Практическая работа № 6

БИНАРНОЕ ДЕРЕВО ПОИСКА

Вариант 22

Постановка 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.

1. Описание алгоритма

Бинарное дерево поиска — это бинарное дерево, обладающее дополнительными свойствами: значение левого потомка меньше значения родителя, а значение правого потомка больше значения родителя для каждого узла дерева. То есть, данные в бинарном дереве поиска хранятся в отсортированном виде. При каждой операции вставки нового или удаления существующего узла отсортированный порядок дерева сохраняется. При поиске элемента сравнивается искомое значение с корнем. Если искомое больше корня, то поиск продолжается в правом потомке корня, если меньше, то в левом, если равно, то значение найдено и поиск прекращается.

В данной практической реализован алгоритм двоичного дерева на языке C++. Распределение строк в дереве осуществляется путем их сравнения. Реализация прямого и обратного обхода выполнена рекурсивно.

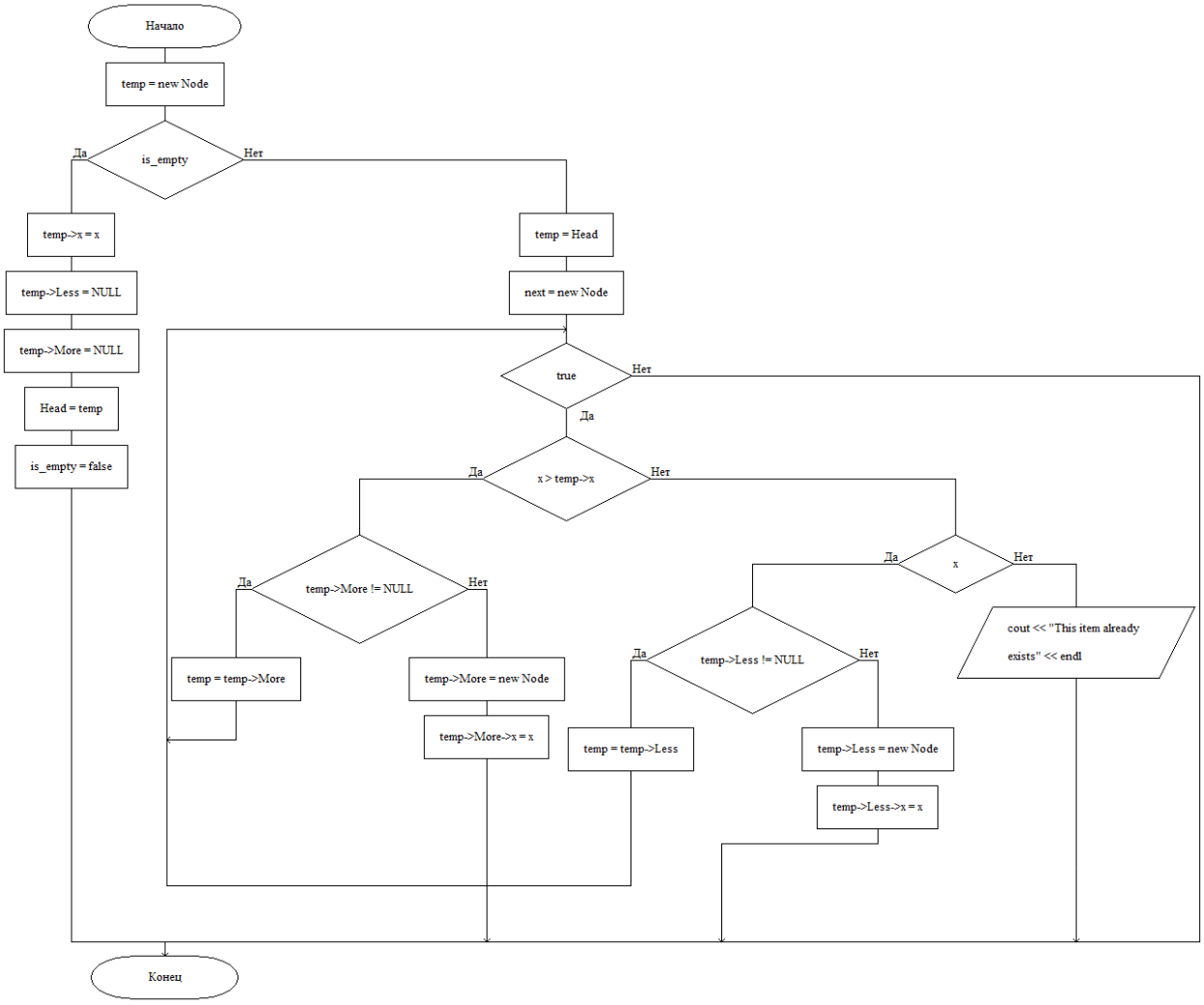

