Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №4  
по дисциплине: «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Бинарное дерево поиска»

Выполнили студенты группы 22ВВВ2:  
Беляев Д.

Приняли:  
Акифьев И. В.

Митрохин М. А.

Пенза 2023

**Название**

Бинарное дерево поиска

**Цель работы**

Разработать алгоритм бинарного дерева поиска. Научиться работать с данным типом данных

**Лабораторное задание**

1. Реализовать алгоритм поиска вводимого с клавиатуры значения в уже созданном дереве.
2. Реализовать функцию подсчёта числа вхождений заданного элемента в дерево.
3. \* Изменить функцию добавления элементов для исключения добавления одинаковых символов.
4. \* Оценить сложность процедуры поиска по значению в бинарном дереве.

**Листинг**

* **main.c**

#include <stdio.h>

#include "binTree.h"

int main()

{

Tree\* root = CreateTree(50);

root = AddNode(10, root);

root = AddNode(12, root);

root = AddNode(11, root);

root = AddNode(20, root);

root = AddNode(19, root);

root = AddNode(25, root);

root = AddNode(24, root);

root = AddNode(100, root);

root = AddNode(60, root);

root = AddNode(110, root);

TreePrint(root, 0);

printf("---- Finded node ----\n");

Tree\* find = SearchNode(100, root);

PrintOneNode(find);

printf("---------------------\n");

printf("Entry count of %d: %d\n", 19, EntryCount(19, root, 0));

ClearTree(root);

return 0;

}

* **binTree.h**

#ifndef BIN\_TREE

#define BIN\_TREE

typedef struct Node {

int value;

struct Node\* left;

struct Node\* right;

} Tree;

Tree\* CreateTree(int val);

Tree\* AddNode(int val, Tree\* root);

Tree\* SearchNode(int val, Tree\* root);

int EntryCount(int value, Tree\* root, int entry);

void TreePrint(Tree\* root, int tabs);

void ClearTree(Tree\* root);

void PrintOneNode(Tree\* root);

#endif

* **binTree.c**

#include <stdio.h>

#include <stdlib.h>

#include "binTree.h"

Tree\* CreateNode(int value)

{

Tree\* tmp = (Tree\*)malloc(sizeof(Tree));

tmp->value = value;

tmp->left = NULL;

tmp->right = NULL;

}

Tree\* CreateTree(int val)

{

return CreateNode(val);

}

Tree\* AddNode(int val, Tree\* root)

{

if(root != NULL && root->value == val)

{

printf("ERROR: same value in tree\n");

return root;

}

if (root == NULL)

root = CreateNode(val);

else if (val < root->value)

root->left = AddNode(val, root->left);

else

root->right = AddNode(val, root->right);

return(root);

}

Tree\* SearchNode(int val, Tree\* root)

{

if (root == NULL)

return NULL;

if (root->value == val)

return root;

if (val < root->value)

return SearchNode(val, root->left);

else

return SearchNode(val, root->right);

}

int EntryCount(int value, Tree\* root, int entry)

{

if (root == NULL)

return -1;

if (root->value == value)

return entry + 1;

if (value < root->value)

return EntryCount(value, root->left, entry + 1);

else

return EntryCount(value, root->right, entry + 1);

}

void ClearTree(Tree\* root)

{

if (root == NULL)

return;

ClearTree(root->left);

ClearTree(root->right);

free(root);

}

void TreePrint(Tree\* root, int tabs)

{

for (int i = 0; i < tabs \* 2; i++)

printf(" ");

if (root == NULL)

{

printf("-\n");

return;

}

printf("%d\n", root->value);

TreePrint(root->left, tabs + 1);

TreePrint(root->right, tabs + 1);

}

void PrintOneNode(Tree\* node)

{

if (node == NULL)

{

printf("No node\n");

return;

}

printf("%d\n", node->value);

if (node->left != NULL)

{

printf("%d", node->left->value);

}

printf(" ");

if (node->right != NULL)

{

printf("%d", node->right->value);

}

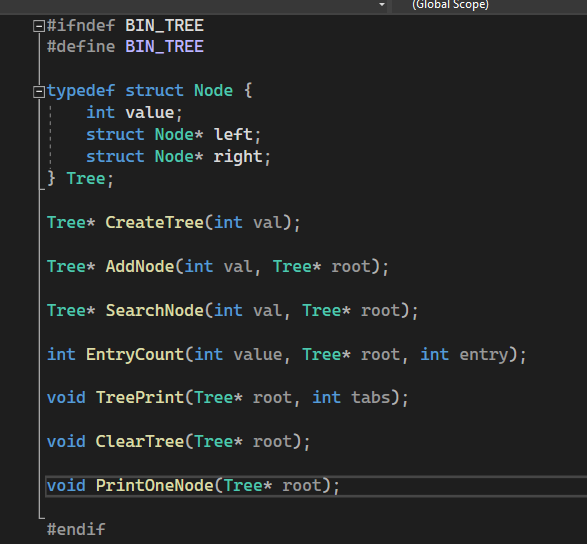
printf("\n");

