**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕНЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.Шухова»**

**(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа № 7

По дисциплине: Алгоритмы и СД

Тема: **Структура данных типа «дерево» (Pascal/С)**

Выполнил: ст.группы ПВ - 21

Браткова И.О.

Белгород 2017

**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.

**Задание 1**

**1.1. Абстрактный уровень представления СД:**

1.1.1. Характер организованности и изменчивости:

Динамический

1.1.2. Набор допустимых операций:

Инициализация, создание корня, запись данных, чтение данных, проверка — есть ли левый сын, проверка — есть ли правый сын, переход к левому сыну, переход к правому сыну, проверка — пустое ли дерево, удаление листа.

**1.2. Физический уровень представления СД:**

1.2.1. Схема хранения:

связная.

1.2.2. Объем памяти, занимаемый экземпляром СД:

n\*sizeof(tree).

1.2.3. Формат внутреннего представления СД и способ его интерпретации:

БД располагается в динамической памяти. Каждая вершина БД представляет собой СД типа «запись», содержащую информационную часть и адреса сыновей.

1.2.4. Характеристика допустимых значений:

Зависит от базового типа.

1.2.5. Тип доступа к элементам:

последовательный.

**1.3. Логический уровень представления СД.**

Способ описания СД и экземпляра СД на языке программирования:

typedef struct node

{

int data; // данные

struct node\* L\_son; // левый сын

struct node\* R\_son; // правый сын

};

typedef struct node tree;

**Задание 2**

**Derevo.h**

#ifndef DEREVO\_H\_INCLUDED #define DEREVO\_H\_INCLUDED

enum {TreeOk, TreeNotMem, TreeUnder}; typedef char BaseType;

typedef struct El \*elptr;

typedef struct El{ BaseType Data; elptr LSon; elptr RSon; } El; typedef elptr Tree;

void InitTree(Tree \*T); //Инициализация дерева

void CreateRoot(Tree\* T); //создание корня

int IsLSon(Tree T); //Есть левый сын

int IsRSon(Tree T); //Есть правый сын

void WriteDataTree(Tree \*T, BaseType E); //запись данных в дерево

void ReadDataTree(Tree T, BaseType\* E); //Чтение элемента дерева

void MoveToLSon(Tree T, Tree \*TS); //Переход к левому сыну

void MoveTorSon(Tree T, Tree \*TS); //Переход к правому сыну

int IsEmptyTree(Tree T); //Дерево пусто

void DelTree(Tree T); //Удалить дерево

#endif // DEREVO\_H\_INCLUDED

**Derevo.c**

#include<stdio.h> #include<stdlib.h> #include"Derevo.h" short TreeError;

void InitTree(Tree \*T) { CreateRoot(T); }

void CreateRoot(Tree\* T) { (\*T) = (elptr)malloc(sizeof(elptr));(\*T)->LSon = NULL =(\*T)->RSon; (\*T)->Data = 0; }

int IsLSon(Tree T) { return (T->LSon != NULL); }

int IsRSon(Tree T) { return (T->RSon != NULL); }

void WriteDataTree(Tree \*T, BaseType E)

{

if (IsEmptyTree(T)) { T->Data = E; return; }

if (E<T->Data) { CreateRoot(&T->LSon); WriteDataTree(T->LSon, E); return; }

if (E>T->Data) { CreateRoot(&T->RSon); WriteDataTree(T->RSon, E); return; }

}

void ReadDataTree(Tree T, BaseType\* E)

{

if (T == NULL) { TreeError = TreeUnder; return; }

else \*E = T->Data;

}

void MoveToLSon(Tree T, Tree \*TS)

{ if (IsLSon(T)) \*TS = T->LSon; else TreeError = TreeUnder; }

void MoveTorSon(Tree T, Tree \*TS)

{ if (IsRSon(T)) \*TS = T->RSon; else TreeError = TreeUnder; }

int IsEmptyTree(Tree T) { return(T == NULL); }

void DelTree(Tree T)

{

Tree \*tmp;

if (IsRSon(T)) { MoveTorSon(T,&tmp); DelTree(tmp); }

if (IsLSon(T)) { MoveToLSon(T,&tmp); DelTree(tmp); }

free(tmp); free(T);

}

**main.c**

void BildRegTree(Tree \*T); //строит упорядоченное дерево

void WriteSequence(Tree \*T, int N); /\*Выводит упорядоченную по возрастанию последовательность, составленную из элементов дерева, меньших k.\*/

int WriteNodes(Tree \*T); /\*Выводит количество вершин правого и левого поддеревьев и возвращает общее количество вершин\*/

int main()

{

Tree T;

int N;

InitTree(&T);

BildRegTree(&T);

N=WriteNodes(&T);

WriteSequence(&T, 5);

printf("%d\n", N);

return 0;

}

void BildRegTree(Tree \*T)

{

int a; scanf("%d", &a);

while (a!=0)

{

WriteDataTree(T,a); scanf("%d", &a);

}

}

void WriteSequence(Tree \*T, int N)

{

if (N!=0)

{

Tree TS;

InitTree(&TS);

MoveToLSon(T,&TS);

WriteSequence(&TS,N-1);

MoveToRSon(T,&TS);

WriteSequence(&TS,N-1);

}

printf("%d \n", T->data);

}

int WriteNodes(Tree \*T)

{

Tree TS; int M=0,K=0;

if (IsLSon(T))

{

MoveToLSon(T,&TS); M+=WriteNodes(&TS);

printf("LD-%d\n", M);//кол-во левых вершин

}

if (IsRSon(T))

{

MoveToRSon(T,&TS); K+=WriteNodes(&TS);

printf("RD-%d\n", K); //кол-во правых вершин

}

return M+K;

}