 – Описание задания, выбор типа данных и создание корневого узла бинарного дерева

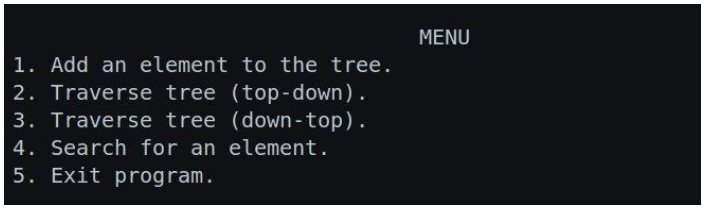

Рисунок 4.2 – Главное меню задание

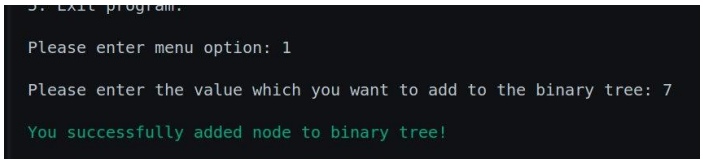

Рисунок 4.3 – Добавление узла бинарного дерева

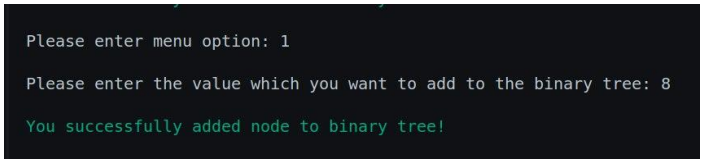

Рисунок 4.4 – Повторное добавление узла бинарного дерева

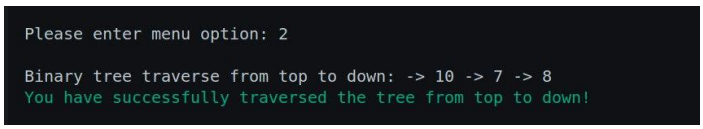

Рисунок 4.5 – Обход дерева в нисходящем порядке

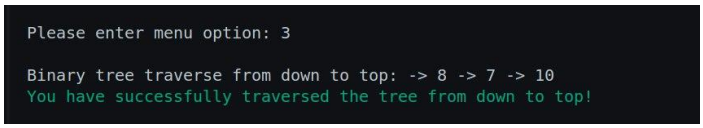

Рисунок 4.6 – Обход дерева в восходящем порядке

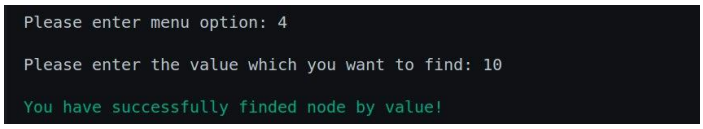

Рисунок 4.7 – Успешный поиск узла бинарного дерева по значению

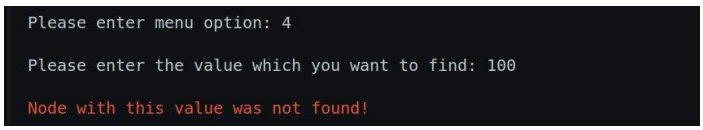

Рисунок 4.8 – Отрицательный результат поиска узла бинарного дерева по значению

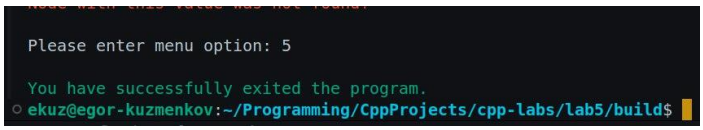

Рисунок 4.9 – Завершение программы

5 ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были закреплены навыки разработки параметризованных классов на языке C++. Был реализован обобщённый класс бинарного дерева, поддерживающий добавление элементов, обход дерева в нисходящем (прямом) и восходящем (обратном) порядке, а также поиск узлов по значению. В процессе работы были реализованы конструкторы, деструктор и необходимые методы для корректного управления памятью и структуры дерева. Проведённое тестирование подтвердило правильность работы алгоритмов добавления, поиска и обхода, а также корректность взаимодействия между узлами и корнем бинарного дерева.