МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

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

по дисциплине: Алгоритмы и структуры данных
тема: «Структура данных «дерево» (Pascal/C)»

Выполнил: ст. группы ПВ-202
Буйвало Анастасия Андреевна

Проверил:
Кабалянц Петр Степанович
Маньшин Илья Михайлович

Белгород 2021 г.

Лабораторная работа №7

«Структура данных «дерево» (Pascal/C)»

Цель работы:

Изучить СД типа «дерево», научиться их программно реализовывать и использовать.

Задание:

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.
      a. Способ описания СД и экземпляра СД на языке программирования.
2. Реализовать СД типа «дерево» в соответствии с вариантом индивидуального задания в виде модуля.
3. Разработать программу для решения задачи в соответствии с вариантом индивидуального задания с использованием модуля, полученного в результате выполнения пункта 2 задания.

7	7	7

Выполнение работы:
Задание 1:

СД «Дерево»:

Абстрактный уровень представления СД:

1. Характер организованности – дерево;
2. Изменчивость – динамическая СД;
3. 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.

Физический уровень представления СД:

1. Схема хранения: связная.
2. Объем памяти, занимаемый экземпляром СД: в зависимости от реализации.
3. Формат внутреннего представления СД и способ его интерпретации:

Объем занимаемой памяти: $N * M + T * Y + 4$, где N – размер массива, M – размер узла дерева, T – количество элементов в древе, Y – размер одного элемента, 12 - размер узла (2 числа + 1 указатель) , 4 – указатель на дерево.

4. Характеристика допустимых значений: в зависимости от реализации
5. Тип доступа к элементам: последовательный

Логический уровень представления СД:

Способ описания СД и экземпляра СД на языке С:

Дерево - это указатель на индекс корня в массиве, элементы массива представляют собой записи состоящие из полей:

1- Указатель на данные
2- Индекс левого сына
3- Индекс правого сына

Способ описания СД:

```
typedef ptrel *Tree;
```

Способ описания экземпляра: `Tree T;`

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

7. Элементы дерева находятся в массиве, расположенном в динамической памяти. Базовый тип - произвольный. «Свободные» элементы массива объединяются в список (ССЭ), на начало которого указывает левый сын первого элемента массива. В массиве может находиться несколько деревьев.

Задание 2:

Файл _AiSD7.h:

```
//
// Created by настя буйвало on 16/11/2021.
//

#ifndef AISDTEST__AISD7_H
#define AISDTEST__AISD7_H


#define Index 1000

extern const int TreeOk;
extern const int TreeNotMem;
extern const int TreeUnder;

extern short TreeError;

typedef void* BaseType;
typedef unsigned ptrel;
typedef struct element{
    BaseType data;
    ptrel LSon;
    ptrel RSon;
};
typedef struct element Mem[Index];
typedef Mem *Pmem;
typedef ptrel *Tree;
unsigned Size; //размер массива
Pmem Pbuf;//указатель на массив
unsigned SizeEl;

void InitTree_arr(Tree* T, unsigned size, Pmem *Pbuf);
void InitTree(Tree* T);// инициализация дерева
void CreateRoot(Tree T, Pmem Pbuf); //создание корня
void WriteDataTree(Tree T, BaseType E, Pmem Pbuf, unsigned SizeEl, unsigned size);
//запись данных
void ReadDataTree(Tree T,BaseType *E, Pmem Pbuf, unsigned SizeEl);//чтение
int IsLSon(Tree *T, unsigned size, Pmem Pbuf);//1 – есть левыи˘сын, 0 – нет
int IsRSon(Tree *T,  unsigned size, Pmem Pbuf);//1 – есть правыи˘сын, 0 – нет
void MoveToLSon(Tree T, Tree TS);// переиҐи к левому сыну, где T –адрес ячеиќи,
содержащеи˘адрес
// текущеи˘вершины, TS – адрес ячеиќи, содержащеи˘адрес корня левого поддерева(левого
сына)
void MoveToRSon(Tree T, Tree TS);//переиҐи к правому сыну
int IsEmptyTree(Tree T, Pmem Pbuf);//1 – пустое дерево,0 – не пустое
void DelTree(Tree *T, Pmem Pbuf);//удаление листа

#endif
```