Рис.1 Схема алгоритма функции add

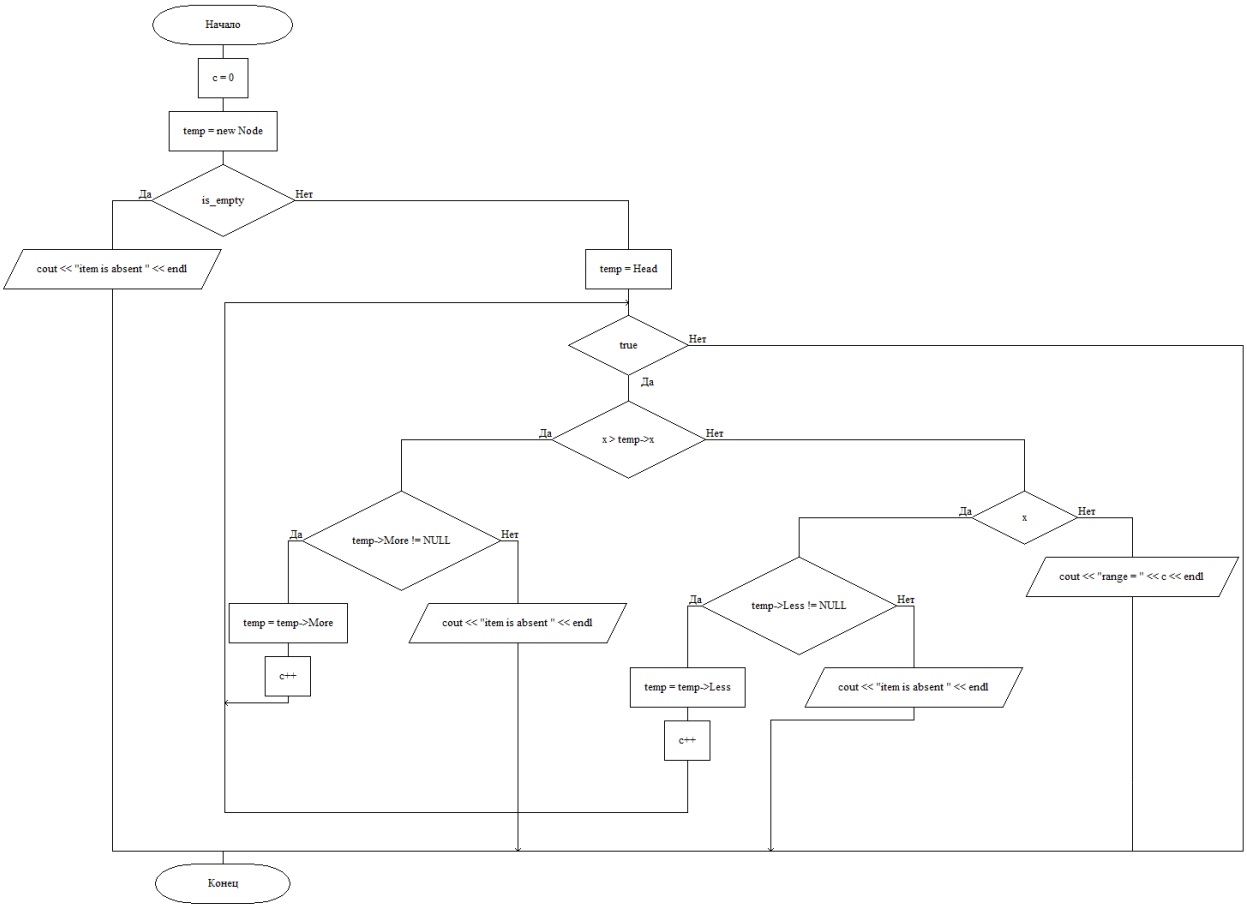

Рис.2 Схема алгоритма функции chek

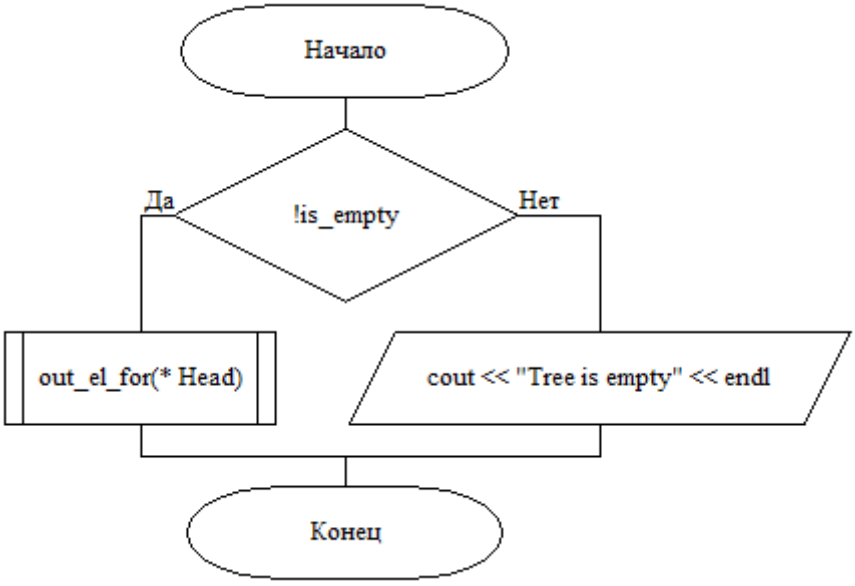

Рис.3 Схема алгоритма функции forward

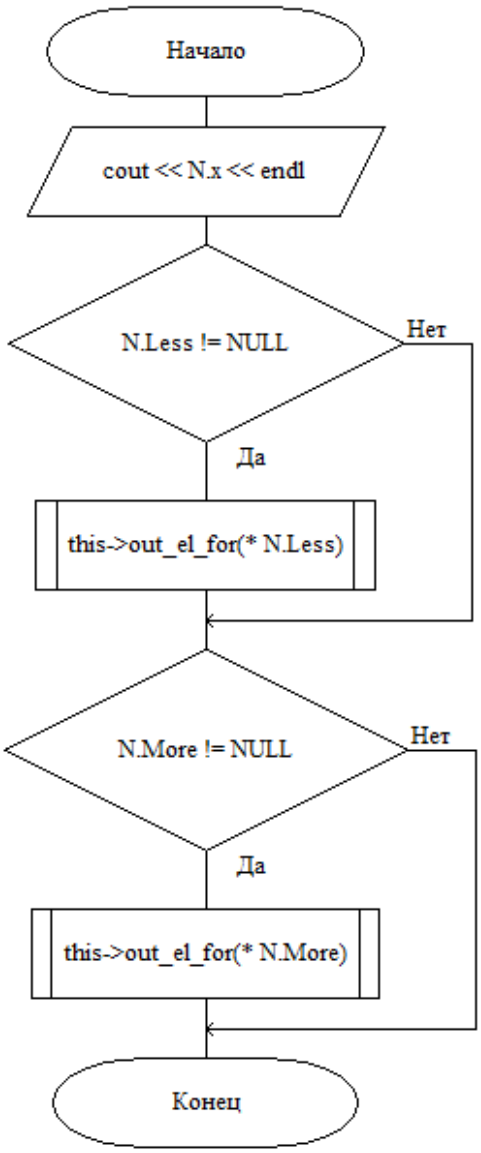

Рис.4 Схема алгоритма функции out_el_for

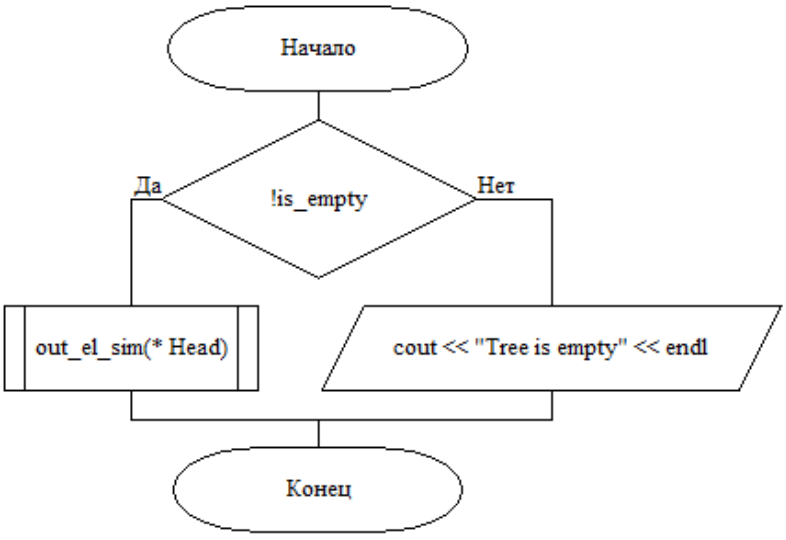

Рис.5 Схема алгоритма функции simetr

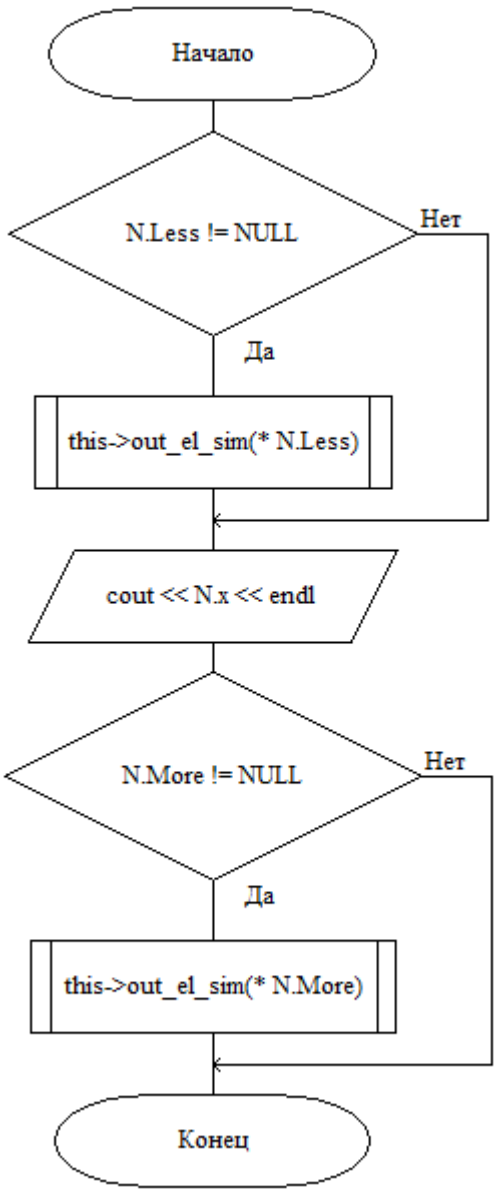

Рис.6 Схема алгоритма функции out_el_sim

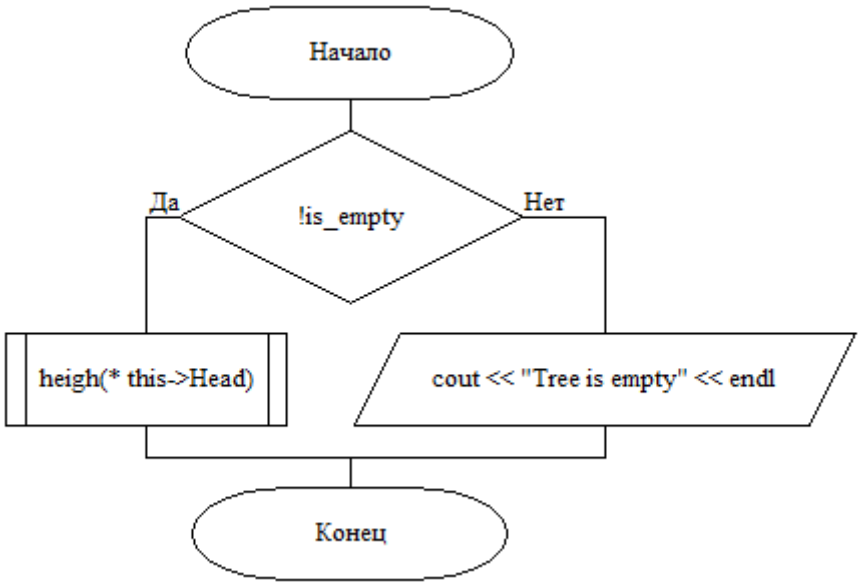

Рис.7 Схема алгоритма функции out_heigh

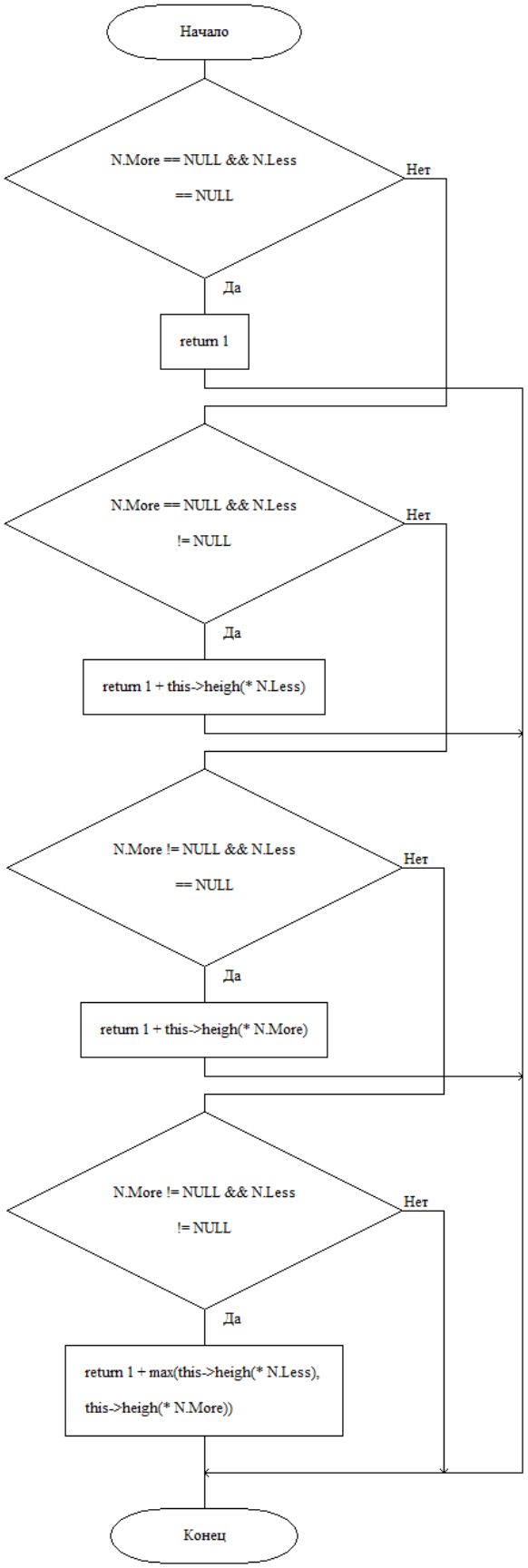

Рис.8 Схема алгоритма функции heigh

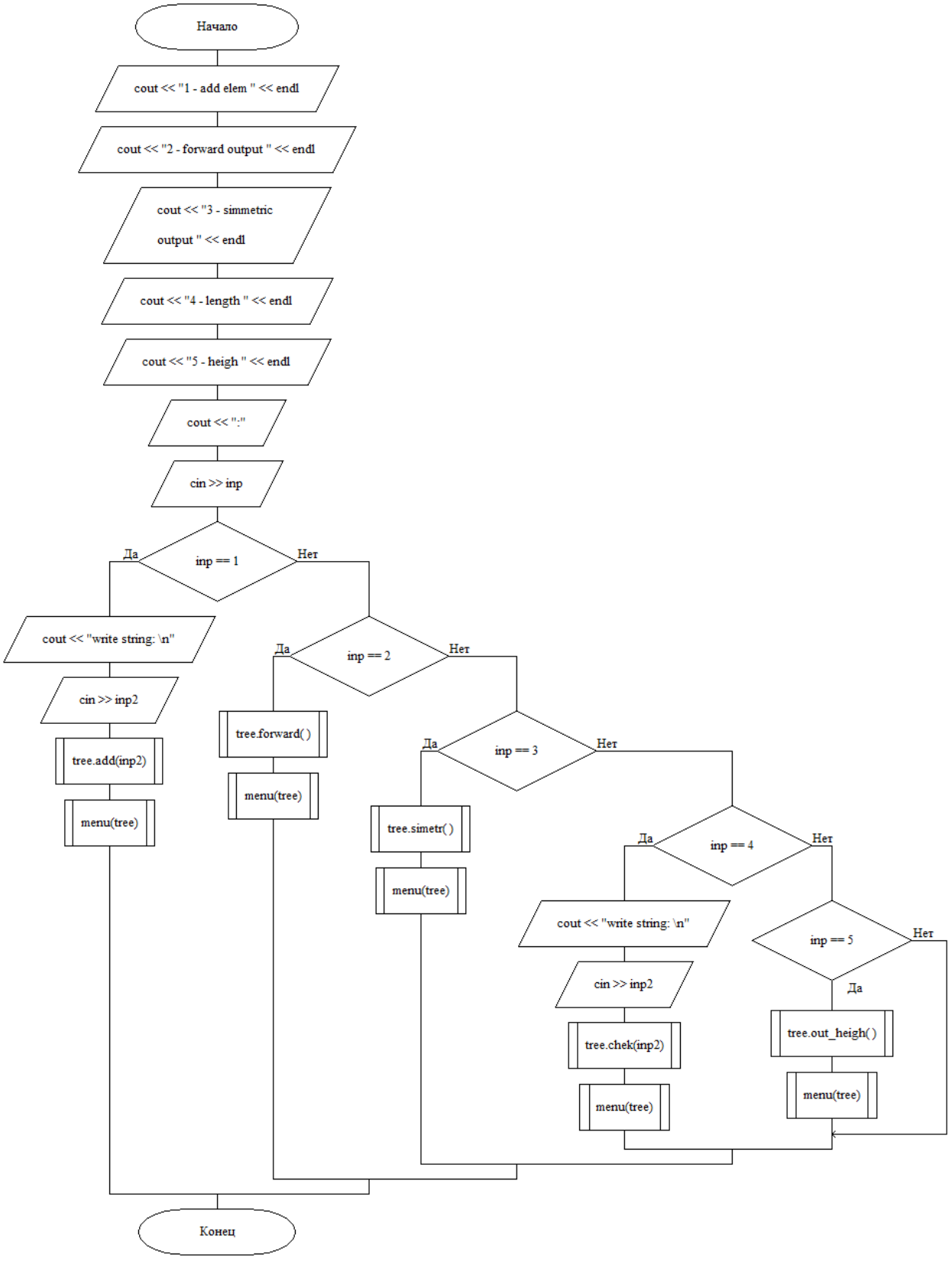

Рис.9 Схема алгоритма функции menu

2. Реализация алгоритма

Текст исходного кода программы

```
#include <iostream>

using namespace std;

struct Node {
    string x;
    Node *More = NULL;
    Node *Less = NULL;
};

class Tree {
    Node *More, *Less;

public:
    bool is_empty;
    Node *Head = NULL;