}

**Задания**

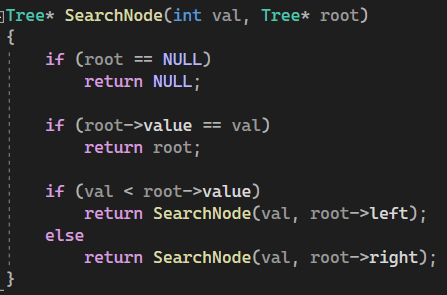
Задание 0

Реализовал бинарное дерево



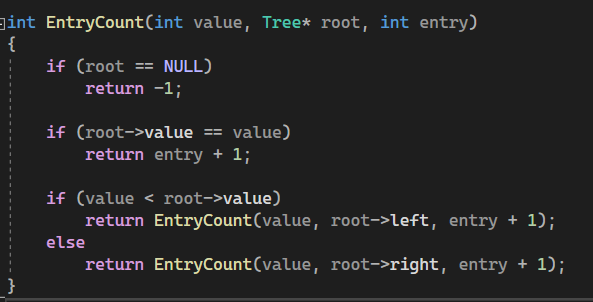
Задание 1

Поиск значения в дереве



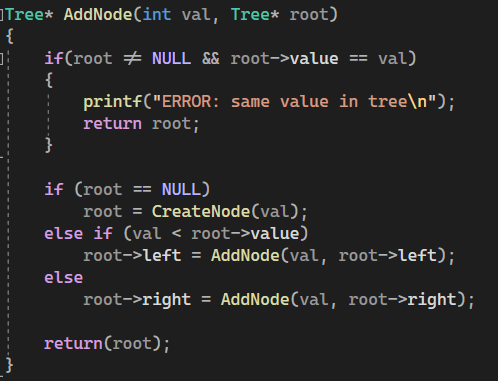
Задание 2

Подсчет числа вхождений элемента в дерево



Задание 3

Исключение одинаковых элементов в дереве

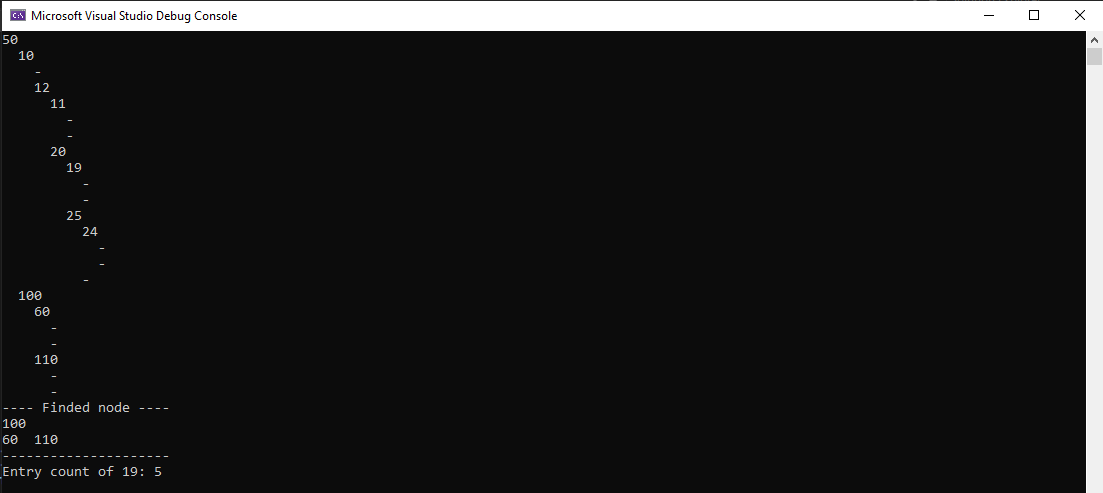


Задание 4

Средняя сложность – O(log(n))

Наихудшая сложность – O(n) – дерево выродилось

**Результат работы программы**

****

**Вывод**

Разработал алгоритм бинарного дерева поиска. Научился работать с данным типом данных