МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания №7.1

Тема: Балансировка дерева поиска

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Зуев Д.А.
Группа ИКБО-68-23

Москва 2025

Цель работы: освоить приёмы работы с двоичным деревом

ЗАДАНИЕ

Формулировка задачи

Составить программу создания двоичного дерева поиска и реализовать процедуры для работы с деревом согласно варианту.

Процедуры оформить в виде самостоятельных режимов работы созданного дерева. Выбор режимов производить с помощью пользовательского (иерархического ниспадающего) меню. Провести полное тестирование программы на дереве размером n=10 элементов, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Сделать выводы о проделанной работе, основанные на полученных результатах.

Оформить отчет с подробным описанием созданного дерева, принципов программной реализации алгоритмов работы с деревом, описанием текста исходного кода и проведенного тестирования программы.

Персональный вариант:

№	Тип значения узла	Тип дерева	Реализовать алгоритмы
30	Число	АВЛ - дерево	Вставка элемента и балансировка, обратный обход, симметричный обход, поиск суммы значений листьев, поиск среднего арифметического всех узлов

Описание структуры данных

AVL-дерево — это структура данных, в которой каждый узел содержит значение, ссылки на левого и правого потомков, а также высоту узла. Оно поддерживает балансировку, чтобы высота левого и правого поддеревьев любого узла отличалась не более чем на 1.

Основные операции

Вставка: Добавление нового элемента в дерево с последующей балансировкой.

Поиск: Нахождение элемента в дереве.

Обходы:

- Симметричный (in-order)
- Обратный (post-order)

Сумма значений листьев: Подсчет суммы в листьях.

Среднее арифметическое всех узлов: Вычисление среднего значения чисел всех узлов.

Дополнительно добавлена функция вывода дерева для наглядности и проверки.

