**МИНОБРНАУКИ РОССИИ**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ  
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»  
(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №7

Дисциплина: Алгоритмы и структуры данных

по теме «Структуры данных «дерево»

Выполнил: ст. группы ВТ-22  
Макаров Даниил Сергеевич

Проверил: Синюк В.Г.

Белгород 2018Л а б о р а т о р н а я р а б о т а № 7

**Структуры данных типа «дерево» (Pascal/С)**

***Цель работы****: изучить СД типа «дерево», научиться их программно реализовывать и использовать*.

З а д а н и е

1. Для СД типа «дерево» определить:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.

1.1.2. Набор допустимых операций.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения.

1.2.2. Объем памяти, занимаемый экземпляром СД.

1.2.3. Формат внутреннего представления СД и способ его интерпретации.

1.2.4. Характеристику допустимых значений.

1.2.5. Тип доступа к элементам.

1.3. Логический уровень представления СД.

Способ описания СД и экземпляра СД на языке программирования.

2. Реализовать СД типа «дерево» в соответствии с вариантом индивидуального (табл.17) задания в виде модуля.

3. Разработать программу для решения задачи в соответствии с вариантом индивидуального задания (см. табл.17) с использованием модуля, полученного в результате выполнения пункта 2 задания.

Вариант 7

а) Procedure BildTree(var T:Tree);

Строит дерево в глубину.

б) Function CalcLevel(T:Tree; n:byte):byte;

Определяет количество вершин в дереве T на n-ом уровне.

в) Procedure WriteWays(T:Tree);

Выводит все пути от листьев до корня (в i-ю строку вывода — i-ый путь).

#ifndef TREE\_H\_

#define TREE\_H\_

#include <stdlib.h>

#include <stdio.h>

extern const char tree\_ok;

extern const char tree\_memerror;

extern const char tree\_under;

typedef char base\_type;

typedef struct e node;

typedef struct e{

base\_type data;

node \*left\_son;

node \*right\_son;

};

typedef node tree;

char tree\_error;

tree\* node\_init(); // инициализация — создается элемент, который будет содержать корень дерева

void node\_write(tree \*t, base\_type e); //запись данных

void node\_read(tree \*t,base\_type \*e);//чтение

void node\_delete(tree \*t);//удаление листа

void move\_left(tree \*t);//перейти к левому сыну

void move\_right(tree \*t);//перейти к правому сыну

int check\_left\_son(tree \*t);//1 — есть левый сын, 0 — нет

int check\_right\_son(tree \*t);//1 — есть правый сын, 0 — нет

int tree\_empty(tree \*t);//1 — пустое дерево,0 — не пустое

unsigned count\_node\_lvl(tree \*t,char lvl);

void tree\_data\_set(tree\* t,int data);

#endif

#include <stdlib.h>

#include <stdio.h>

#include "tree.h"

const char tree\_ok=0;

const char tree\_memerror=1;

const char tree\_under=2;

tree\* node\_init(){

tree\* temp\_pointer;

temp\_pointer=(tree\*)malloc(sizeof(tree));

if(temp\_pointer==NULL){

tree\_error=tree\_memerror;

}

temp\_pointer->data=0;

temp\_pointer->left\_son=NULL;

temp\_pointer->right\_son=NULL;

return temp\_pointer;

}

void node\_write(tree \*t,base\_type e){

t->data=e;

}

void node\_read(tree \*t,base\_type \*e){

\*e=t->data;

}

int check\_left\_son(tree \*t){

return !(t->left\_son==NULL);

}

int check\_right\_son(tree \*t){

return !(t->right\_son==NULL);

}

int tree\_empty(tree \*t){

return((!(check\_left\_son(t))&&(!(check\_left\_son(t))&&t->data!=0)));

}

void move\_left(tree \*t){

t=t->left\_son;

}

void move\_right(tree \*t){

t=t->right\_son;

}

void tree\_data\_set(tree\* t,int data){

t->data=data;

}

unsigned count\_node\_lvl(tree \*t,char lvl){

unsigned count\_node(tree \*t,char lvl,int i){

static unsigned count=0;

if(t==NULL) return 0;

else if(i==lvl){

return 1;

}

else{

count+=count\_node(t->left\_son,lvl,i+1);

count+=count\_node(t->right\_son,lvl,i+1);

}

return (i>0)?0:count;}

return count\_node(t,lvl,0);

}

#include <stdlib.h>

#include <stdio.h>

#include "tree.h"

#include <string.h>

int i=0;

int temp\_i=0;

int print\_count=1;

char str[255]="ABCD..E..F..G.HI..J..";

void build\_node(tree\* t){

fflush(stdout);

if(str[i]!='\0'){

if(str[i]!='.'){

t=node\_init();

t->data=str[i];

i++;

build\_node(t->left\_son);

i=temp\_i+1;

build\_node(t->right\_son);

}

temp\_i=i;

}

}

void tree\_paths\_print(tree \*t){

char print\_str[255];

strcat(print\_str,t->data);

if(!(check\_left\_son(t) && check\_right\_son(t))){

printf("%d.",print\_count);

print\_count++;

revers\_print\_str(print\_str);

}

else

{

if(check\_left\_son(t)){

tree\_paths\_print(t->left\_son);

}

if(check\_right\_son(t)){

tree\_paths\_print(t->right\_son);

}

}

}

void revers\_print\_str(char\* str){

int str\_len=strlen(str);

for(i=str\_len;i--;i>0){

printf("%c",str[i]);

}

printf("\n");

}

int main(){

tree \*t;

tree \*start\_node;

char lvl=1;

printf("%s\n",str);

build\_node(t);

printf("count node on lvl %d - %d\n",count\_node\_lvl(t,lvl),lvl);

tree\_paths\_print(t);

return 0;

}