|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 2** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Бинарное дерево»** | |
|  | |
| Выполнил студент группы ИКБО-02 -19 | Миронов А.Д. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_ 201\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_201\_\_ г. | *(подпись руководителя)* |

Москва 2020

# **Цель работы**

Освоение реализации Бинарных деревьев.

# **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.

# **Алгоритм решения**

Алгоритм

# **Исходный код 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);

}

**Вариант №22**.

**Решение:**



Рис. 1. Интерфейс программы

**Тестирование**

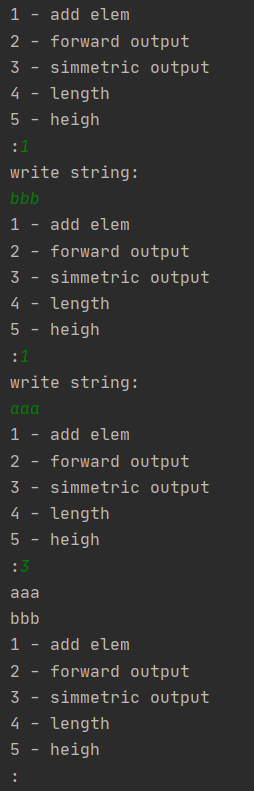


Рис. 2. Результаты контрольного тестирования

На данном примере показано добавление 2-х элементов и их вывод с помощью симметричного обхода

# **Выводы**

В результате выполнения работы я:

1. Освоил алгоритм бинарного дерева и его реализацию на языке программирования C++.

# **Информационные источники**

1. Язык программирования C++
2. Методические указания
3. Drow