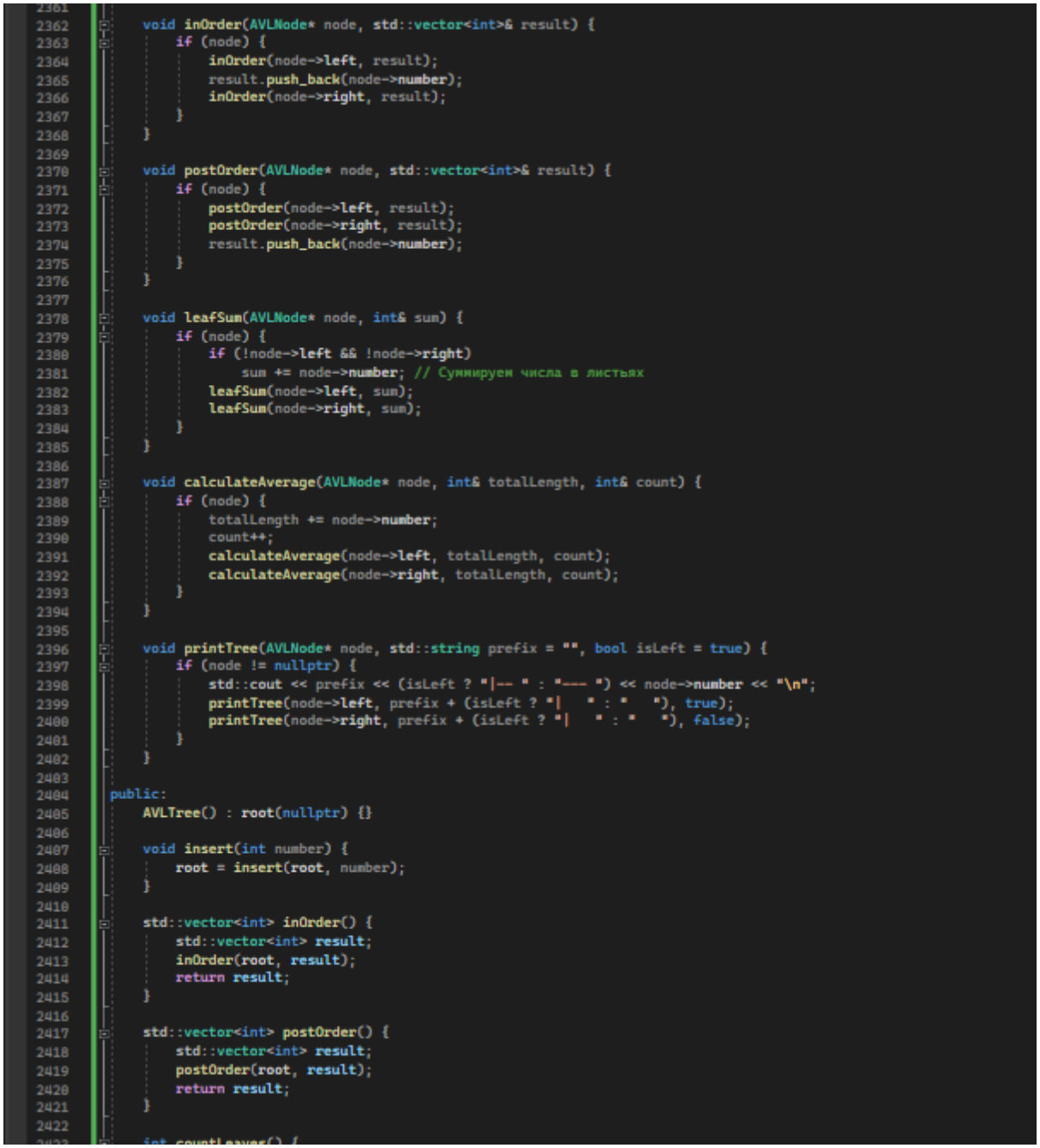
Код программы

```
2276  #include <iostream>
2277  #include <string>
2278  #include <vector>
2279
2280
2281  class AVLNode {
2282  public:
2283      int number;
2284      AVLNode* left;
2285      AVLNode* right;
2286      int height;
2287
2288      AVLNode(int name) : number(name), left(nullptr), right(nullptr), height(1) {}
2289  };
2290
2291  class AVLTree {
2292  private:
2293      AVLNode* root;
2294
2295      int getHeight(AVLNode* node) {
2296          return node ? node->height : 0;
2297      }
2298
2299      int getBalance(AVLNode* node) {
2300          return node ? getHeight(node->left) - getHeight(node->right) : 0;
2301      }
2302
2303      AVLNode* rightRotate(AVLNode* y) {
2304          AVLNode* x = y->left;
2305          AVLNode* T2 = x->right;
2306
2307          x->right = y;
2308          y->left = T2;
2309
2310          y->height = std::max(getHeight(y->left), getHeight(y->right)) + 1;
2311          x->height = std::max(getHeight(x->left), getHeight(x->right)) + 1;
2312
2313          return x;
2314      }
```

Рисунок 1 – реализация программы (часть 1)