    Tree() : More(NULL), Less(NULL) { is_empty = true; };

    void add(string x);

    void forward();

    void simetr();

    void out_el_for(Node N);

    void out_el_sim(Node N);

    void chek(string x);

    int heigh(Node N);

    void out_heigh();

};

void Tree::add(string x) {
    Node *temp = new Node;
    if (is_empty) {
        temp->x = x;
        temp->Less = NULL;
        temp->More = NULL;
        Head = temp;
        is_empty = false;

    } else {
        temp = Head;
```

```
        Node *next = new Node;
        while (true) {
            if (x > temp->x) {
                if (temp->More != NULL) {
                    temp = temp->More;

                } else {
                    temp->More = new Node;
                    temp->More->x = x;
                    break;
                }
            } else if (x < temp->x) {
                if (temp->Less != NULL) {
                    temp = temp->Less;

                } else {
                    temp->Less = new Node;
                    temp->Less->x = x;
                    break;
                }
            } else {
                cout << "This item already exists" << endl;
                break;
            }
        }
    }
}

void Tree::chek(string x) {
    int c = 0;
    Node *temp = new Node;
    if (is_empty) {
        cout << "item is absent " << endl;

    } else {
        temp = Head;

        while (true) {
            if (x > temp->x) {
                if (temp->More != NULL) {
                    temp = temp->More;
                    c++;

                } else {


                    cout << "item is absent " << endl;
                    break;
                }
            } else if (x < temp->x) {
                if (temp->Less != NULL) {
                    temp = temp->Less;
                    c++;

                } else {

                    cout << "item is absent " << endl;
                    break;
                }
            } else {
