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

**ОТЧЕТ**

по лабораторной работе №3

по дисциплине «Трансляторы и компиляторы»

## на тему «Построение таблиц идентификаторов методом бинарного дерева»

Выполнили студенты группы 22ВВП1

Демин М.С.

Беляев Д. И.

Приняли:

Дубинин В.Н.

Карамышева Н.С.

Пенза 2024

**Название**

Построение таблиц идентификаторов методом бинарного дерева

**Цель работы**

Ознакомление с методами построения таблиц идентификаторов. Изучение метода бинарного дерева для построения таблиц идентификаторов.

**Задание**

Написать программу на языке C, реализующую алгоритм построения бинарного дерева. Входной поток идентификаторов следует получать с консоли, либо из текстового файла, используя в качестве разделителя символы перевода строки и возврата каретки. Необходимо также предусмотреть функцию поиска заданного идентификатора в программе.

**Листинг**

**main.c**

#include <stdio.h>

#include "binTree.h"

int main()

{

Tree\* root = CreateTree("a50");

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

root = AddNode("a10", root);

root = AddNode("a12", root);

root = AddNode("a11", root);

root = AddNode("a20", root);

root = AddNode("a19", root);

root = AddNode("a25", root);

root = AddNode("a24", root);

root = AddNode("a100", root);

root = AddNode("a60", root);

root = AddNode("a110", root);

TreePrint(root, 0);

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

Tree\* find = SearchNode("a11", root);

PrintOneNode(find);

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

printf("Entry count of %s: %d\n", "a19", EntryCount("a19", root, 0));

ClearTree(root);

return 0;

}

**binTree.c**

#include <stdio.h>

#include <stdlib.h>

#include "binTree.h"

Tree\* CreateNode(char value[SIZE])

{

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

if (tmp == NULL) {

printf("ERROR: can't allocate memory\n");

return NULL;

}

strcpy(tmp->value, value);

tmp->left = NULL;

tmp->right = NULL;

return tmp;

}

Tree\* CreateTree(char\* value)

{

return CreateNode(value);

}

Tree\* AddNode(char\* value, Tree\* root)

{

if (root != NULL && strcmp(root->value, value) == 0) {

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

return root;

}

if (root == NULL) {

root = CreateNode(value);

} else if (strcmp(value, root->value ) < 0) {

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

} else {

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

}

return(root);

}

Tree\* SearchNode(char\* value, Tree\* root)

{

if (root == NULL) {

return NULL;

}

if (strcmp(root->value, value) == 0) {

return root;

}

if (strcmp(value, root->value ) < 0) {

return SearchNode(value, root->left);

} else {

return SearchNode(value, root->right);

}

}

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

{

if (root == NULL) {

return -1;

}

if (strcmp(root->value, value) == 0) {

return entry + 1;

}

if (strcmp(value, root->value ) < 0) {

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

if (node->left != NULL) {

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

}

printf(" ");

if (node->right != NULL) {

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

}

printf("\n");

}

void TreeWalk(Tree\* root)

{

if (root == NULL) {

return;

}

printf("%s ", root->value);

TreeWalk(root->left);

TreeWalk(root->right);

}

**binTree.h**

#ifndef BIN\_TREE

#define BIN\_TREE

#include <string.h>

#define SIZE 256

typedef struct Node {

char value[SIZE];

struct Node\* left;

struct Node\* right;

} Tree;

Tree\* CreateTree(char\* value);

Tree\* AddNode(char\* value, Tree\* root);

Tree\* SearchNode(char\* value, Tree\* root);