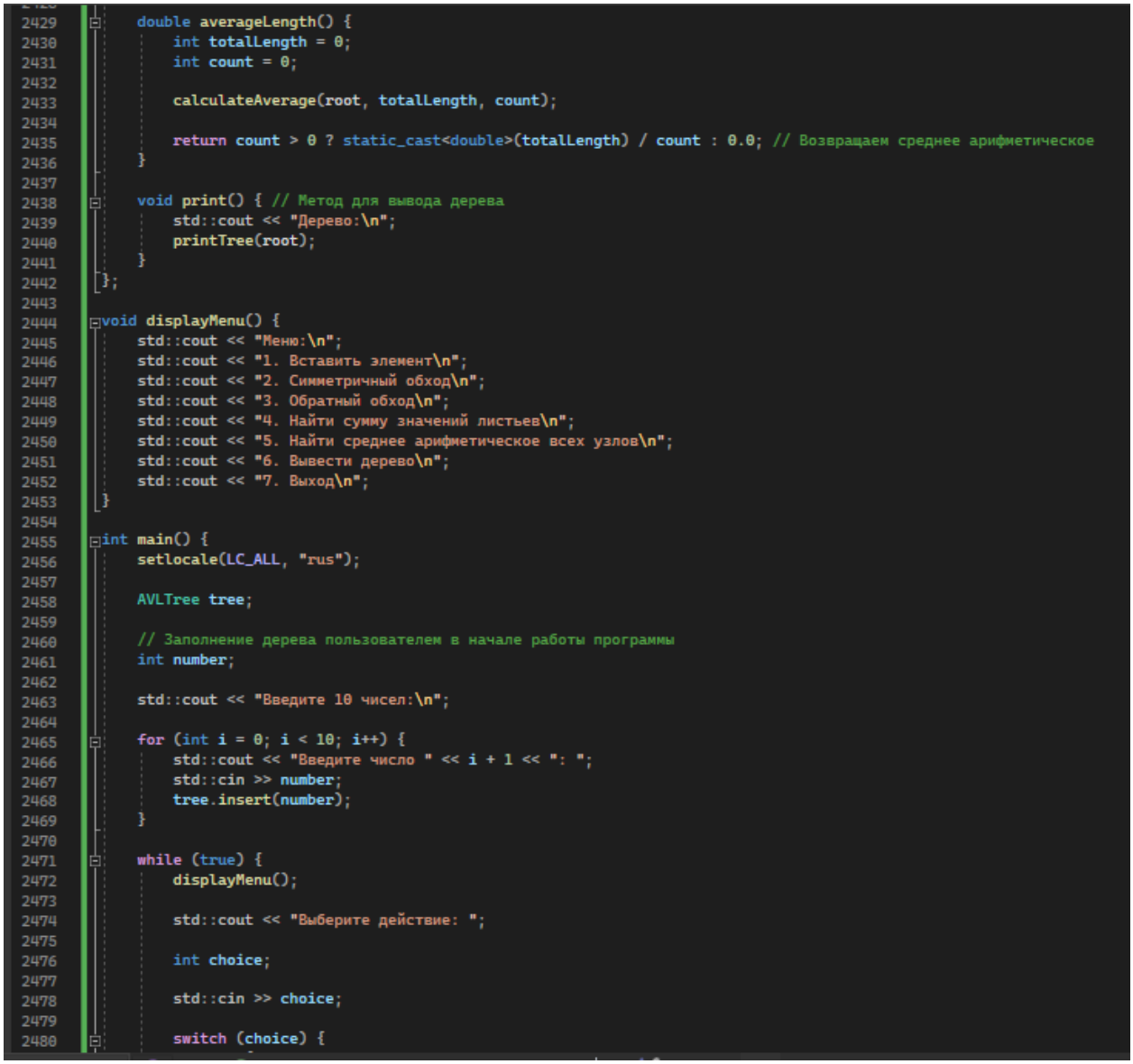
```
2316    AVLNode* leftRotate(AVLNode* x) {
2317        AVLNode* y = x->right;
2318        AVLNode* T2 = y->left;
2319
2320        y->left = x;
2321        x->right = T2;
2322
2323        x->height = std::max(getHeight(x->left), getHeight(x->right)) + 1;
2324        y->height = std::max(getHeight(y->left), getHeight(y->right)) + 1;
2325
2326        return y;
2327    }
2328
2329    AVLNode* insert(AVLNode* node, int name) {
2330        if (!node) return new AVLNode(name);
2331
2332        if (name < node->number)
2333            node->left = insert(node->left, name);
2334        else if (name > node->number)
2335            node->right = insert(node->right, name);
2336        else
2337            return node; // Duplicate names are not allowed
2338
2339        node->height = 1 + std::max(getHeight(node->left), getHeight(node->right));
2340
2341        int balance = getBalance(node);
2342
2343        if (balance > 1 && name < node->left->number)
2344            return rightRotate(node);
2345
2346        if (balance < -1 && name > node->right->number)
2347            return leftRotate(node);
2348
2349        if (balance > 1 && name > node->left->number) {
2350            node->left = leftRotate(node->left);
2351            return rightRotate(node);
2352        }
2353
2354        if (balance < -1 && name < node->right->number) {
2355            node->right = rightRotate(node->right);
2356            return leftRotate(node);
2357        }
2358
2359        return node;
2360    }
2361
```