Файл _AiSD7.c:

```
//
// Created by настя буйвало on 16/11/2021.
//
#include <mm_malloc.h>
#include "_AiSD7.h"
```

```
#include <stdio.h>
#define Index2 1001

const int TreeOk = 0;
const int TreeNotMem = 1;
const int TreeUnder = 2;

short TreeError = TreeOk;

void List_free_el(struct element Mem[Index], unsigned size)
{
    for(int i = 0; i < size-1; i++)
        Mem[i].RSon = i+1;
    Mem[size-1].RSon = 0;
}

void InitTree(Tree* T)
{
    *T = (Tree)calloc(1, sizeof(ptrel));
    if(T == NULL)
        TreeError = TreeNotMem;
    else
        TreeError = TreeOk;
}

//инициализация дерева
void InitTree_arr(Tree* T, unsigned size, Pmem *Pbuf)
{
    if(size <= Index) {
        *Pbuf = (Pmem) calloc(size, sizeof(struct element)); //выделение памяти под
массив
        List_free_el(*Pbuf, size);
    }
    else
        TreeError = TreeNotMem;
}

//пустое ли дерево(если есть в списке свободных, то пустое или если оно вне границ
массива, 0- не пустое
int IsEmptyTree(Tree T, Pmem Pbuf)
{
    int t = 0;
    if(*T > Index || *T <= 0)
        return 1;
    while((*T) != (*Pbuf)[t].RSon && (*Pbuf)[t].RSon != 0){
        t = (*Pbuf)[t].RSon;
    }
    if((*Pbuf)[t].RSon == 0)
        return 0;
    else
        return 1;
}

//создание корня
void CreateRoot(Tree T, Pmem Pbuf)
{
    if((*Pbuf)[0].RSon != 0){
        unsigned t = (*Pbuf)[0].RSon;
        (*Pbuf)[0].RSon = (*Pbuf)[t].RSon;
        *T = t;
        (*Pbuf)[t].RSon = Index2;
        (*Pbuf)[t].LSon = Index2;
```

```
        (*Pbuf)[t].data = (void*)calloc(1, sizeof(BaseType));
    }
    else
        TreeError = TreeNotMem;
}

//запись данных
void WriteDataTree(Tree T, BaseType E, Pmem Pbuf, unsigned SizeEl, unsigned size)
{
    if(*T <= 0 && *T > size){
        TreeError = TreeUnder;
        return;
    }
    char *w = (char*)(*Pbuf)[*T].data;
    char *r = (char*)E;
    for(int j = 0; j < SizeEl; j++, w++, r++)
        *w = *r;
}

//чтение данных
void ReadDataTree(Tree T,BaseType *E, Pmem Pbuf, unsigned SizeEl)
{
    if(IsEmptyTree(T, Pbuf) == 1){
        TreeError = TreeUnder;
        return;
    }
    char *w = (char*)E;
    char *r = (char*)(*Pbuf)[*T].data;
    for(int j = 0; j < SizeEl; j++, w++, r++)
        *w = *r;
}

int IsLSon(Tree *T, unsigned size, Pmem Pbuf)
{
    return((*Pbuf)[(**T)]->LSon != 0 && (*Pbuf)[(**T)]->LSon < size);
}

int IsRSon(Tree *T,  unsigned size, Pmem Pbuf)
{
    return((*Pbuf)[(**T)]->RSon != 0 && (*Pbuf)[(**T)]->RSon < size);
}

void MoveToLSon(Tree T, Tree TS)
{
    if(T != NULL && TS != NULL)
        T = TS;
    else
        TreeError = TreeUnder;
}

void MoveToRSon(Tree T, Tree TS)
{
    if(T != NULL && TS != NULL)
        T = TS;
    else
        TreeError = TreeUnder;
}

//удаление листа
void DelTree(Tree *T, Pmem Pbuf)
{
    if(!((*Pbuf)[**T]->RSon == Index2 && (*Pbuf)[**T]->LSon == Index2)){
```

```
        TreeError = TreeUnder;
        return;
    }
    free((*Pbuf)[**T]->data);
    free(*T);
    free(T);
}
```

Задание 3.

