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

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

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

З а д а н и е

1. Для СД «дерево» определить:

1.1. Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.

Нелинейная структура.

1.1.2. Набор допустимых операций.

Инициализация дерева; создание корня; запись данных в дерево; чтение элемента дерева; проверка на наличие левого сына; проверка на наличие правого сына; переход к левому сыну; переход к правому сыну; удаление листа.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения.

Связная.

1.2.2. Объем памяти, занимаемый экземпляром СД.

Зависит от реализации.

1.2.3. Формат внутреннего представления СД и способ его интерпретации.

Может быть реализован в статической, динамической памяти, на массиве.

1.2.4. Характеристику 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.

Tree T;

2. Реализовать СД «дерево» в соответствии с вариантом индивидуального (табл.17) задания в виде модулей на языках Pascal и С.

3. Разработать программы на языках Pascal и С для решения задачи в соответствии с вариантом индивидуального задания (см. табл.17) с использованием модулей, полученных в результате выполнения пункта 2 задания.

|  |  |  |
| --- | --- | --- |
| Номер варианта | Номер модуля | Варианты задач |
| 10 | 6 | 10 |

**Вариант 6**

а) Построить упорядоченное дерево.

б) Вывести упорядоченную по возрастанию последовательность, составленную из элементов дерева.

в) Вывести упорядоченную по убыванию последовательность, составленную из элементов дерева.

**Модуль**

10. Дерево в динамической памяти (базовый тип — произвольный).

**Спецификация СД на языке C:**

#if !defined(\_\_TREE5\_H)

const TreeOk = 0;

const TreeNotMem = 1;

const TreeUnder = 2;

typedef void\* BaseType;

typedef struct element \*ptrel;

typedef struct element{basetype data;

ptrel LSon;

ptrel RSon;}

typedef PtrEl \*Tree;

unsigned Size;

short TreeError;

void InitTree(Tree \*T,unsigned size)// инициализация дерева

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

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

void ReadDataTree(Tree \*T,BaseType \*E)//чтение

int IsLSon(Tree \*T)//1 — есть левый сын, 0 — нет

int IsRSon(Tree \*T)//1 — есть правый сын, 0 — нет

void MoveToLSon(Tree \*T, Tree \*TS)// перейти к левому сыну, где T — адрес ячейки, содержащей адрес текущей вершины, TS — адрес ячейки, содержащей адрес корня левого поддерева(левого сына)

void MoveToRSon(Tree \*T, Tree \*TS)//перейти к правому сыну

int IsEmptyTree(Tree \*T)//1 — пустое дерево,0 — не пустое

void DelTree(Tree \*T)//удаление листа

#endif

**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);

//создание корня

void CreateRoot(Tree\* T);

//Есть левый сын

int IsLSon(Tree T);

//Есть левый сын

int IsRSon(Tree T);

//запись данных в дерево

void WriteDataTree(Tree \*T, BaseType E);

//Чтение элемента дерева

void ReadDataTree(Tree T, BaseType\* E);

//Переход к левому сыну

void MoveToLSon(Tree T, Tree \*TS);

//Переход к правому сыну

void MoveTorSon(Tree T, Tree \*TS);

//Дерево пусто

int IsEmptyTree(Tree T);

void DelTree(Tree T);

#endif // DEREVO\_H\_INCLUDED

**Derevo.c**

#include<stdio.h>

#include<stdlib.h>

#include"Derevo.h"

short TreeError;

void InitTree(Tree \*T)

{

CreateRoot(T);

}

//создание корня

void CreateRoot(Tree\* T)

{

(\*T) = (elptr)malloc(sizeof(elptr));

(\*T)->LSon = NULL;

(\*T)->RSon = NULL;

(\*T)->Data = 0;

}

//Есть левый сын

int IsLSon(Tree T)

{

return (T->LSon != NULL);

}

//Есть левый сын

int IsRSon(Tree T)

{

return (T->RSon != NULL);

}

//запись данных в дерево

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;

}

}

//Чтение элемента дерева

void ReadDataTree(Tree T, BaseType\* E)

{

if (T == NULL)

{

TreeError = TreeUnder;

return;

}

else

\*E = T->Data;

}

//Переход к левому сыну

void MoveToLSon(Tree T, Tree \*TS)

{

if (IsLSon(T))

\*TS = T->LSon;

else

TreeError = TreeUnder;

}

//Переход к правому сыну

void MoveTorSon(Tree T, Tree \*TS)

{

if (IsRSon(T))

\*TS = T->RSon;

else

TreeError = TreeUnder;

}

//Дерево пусто

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**

#include <stdio.h>

#include <stdlib.h>

#include"Derevo.h"

//симметричный проход слева-направо

void Simm\_Front(Tree T)

{

if (!IsEmptyTree(T))

{

Simm\_Front(T->LSon);

printf("%d ", T->Data);

Simm\_Front(T->RSon);

}

}

//симметричный проход справа-налево

void Simm\_Back(Tree T)

{

if (!IsEmptyTree(T))

{

Simm\_Back(T->RSon);

printf("%d ", T->Data);

Simm\_Back(T->LSon);

}

}

void BuildTree(Tree \*T1, int k){

int E;

for (i=0;i<k;i++)

{

scanf("%d",&E);

WriteDataTree(T, E);

}

}

int main()

{

Basetype T;

int k;

Printf ("k = ");

Scanf ("%d",&k);

BuildTree(&T,k);

printf("Tree tops in direct order:\n");

Simm\_Front(T);

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

printf("Tree tops in reverse order:\n");

Simm\_Back(T);

return 0;

}