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:
«Структуры данных типа «дерево» (Pascal/C)»

Выполнил: ст. группы ПВ-211

Чувилко Илья Романович

Проверил:

Синюк Василий Григорьевич

Белгород 2022 г.

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.

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.

6 Вариант модуля

Элементы дерева находятся в массиве MemTree, расположенном в статической памяти. Базовый тип — произвольный. «Свободные» элементы массива объединяются в список (ССЭ), на начало которого указывает правый сын первого элемента массива. В массиве MemTree могут располагаться несколько деревьев.

Содержимое заголовочного файла:

```
#ifndef CODE_TREE_H
#define CODE_TREE_H

#include "stdio.h"
#include "stdlib.h"

#define capacity 1000

typedef int BaseType;
```

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

typedef unsigned int PtrEl;

struct element {
    BaseType data;
    PtrEl LSon;
    PtrEl RSon;
} typedef element;

typedef PtrEl *Tree;
unsigned Size;
element MemTree[capacity];


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

// Создание корня
void CreateRoot(Tree *T, BaseType x);

// Возвращает 1, если у элемента под номером index есть левый сын.
// 0 - в ином случае
int IsLSon(Tree *T, unsigned index);

// Возвращает 1, если у элемента под номером index есть правый сын.
// 0 - в ином случае
int IsRSon(Tree *T, unsigned index);

// Возвращает 1, если БД пустое. 0 - в ином случае
int IsEmptyTree(Tree *T);

// Возвращает 1, если в массиве нет свободных элементов,
// 0 — в противном случае
int EmptyMem(Tree *T);

// возвращает номер свободного элемента и исключает его из ССЭ
unsigned int NewMem(Tree *T);

//запись данных
void WriteDataTree(Tree *T, BaseType x);

void WriteDataTree_(Tree *T, BaseType x, unsigned index);

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

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

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

// связывает все элементы массива в список свободных элементов
void InitMem(Tree *T);

//удаление листа
```

```
void DelTree(Tree *T);

// делает n-й элемент массива свободным и включает его в ССЭ
void DisposeMem(unsigned n);

#endif //CODE_TREE_H
```

Содержимое исполняемого файла:

```
#include "Tree.h"

// Связывает все элементы массива в список свободных элементов
void InitMem(Tree *T) {
    for (int i = 0; i < Size; i++)
        MemTree[i].RSon = i + 1;
}

// Инициализация БД
void InitTree(Tree *T, unsigned size) {
    MemTree[0].RSon = 1;
    Size = size;
    InitMem(T);
}

// Возвращает 1, если в массиве нет свободных элементов,
// 0 — в противном случае
int EmptyMem(Tree *T) {
    return MemTree[0].RSon == Size;
}

// Возвращает номер свободного элемента и исключает его из ССЭ
unsigned int NewMem(Tree *T) {
    if (EmptyMem(T)) {
        TreeError = TreeNotMem;
        exit(TreeNotMem);
    }

    unsigned freeElementIndex = MemTree[0].RSon;
    MemTree[0].RSon = MemTree[freeElementIndex].RSon;
    return freeElementIndex;
}

// Создает корень дереву T, со значением x
void CreateRoot(Tree *T, BaseType x) {
    unsigned index = NewMem(T);
    element el = MemTree[index];
    el.data = x;
    el.RSon = 0;
    el.LSon = 0;
}

// Возвращает 1, если у элемента под номером index есть левый сын.
// 0 - в ином случае
int IsLSon(Tree *T, unsigned index) {
    return MemTree[index].LSon != 0;
}

// Возвращает 1, если у элемента под номером index есть правый сын.
// 0 - в ином случае
int IsRSon(Tree *T, unsigned index) {
    return MemTree[index].RSon != 0;
```

```
}

// Возвращает 1, если БД пустое. 0 - в ином случае
int IsEmptyTree(Tree *T) {
   return MemTree[0].RSon == 1;
}

// Записывает элемент x, в дерево Tree
void WriteDataTree(Tree *T, BaseType x) {
   if (IsEmptyTree(T)) {
      CreateRoot(T, x);
   } else if (MemTree[1].data < x) {
      if (IsLSon(T, 1)) {
         WriteDataTree_(T, x, MemTree[1].LSon);
      } else {
         MemTree[1].LSon = MemTree[0].RSon;
         CreateRoot(T, x);
      }
   } else {
      if (IsRSon(T, 1)) {
         WriteDataTree_(T, x, MemTree[1].RSon);
      } else {
         MemTree[1].RSon = MemTree[0].RSon;
         CreateRoot(T, x);
      }
   }
}

// Рекурсивно погружется для записи элемента x
void WriteDataTree_(Tree *T, BaseType x, unsigned index) {
   if (MemTree[index].data < x) {
      if (IsLSon(T, index)) {
         WriteDataTree_(T, x, MemTree[index].LSon);
      } else {
         MemTree[index].LSon = MemTree[0].RSon;
         CreateRoot(T, x);
         return;
      }
   } else {
      if (IsRSon(T, index)) {
         WriteDataTree_(T, x, MemTree[index].RSon);
      } else {
         MemTree[index].RSon = MemTree[0].RSon;
         CreateRoot(T, x);
         return;
      }
   }
}