Рисунок 2 – реализация программы (часть 2)



```
    void inOrder(AVLNode* node, std::vector<int>& result) {
        if (node) {
            inOrder(node->left, result);
            result.push_back(node->number);
            inOrder(node->right, result);
        }
    }

    void postOrder(AVLNode* node, std::vector<int>& result) {
        if (node) {
            postOrder(node->left, result);
            postOrder(node->right, result);
            result.push_back(node->number);
        }
    }

    void leafSum(AVLNode* node, int& sum) {
        if (node) {
            if (!node->left && !node->right)
                sum += node->number; // Суммируем числа в листьях
            leafSum(node->left, sum);
            leafSum(node->right, sum);
        }
    }

    void calculateAverage(AVLNode* node, int& totalLength, int& count) {
        if (node) {
            totalLength += node->number;
            count++;
            calculateAverage(node->left, totalLength, count);
            calculateAverage(node->right, totalLength, count);
        }
    }

    void printTree(AVLNode* node, std::string prefix = "", bool isLeft = true) {
        if (node != nullptr) {
            std::cout << prefix << (isLeft ? "|-- " : "--- ") << node->number << "\n";
            printTree(node->left, prefix + (isLeft ? "|   " : "    "), true);
            printTree(node->right, prefix + (isLeft ? "|   " : "    "), false);
        }
    }

public:
    AVLTree() : root(nullptr) {}

    void insert(int number) {
        root = insert(root, number);
    }

    std::vector<int> inOrder() {
        std::vector<int> result;
        inOrder(root, result);
        return result;
    }

    std::vector<int> postOrder() {
        std::vector<int> result;
        postOrder(root, result);
        return result;
    }

    int countLeaves() {
```

Рисунок 3 – реализация программы (часть 3)



```
2429    double averageLength() {
2430        int totalLength = 0;
2431        int count = 0;
2432
2433        calculateAverage(root, totalLength, count);
2434
2435        return count > 0 ? static_cast<double>(totalLength) / count : 0.0; // Возвращаем среднее арифметическое
2436    }
2437
2438    void print() { // Метод для вывода дерева
2439        std::cout << "Дерево:\n";
2440        printTree(root);
2441    }
2442 };
2443
2444 void displayMenu() {
2445     std::cout << "Меню:\n";
2446     std::cout << "1. Вставить элемент\n";
2447     std::cout << "2. Симметричный обход\n";
2448     std::cout << "3. Обратный обход\n";
2449     std::cout << "4. Найти сумму значений листьев\n";
2450     std::cout << "5. Найти среднее арифметическое всех узлов\n";
2451     std::cout << "6. Вывести дерево\n";
2452     std::cout << "7. Выход\n";
2453 }
2454
2455 int main() {
2456     setlocale(LC_ALL, "rus");
2457
2458     AVLTree tree;
2459
2460     // Заполнение дерева пользователем в начале работы программы
2461     int number;
2462
2463     std::cout << "Введите 10 чисел:\n";
2464
2465     for (int i = 0; i < 10; i++) {
2466         std::cout << "Введите число " << i + 1 << ": ";
2467         std::cin >> number;
2468         tree.insert(number);
2469     }
2470
2471     while (true) {
2472         displayMenu();
2473
2474         std::cout << "Выберите действие: ";
2475
2476         int choice;
2477
2478         std::cin >> choice;
2479
2480         switch (choice) {
```