                cout << "range = " << c << endl;
                break;
            }
        }
```

```
    }
}


void Tree::forward() {
    if (!is_empty)out_el_for(*Head);
    else cout << "Tree is empty" << endl;
}

void Tree::out_el_for(Node N) {
    cout << N.x << endl;
    if (N.Less != NULL) {
        this->out_el_for(*N.Less);
    }
    if (N.More != NULL) {
        this->out_el_for(*N.More);
    }

}

void Tree::simetr() {

    if (!is_empty)out_el_sim(*Head);
    else cout << "Tree is empty" << endl;
}

void Tree::out_el_sim(Node N) {

    if (N.Less != NULL) {
        this->out_el_sim(*N.Less);
    }
    cout << N.x << endl;
    if (N.More != NULL) {
        this->out_el_sim(*N.More);
    }

}

int max(int a, int b) {
    if (a >= b) return a;
    else return b;
}

void Tree::out_heigh() {
    if (!is_empty) cout << "heigh = " << this->heigh(*this->Head) << endl;
    else cout << "Tree is empty" << endl;

}


int Tree::heigh(Node N) {

    if (N.More == NULL && N.Less == NULL) return 1;
    if (N.More == NULL && N.Less != NULL) return 1 + this->heigh(*N.Less);
    if (N.More != NULL && N.Less == NULL) return 1 + this->heigh(*N.More);
    if (N.More != NULL && N.Less != NULL) return 1 + max(this-
>heigh(*N.Less), this->heigh(*N.More));


}

void menu(Tree tree) {
    cout << "1 - add elem " << endl;
```

```
    cout << "2 - forward output " << endl;
    cout << "3 - simmetric output " << endl;
    cout << "4 - length " << endl;
    cout << "5 - heigh " << endl;
    cout << ":";

    int inp;
    cin >> inp;

    if (inp == 1) {
        cout << "write string: \n";
        string inp2;
        cin >> inp2;
        tree.add(inp2);
        menu(tree);
    } else if (inp == 2) {
        tree.forward();
        menu(tree);

    } else if (inp == 3) {
        tree.simetr();
        menu(tree);
    } else if (inp == 4) {
        cout << "write string: \n";
        string inp2;
        cin >> inp2;
        tree.chek(inp2);
        menu(tree);