task.h

```
//
// Created by настя буйвало on 16/11/2021.
//

#ifndef AISDTEST_TASK_H
#define AISDTEST_TASK_H

#include "_AiSD7.h"

//Строит дерево в глубину.
void BuildTree(Tree* T, Pmem Pbuf, char* s);

void PrintCharTree(Tree T, Pmem Pbuf);

//Определяет количество вершин в дереве T на n-ом уровне.
unsigned CalcLevel(Tree T, Pmem Pbuf, unsigned n);
void CalcLevel_inner(Tree T, Pmem Pbuf, unsigned n, int i, int k);

//Выводит выводит все пути от листьев до корня (в i-ю строку
//вывода – i-ый путь)
void WriteWays(Tree T, Pmem Pbuf);
void WriteWays_innner(Tree T, Pmem Pbuf, int i);

#endif //AISDTEST_TASK_H
```

task.c

```
//
// Created by настя буйвало on 16/11/2021.
//
#include <stdlib.h>
#include <stdio.h>
#include "task.h"

#define Index2 1001

const int TreeOk = 0;
const int TreeNotMem = 1;
const int TreeUnder = 2;

short TreeError = TreeOk;


//Строит дерево в глубину.
void BuildTree(Tree* T, Pmem Pbuf, char* s)
{
```

```
    static int i = 0;
    static int rc = 0; //глубина рекурсии
    if(s[i] != '.' && s[i] != '\0'){
        char *wr = (char*)calloc(2, sizeof(char));
        wr[0] = s[i];

        CreateRoot(*T, Pbuf); //создаем корень
        WriteDataTree(*T, wr, Pbuf, sizeof(char)*2, Index); //записываем данные в
корень
        rc++;
        i++;

        if(s[i] != '.' && s[i] != '\0'){ //если есть левый сын, то начинаем строить
корень в нем
            CreateRoot(&((*Pbuf)[**T].LSon), Pbuf);
            Tree t1 = &((*Pbuf)[**T].LSon);
            BuildTree(&t1, Pbuf, s);
        }
        i++;

        //если есть правый сын, то начинаем строить корень в нем
        if(s[i] != '.' && s[i] != '\0'){
            CreateRoot(&((*Pbuf)[**T].RSon), Pbuf);
            Tree t2 = &((*Pbuf)[**T].RSon);
            BuildTree(&t2, Pbuf, s);
        }
        rc--;
    }
}

void PrintCharTree(Tree T, Pmem Pbuf)
{
    if(!IsEmptyTree(T, Pbuf)){
        char *s = (char*)calloc(2, sizeof(char));
        ReadDataTree(T, s, Pbuf, sizeof(char)*2);
        printf("%s", s);
    }
    //если не лист - идем дальше
    if(!((*Pbuf)[*T].LSon == Index2 && (*Pbuf)[*T].RSon == Index2))
    {
        if((*Pbuf)[*T].LSon != Index2) {
            MoveToRSon(T, (*Pbuf)[*T].LSon);
            PrintCharTree(&((*Pbuf)[*T].LSon), Pbuf);
        }
        if((*Pbuf)[*T].RSon != Index2) {
            MoveToRSon(T, &((*Pbuf)[*T].RSon));
            PrintCharTree(&((*Pbuf)[*T].RSon), Pbuf);
        }
    }
}



//Определяет количество вершин в дереве T на n-ом уровне.
void CalcLevel_inner(Tree T, Pmem Pbuf, unsigned n, int i, int k)
{
    if(T != NULL && *T > 0 && *T < Index){
        if(!IsEmptyTree(&T, Pbuf)){
            if(i == n)
                k++;
            CalcLevel_inner(&((*Pbuf)[*T]->LSon), Pbuf, n,  i+1, k);
            CalcLevel_inner(&((*Pbuf)[*T]->RSon), Pbuf, n, i+1, k);
```

```
        }
        else
            return;
    }
    else
        TreeError = TreeUnder;
}

unsigned CalcLevel(Tree T, Pmem Pbuf, unsigned n)
{
    int k = 0;
    CalcLevel_inner(T, Pbuf, n, 0, k);
    return k;
}

void output_arr(int* a, int n)
{
    for(int i = 0; i < n; i++)
        printf("%d", a[i]);
}

//Выводит выводит все пути от листьев до корня (в i-ю строку
// вывода — i-ый путь)
void WriteWays_innner(Tree T, Pmem Pbuf, int i)
{
    static int a[Index] = {0};
    a[i] = *T;
    if(T != NULL && *T > 0 && *T < Index){
        if((*Pbuf)[*T]->LSon != Index2 || (*Pbuf)[*T]->RSon != Index2){
            WriteWays_innner(&((*Pbuf)[*T]->LSon), Pbuf, i+1);
            WriteWays_innner(&((*Pbuf)[*T]->RSon), Pbuf, i+1);
        }
        else {
            output_arr(a, i);
            return;
        }
    }
    else
        TreeError = TreeUnder;
}

void WriteWays(Tree T, Pmem Pbuf)
{
    WriteWays_innner(T, Pbuf, 0);
}
```

