**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

Лабораторная работа №6

по дисциплине «Программирование»

Деревья

Группа: АВТ-041

Студенты: Антонов А.

Преподаватель: Балакин В.

НОВОСИБИРСК 2021

**Задача**

Вершина дерева содержит динамический массив целых значений и два указателя на потомков. Значения в дереве не упорядочены. Размерность динамического массива в корневой вершине - N, на каждом следующем уровне - в 2 раза больше. Функция включает новое значение в свободное место в массиве ближайшей к корню вершины.

**Проектирование программы**

Идея: Создание функций: включение нового значения, рекурсивная функция обхода и вывода, функция вывода, функция создание ноды (с учитыванием имеющихся вершин)

Составные части программы:

1. Создание структуры дерева:

typedef struct tree {

char\* value;

unsigned int size;

struct tree\*\* child;

} TRE

1. Создание ноды, с выделением памяти для новой вершины:

TREE\* CreateNode(const char\* val, unsigned int size) // создание ноды для нового значения, учитывание уже имеющихся вершин

{

TREE\* new\_node = (TREE\*)malloc(sizeof(TREE));

new\_node->value = (char\*)malloc(sizeof(char) \* (strlen(val) + 1));

//strcpy(new\_node->value, value);

for (int i = 0; i < strlen(val); i++) {

char s = val[i];

new\_node->value[i] = s;

}

new\_node->value[strlen(val)] = '\0';

new\_node->size = size;

new\_node->child = (TREE\*\*)malloc(sizeof(TREE) \* new\_node->size);

for (unsigned int i = 0; i < new\_node->size; i++)

\*(new\_node->child + i) = NULL;

return (new\_node);

}

1. Сканирование нод дерева, поиск свободного места в дереве:

void ScanTree(TREE\* node) // рекурсивная функция обхода и вывода

{

if (node == NULL)

return;

for (unsigned int i = 0; i < node->size; i++)

if (node->child[i] == NULL) {

if (found\_node == NULL) {

found\_node = node;

break;

}

else {

if (node->size < found\_node->size) {

found\_node = node;

break;

}

}

}

for (unsigned int i = 0; i < node->size; i++)

ScanTree(node->child[i]);

return;

}

1. Включение нового значения в дерево:

TREE\* InsertNode(TREE\* root, const char\* value) // включение нового значения в свободное место в массиве

{

TREE\* new\_node = NULL;

if (root == NULL) {

root = CreateNode(value, N);

return root;

}

found\_node = NULL;

ScanTree(root);

new\_node = CreateNode(value, found\_node->size \* 2);

for (unsigned int i = 0; i < found\_node->size; i++)

if (found\_node->child[i] == NULL) {

found\_node->child[i] = new\_node;

break;

}

return new\_node;

}

1. Функция вывода дерева на консоль:

void print(TREE\* root, int lvl) { // функция вывода дерева на консоль

if (root == NULL) return;

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

for (int i = 0; i < root->size; i++)

print(root->child[i], lvl + 1);

}

1. Функция main:

int main(void)

{

int lvl = 0;

TREE\* root = NULL;

root = InsertNode(root, "root");

InsertNode(root, "child\_1");

InsertNode(root, "child\_2");

InsertNode(root, "child\_3");

InsertNode(root, "child\_4");

InsertNode(root, "child\_5");

InsertNode(root, "child\_6");

print(root, lvl);

}

Переменные:

size - размерность

child - новое значение

val,s, value - массивы

root - ветка

lvl - уровень вершины

**Текст программы с комментариями**

#define \_CTR\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define N 2

typedef struct tree {

char\* value;

unsigned int size;

struct tree\*\* child;

} TREE;

TREE\* CreateNode(const char\* val, unsigned int size) // создание ноды для нового значения, учитывание уже имеющихся вершин

{

TREE\* new\_node = (TREE\*)malloc(sizeof(TREE));

new\_node->value = (char\*)malloc(sizeof(char) \* (strlen(val) + 1));

//strcpy(new\_node->value, value);

for (int i = 0; i < strlen(val); i++) {

char s = val[i];

new\_node->value[i] = s;

}

new\_node->value[strlen(val)] = '\0';

new\_node->size = size;

new\_node->child = (TREE\*\*)malloc(sizeof(TREE) \* new\_node->size);

for (unsigned int i = 0; i < new\_node->size; i++)

\*(new\_node->child + i) = NULL;

return (new\_node);

}

TREE\* found\_node;

void ScanTree(TREE\* node) // рекурсивная функция обхода и вывода

{

if (node == NULL)

return;

for (unsigned int i = 0; i < node->size; i++)

if (node->child[i] == NULL) {

if (found\_node == NULL) {

found\_node = node;

break;

}

else {

if (node->size < found\_node->size) {

found\_node = node;

break;

}

}

}

for (unsigned int i = 0; i < node->size; i++)

ScanTree(node->child[i]);

return;

}

TREE\* InsertNode(TREE\* root, const char\* value) // включение нового значения в свободное место в массиве

{

TREE\* new\_node = NULL;

if (root == NULL) {

root = CreateNode(value, N);

return root;

}

found\_node = NULL;

ScanTree(root);

new\_node = CreateNode(value, found\_node->size \* 2);

for (unsigned int i = 0; i < found\_node->size; i++)

if (found\_node->child[i] == NULL) {

found\_node->child[i] = new\_node;

break;

}

return new\_node;

}

void print(TREE\* root, int lvl) { // функция вывода дерева на консоль

if (root == NULL) return;

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

for (int i = 0; i < root->size; i++)

print(root->child[i], lvl + 1);

}

int main(void)

{

int lvl = 0;

TREE\* root = NULL;

root = InsertNode(root, "root");

InsertNode(root, "child\_1");

InsertNode(root, "child\_2");

InsertNode(root, "child\_3");

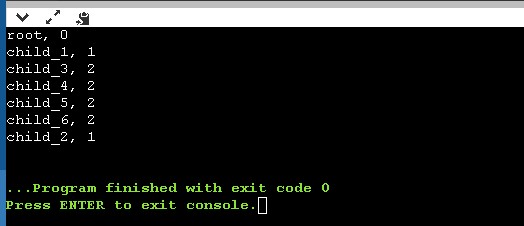
InsertNode(root, "child\_4");

InsertNode(root, "child\_5");

InsertNode(root, "child\_6");

print(root, lvl);

}

**Пример работы программы**

**Вывод**  
Функция была реализована, программа работает корректно.