Рисунок 4 – реализация программы (часть 4)

```
            std::cout << "Симметричный обход: ";
            for (const auto& number : numbers)
                std::cout << number << " ";
            std::cout << "\n";
            break;
        }
        case 3: {
            auto numbers = tree.postOrder();
            std::cout << "Обратный обход: ";
            for (const auto& number : numbers)
                std::cout << number << " ";
            std::cout << "\n";
            break;
        }
        case 4: {
            int leavesCount = tree.countLeaves();
            std::cout << "Сумма чисел в листьях: " << leavesCount << "\n";
            break;
        }
        case 5: {
            double averageLen = tree.averageLength();
            std::cout << "Среднее арифметическое всех узлов: " << averageLen << "\n";
            break;
        }
        case 6: { // Вывод дерева
            tree.print();
            break;
        }
        case 7:
            return 0; // Exit
        default:
            std::cout << "Неверный выбор. Попробуйте снова.\n";
            break;
        }
        std::cout << "\n";
    }
```

Рисунок 5 – реализация основной функции программы

Результаты тестирования

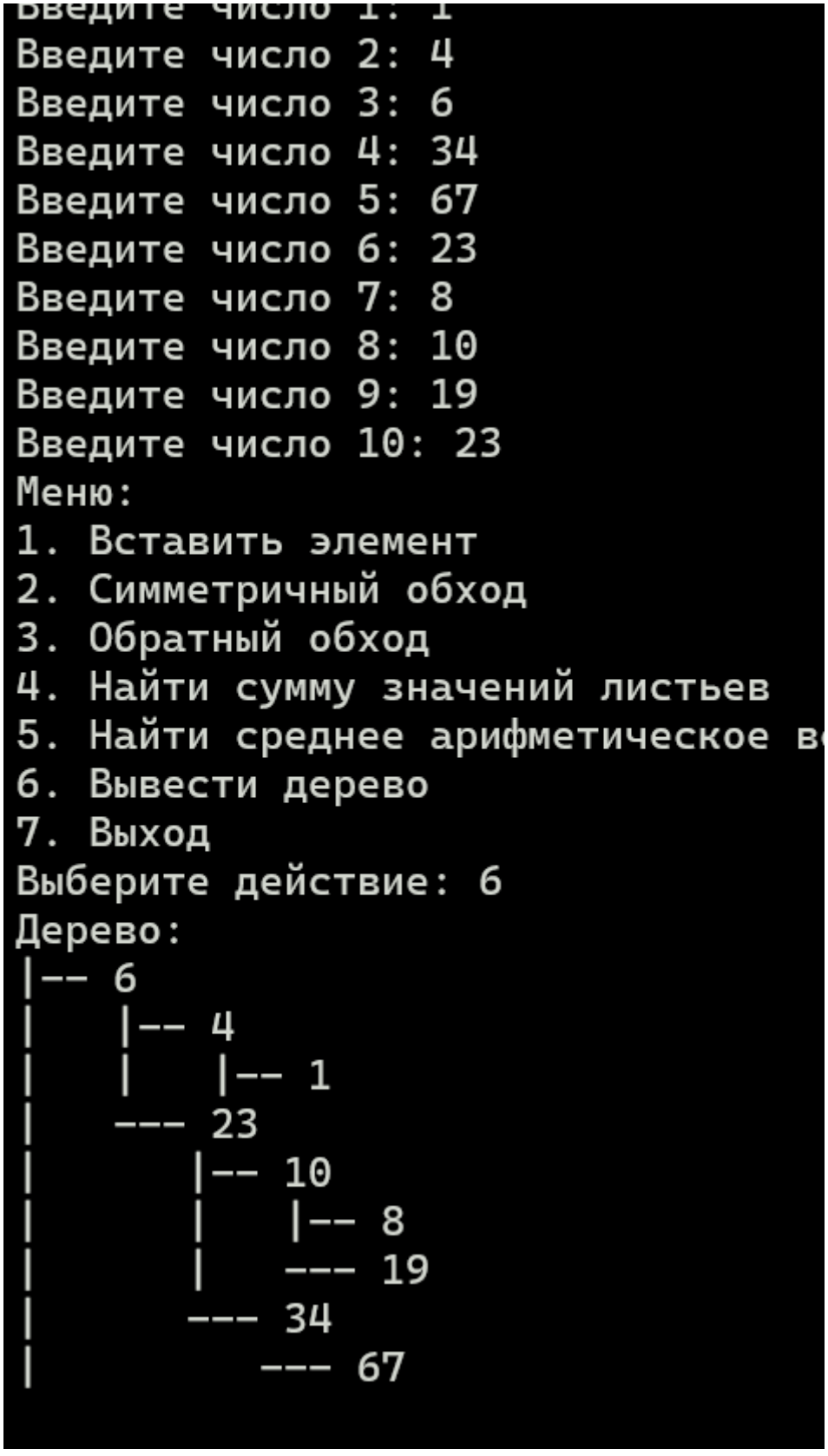


Рисунок 6 – ввод и вывод дерева

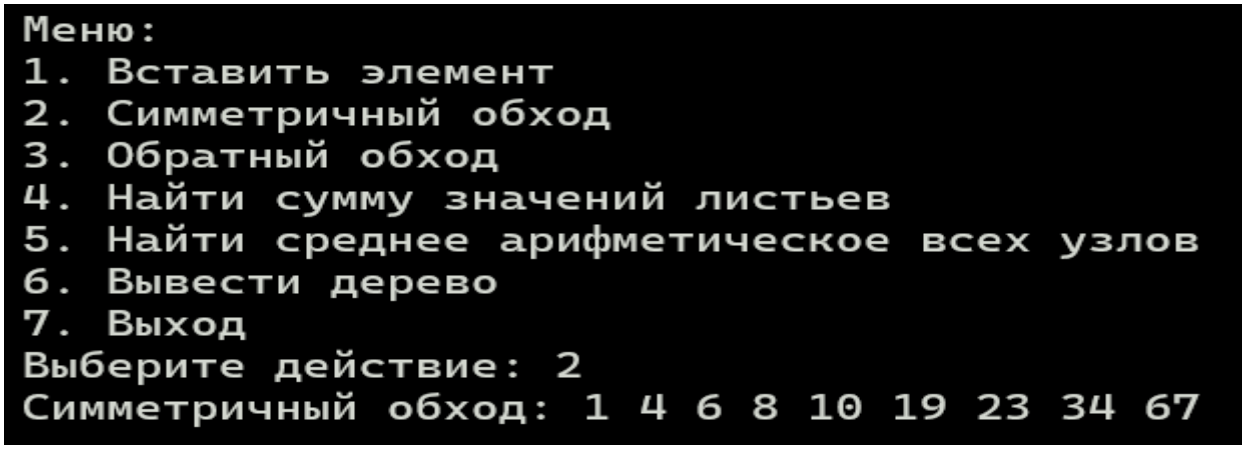


Рисунок 8 – тестирование симметричного обхода дерева

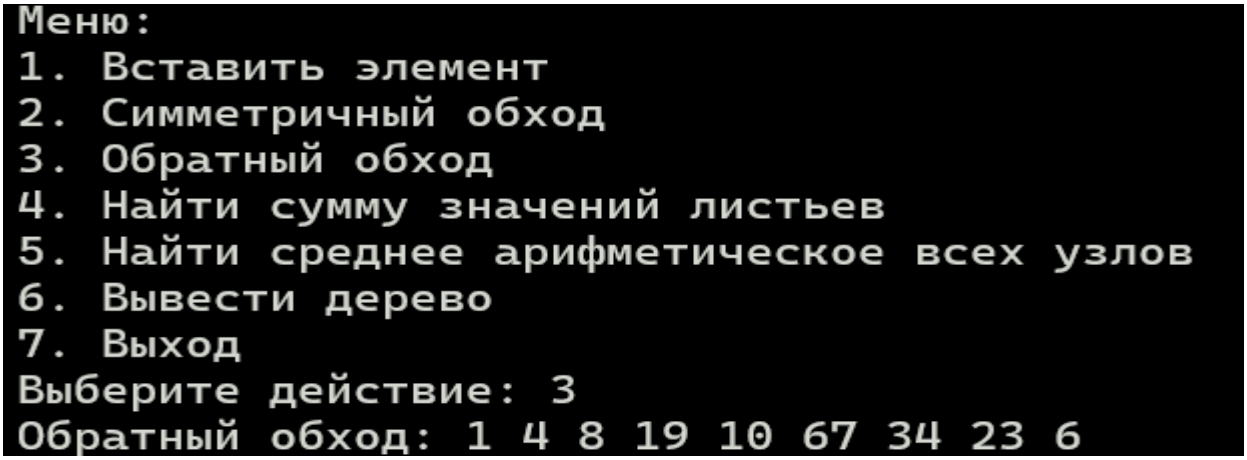


Рисунок 10 – тестирование обратного обхода дерева

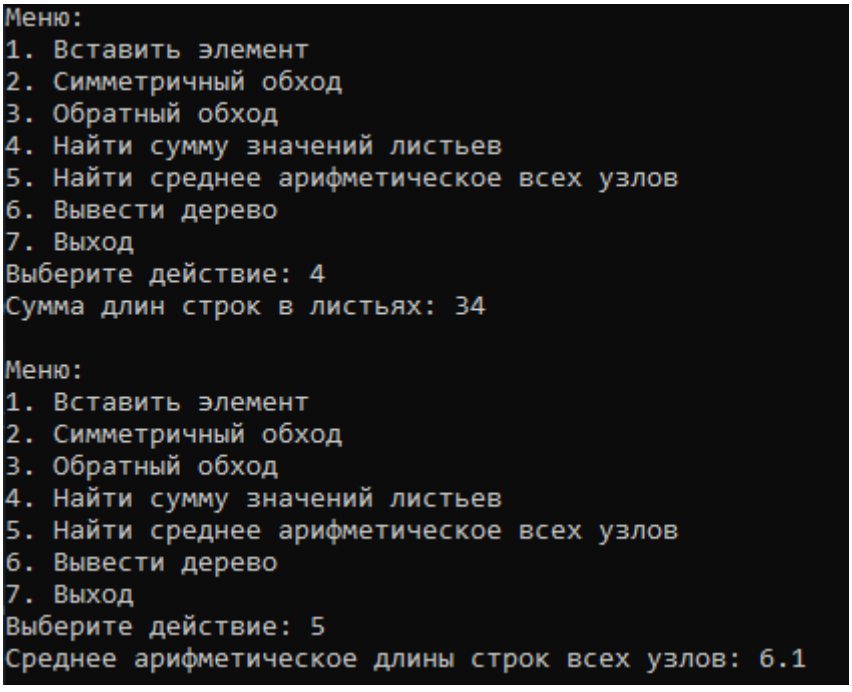


Рисунок 12 – тестирование функций среднего арифметического и суммы значений листьев

ВЫВОД

В ходе выполнения работы была успешно реализована программа для создания AVL-дерева с возможностью вставки элементов и выполнения различных обходов. Программа прошла тестирование на наборе из 10 элементов, результаты которого подтвердили корректность работы всех реализованных алгоритмов.

Данная работа позволила углубить понимание структуры двоичного АВЛ-дерева. Программа может быть расширена дополнительными функциями, такими как удаление узлов и более сложные операции с деревом.

Таким образом, работа над проектом позволила не только реализовать функционал AVL-дерева, но и получить практические навыки работы с важными структурами данных в программировании.