    } else if (inp == 5) {
        tree.out_heigh();
        menu(tree);
    }
}

int main() {

    Tree a;
    menu(a);

}
```

3. Тестирование программы

```
1 - add elem
2 - forward output
3 - simmetric output
4 - length
5 - heigh
:
```

Рис.10 Скриншот интерфейса программы

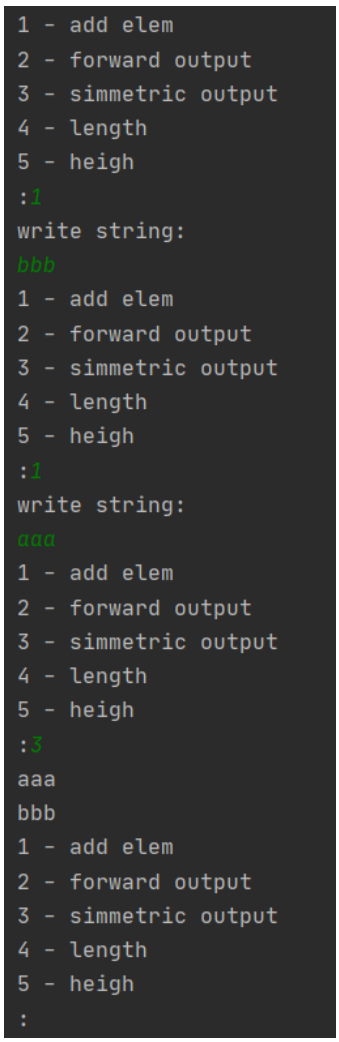

Рис.11 Скриншот добавления 2-х элементов и симметричного вывода

Выводы

1. В ходе работы было создано бинарное дерево поиска, ссылки между элементами которого осуществляются при помощи указателей на объекты класса Node.
2. Также были реализованы функции работы с деревьями: добавление элементов, прямой и симметричный выводы, вывод длинны и высоты дерева.

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 03.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 03.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 03.11.2020).
5. Теоретические сведения о бинарных деревьях поиска, интернет-ресурс: https://habr.com/ru/post/267855 (Дата обращения – 03.11.2020).