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

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

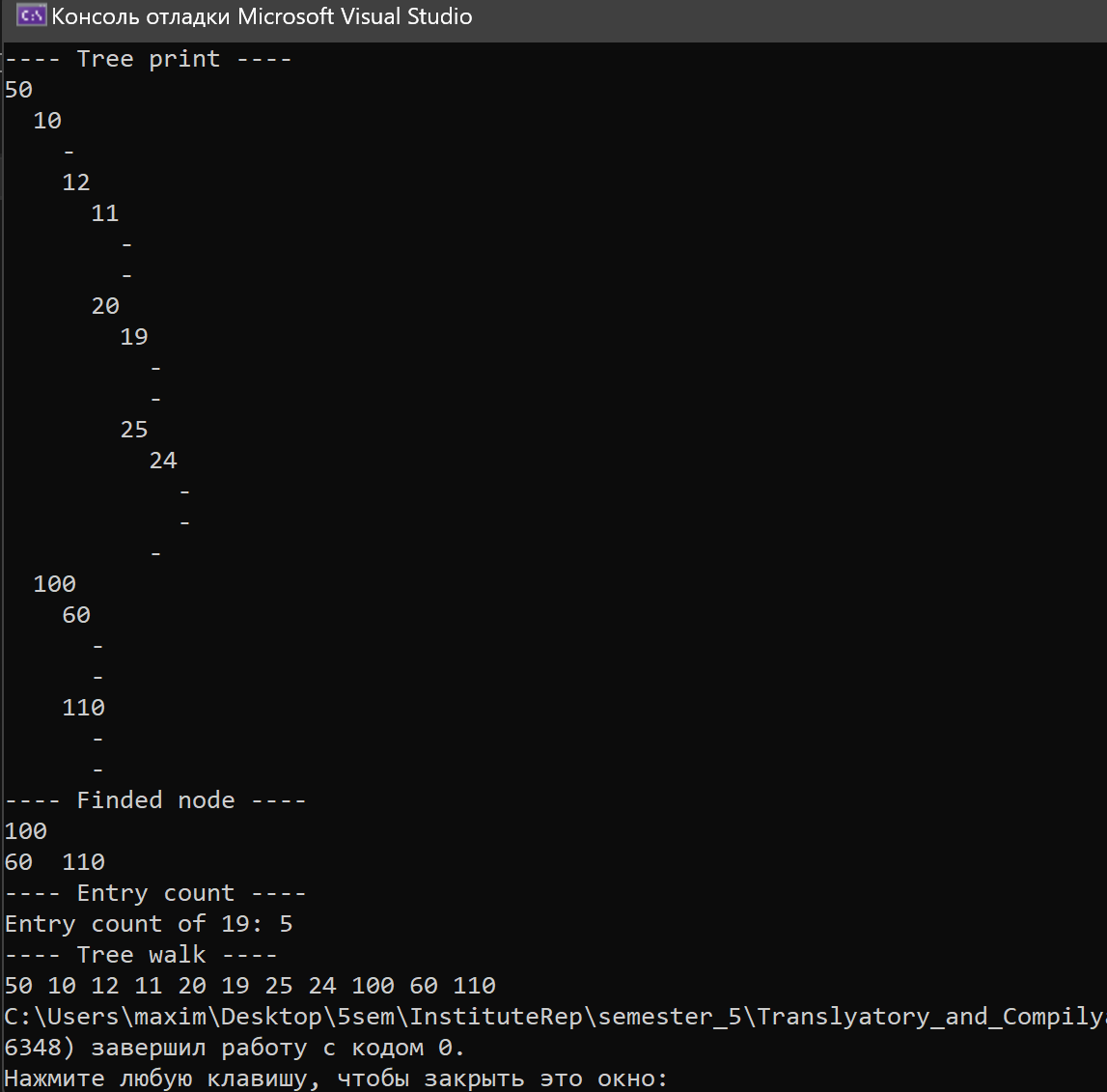
void ClearTree(Tree\* root);

void PrintOneNode(Tree\* root);

void TreeWalk(Tree\* root);

#endif

**Результат работы**



**Вывод**

Ознакомились с методами построения таблиц идентификаторов. Изучили метод бинарного дерева для построения таблиц идентификаторов.