// перейти к левому сыну, где T —адрес ячейки, содержащей адрес текущей
// вершины, TS — адрес ячейки, содержащей адрес корня левого
// поддерева(левого сына)
void MoveToLSon(Tree *T, Tree *TS) {
   **TS = MemTree[**T].LSon;
}

// Удаляет дерево
void DelTree(Tree *T) {
   Size = 0;
}

// делает n-й элемент массива свободным и включает его в ССЭ
void DisposeMem(unsigned n) {
```

```
    MemTree[n].RSon = MemTree[0].RSon;
    MemTree[0].RSon = n;
}
```

3 Вариант задачи

а) Procedure **BildBalansTree**(const M: T_mas; var T:Tree); Строит упорядоченное сбалансированное дерево T. M — упорядоченный массив

```
// Строит сбалансированное дерево T на отсортированном массиве a размера size
void BuildBalansTree(int *a, unsigned size, Tree *T) {
    unsigned mid = size / 2;
    WriteDataTree(T, a[mid]);
    if (mid > 0) {
        BuildBalansTree(a + mid, size - mid, T);
        BuildBalansTree(a, size - mid, T);
    }
}
```

б) Function **HTree**(T:Tree):byte; Вычисляет высоту дерева.

```
// копирует массив a в массив b
void copyArray(unsigned const *const a, unsigned *b) {
    for (int i = 0; i < a[0] + 1; i++)
        b[i] = a[i];
}

// Возвращает высоту дерева
unsigned HTree(Tree *T) {
    unsigned level = 0;
    unsigned nLevel[Size + 1];

    nLevel[0]++;
    nLevel[1] = **T;
    while (nLevel[0] != 0) {
        unsigned tmp[Size + 1];
        for (int i = 1; i < nLevel[0] + 1; i++) {
            tmp[tmp[0]++] = MemTree[nLevel[i]].LSon;
            tmp[tmp[0]++] = MemTree[nLevel[i]].RSon;
        }
        copyArray(tmp, nLevel);
        level++;
    }

    return level;
}
```

в) Procedure **WriteTree**(T:Tree); Выводит дерево по уровням (в i-ю строку вывода — вершины i-го уровня).

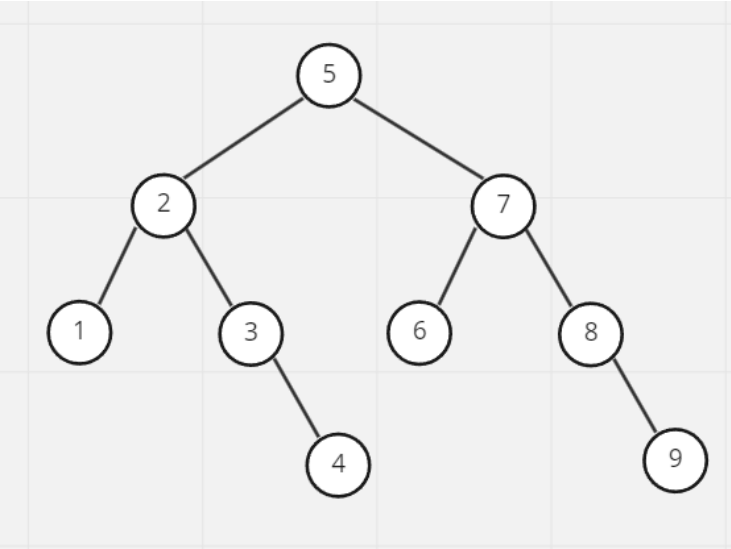
```
// Выводит массив
void outputArray(unsigned *a) {
    for (int i = 1; i < a[0] + 1; i++)
        printf("%d ", a[i]);
    printf("\n");
}

// Выводит дерево T
void WriteTree(Tree *T) {
    unsigned level = 0;
    unsigned nLevel[Size + 1];
```

```
    nLevel[0]++;
    nLevel[1] = **T;
    while (nLevel[0] != 0) {
        unsigned tmp[Size + 1];
        for (int i = 1; i < nLevel[0] + 1; i++) {
            tmp[tmp[0]++] = MemTree[nLevel[i]].LSon;
            tmp[tmp[0]++] = MemTree[nLevel[i]].RSon;
        }

        printf("level %d: ", level);
        outputArray(nLevel);
        copyArray(tmp, nLevel);
        level++;
    }
}
```

Результат работы программы:

```
"C:\BGTU\BGTU\ASD\Lab 7\Code\cmake-build-debug\Code.exe"
10
0 1 2 3 4 5 6 7 8 9
level 0: 5
level 1: 2 7
level 2: 0 1 3 6 8
level 3: 4 9

Process finished with exit code 0
```

Граф на основе этих данных:



Вывод: Изучил структуру данных типа бинарное дерево и научился реализовывать и программно использовать.