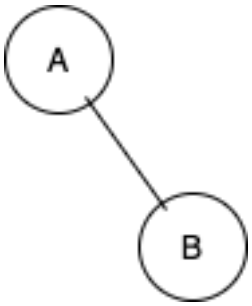
Тестовые данные:

Входные данные	Дерево в глубину	Количество вершин на 1 уровне	Пути от листьев к корню
A..	A	0	A
A.B..	AB	1	BA
ABCD..E..F..G.HI..J..	ABCDEFGHIJ	2	DCBA ECBA FBA IHGA JHGA

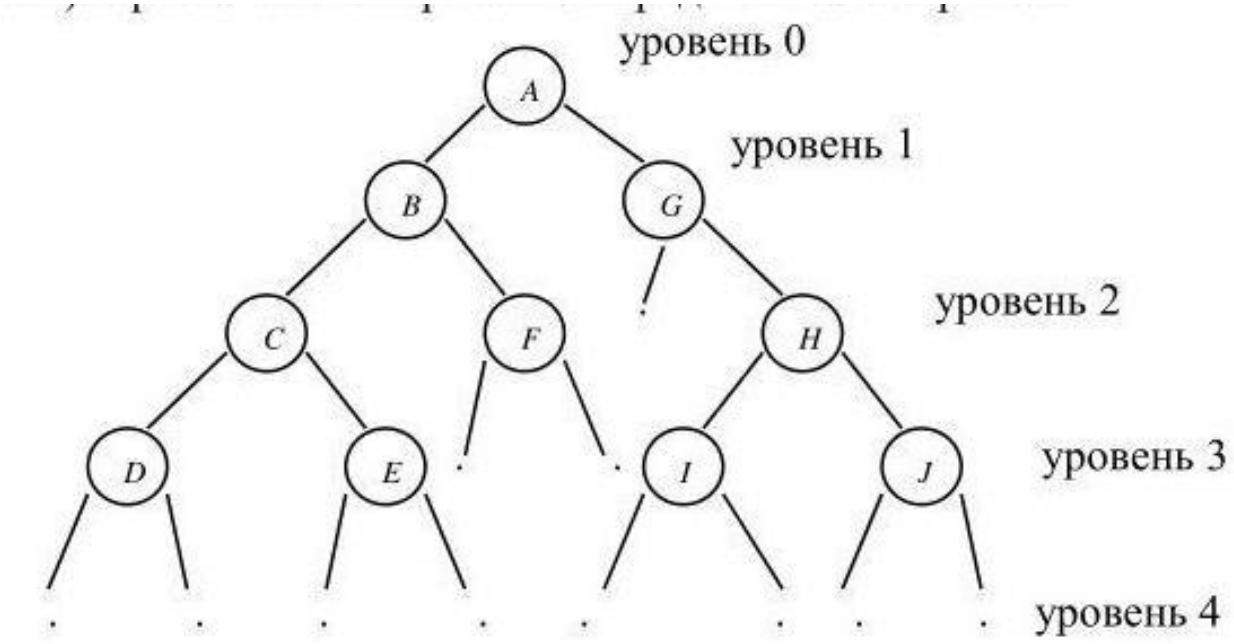
Тестовые данные представим графически:

1 тест

2 тест



3 тест (пример из методички)



Результаты работы:

```
  ABCD..E..F..G.HI..J..
 Дерево в глубину:  ABCDEFGHIJ
```

```
Введите представление: A.B..
Дерево в глубину: AB
```

```
 Введите представление:
 A..
 Дерево в глубину: A
```

```
A..
количество вершин на 1 уровне: 0
```

```
 A.B..
 количество вершин на 1 уровне: 1

 Process finished with exit code 0
```

```
ABCD..E..F..G.HI..J..
количество вершин на 1 уровне: 2
```

```
Введите представление: A..
Пути : A
```

```
Введите представление:
A.B..
Пути : BA
```

```
Введите представление: ABCD..E..F..G.HI..J..
Пути : DCBA
ECBA
FBA
IHGA
JHGA
```

Вывод:

Во время выполнения лабораторной работы получены навыки реализации и использования структуры данных «дерево» соответствующего формата (7 вариант), изучены виды представления СД, на базе формата выполнены задания согласно варианту.