Деревья (на основании заданий от Рогачевой Е.В.)

Некоторые соглашения.
Это занятие посвящено построению и работе с бинарными деревьями. В задачах этого занятия будем использовать объявления, приведенные в примере.

Теоретические положения.
Бинарное дерево – это конечное множество узлов, которое либо является пустым, либо состоит и корня и двух непересекающихся бинарных деревьев, которые называются левым и правым поддеревьями данного корня.

Пример

```
program SimpleTree;

{$APPTYPE CONSOLE}

uses
  SysUtils;

const
  LEFT=1;
  RIGHT=2;
  INROOT=0;

type

  TNode = class
    info: string;
    key: integer;
    parent: TNode;
    leftNode: TNode;
    rightNode: TNode;
    constructor create; overload;
    constructor create(info: string; key: integer);overload;
    constructor create(info: string; key: integer; parent: TNode);overload;
    function compareTo(n: TNode): boolean;
  end;

  TTree = class
      root: TNode;
      constructor create;
      function getRoot: TNode;
      procedure printKey(n: TNode);
      procedure preOrder(n: TNode);
      procedure postOrder(n: TNode);
      procedure inOrder(n:TNode);
      procedure insertLeafInTree(inf: string; k: integer);
      function search(n: TNode; k: integer):string;
      function delFromTree(k: integer): string;
      procedure R_insertLeafInTree(inf: string; k: integer);
      procedure R_insert(inf: string; k: integer; parent: TNode);
      function R_search(n: TNode; k: integer): string;
      function R_delFromTree(k: integer): string;
      function R_delete(k: integer; parentNode: TNode): string;
      function delThisNode(parent: TNode; direct: integer):string;
      function rightest(n: TNode): TNode;
  end;
```

```
//****************** Реализация методов класса "УЗЕЛ" ************************

constructor TNode.create;
begin
  info:='';
  key:=-32565;
  leftNode:=nil;
  rightNode:=nil;
  parent:=nil;
end;

constructor TNode.create(info: string; key: integer);
begin
  self.info:=info;
  self.key:=key;
  leftNode:=nil;
  rightNode:=nil;
end;

constructor TNode.create(info: string; key: integer; parent: TNode);
begin
  self.info:=info;
  self.key:=key;
  leftNode:=nil;
  rightNode:=nil;
  self.parent:=parent;
end;

function TNode.compareTo(n: TNode): Boolean;
begin
  result:=(key>n.key);
end;

//****************** Реализация методов класса "ДЕРЕВО" ************************
constructor TTree.create;
begin
  root:=nil;
end;

function TTree.getRoot: TNode;
begin
  result:= root;
end;

procedure TTree.printKey(n: TNode);
begin
  writeln(n.key);
end;

procedure TTree.preOrder(n: TNode);
begin
  if (n<>nil) then
  begin
    printKey(n);
    preOrder(n.leftNode);
    preOrder(n.rightNode);
  end;
end;

procedure TTree.postOrder(n: TNode);
begin
```

```
  if (n<>nil) then
  begin
      postOrder(n.leftNode);
      postOrder(n.rightNode);
      printKey(n);
  end;
end;

procedure TTree.inOrder(n: TNode);
begin
  if (n<>nil) then
  begin
    inOrder(n.leftNode);
    printKey(n);
    inOrder(n.rightNode);
  end;
end;

procedure TTree.insertLeafInTree(inf: string; k: integer);
var
  currentNode, parentNode, newNode: TNode;
begin
  currentNode:=root;
  parentNode:=nil;

  while (currentNode<>nil) do
  begin
    parentNode:=currentNode;

    if (k<currentNode.key) then
          currentNode:=currentNode.leftNode
     else if (k>currentNode.key) then
          currentNode:=currentNode.rightNode
          else // if k = currentNode.key
          begin
            currentNode.info:=inf;
            exit;
          end;
  end;

  newNode:=TNode.create(inf,k,parentNode);
  if (parentNode=nil) then // создаем корень
      root:=newNode
      else if (k<parentNode.key) then parentNode.leftNode:=newNode
      else parentNode.rightNode:=newNode;
end;

function TTree.search(n: TNode; k: integer): string;
var
  currentNode: TNode;
begin
  currentNode:=root;
  while ((currentNode<>nil)and(currentNode.key<>k)) do
    if (k<currentNode.key) then currentNode:=currentNode.leftNode
    else // k > result.key
    currentNode:=currentNode.rightNode;
  if (currentNode=nil) then Result:='No such element!'
  else Result:=currentNode.info;
end;

function TTree.delFromTree(k: integer): string;
```

```
  var
    currentNode, parentNode, rightestNode: TNode;
  begin
    currentNode:=root;
    parentNode:=nil;

    while ((currentNode<> nil) and (currentNode.key<>k)) do
    begin
      parentNode:= currentNode;
      if (k<currentNode.key) then currentNode:=currentNode.leftNode
      else if (k>currentNode.key) then currentNode:=currentNode.rightNode;
    end;

    if (currentNode=nil) then begin result:='No such node!'; exit; end;
    // k = currentNode.key
    result:=currentNode.info;
    if (currentNode.rightNode=nil) then // нет правого поддерева
      if (parentNode=nil) then // это корень
      begin
        root:=currentNode.leftNode;
        if (currentNode.leftNode<>nil) then
            currentNode.leftNode.parent:=nil
         else
        if (parentNode.leftNode=currentNode) then
        begin
          parentNode.leftNode:=currentNode.leftNode;
          if (currentNode.leftNode<>nil) then currentNode.leftNode.parent:=
parentNode;
        end
        else // это был правый узел
        begin
          parentNode.rightNode:=currentNode.leftNode;
          currentNode.leftNode.parent:=parentNode;
        end;
     end // rightNode = nil
     else if (currentNode.leftNode=nil) then //нет левого поддерева
     begin
      if (parentNode=nil) then
      begin
        root:=currentNode.rightNode;
        if (currentNode.rightNode<>nil) then currentNode.rightNode.parent:=nil;
      end
      else if (parentNode.leftNode=currentNode) then
      begin
        parentNode.leftNode:=currentNode.rightNode;
        if (currentNode.rightNode<>nil) then
currentNode.rightNode.parent:=parentNode;
      end
      else
      begin
        parentNode.rightNode:=currentNode.rightNode;
        if (currentNode.rightNode<>nil)  then
currentNode.rightNode.parent:=parentNode;
      end;
     end
     else // есть оба поддерева - тогда ищем самый правый узел в левом поддереве
     begin
        rightestNode:=currentNode.leftNode;
        parentNode:=currentNode;
        while (rightestNode.rightNode<>nil) do
        begin
```

```
          parentNode:=rightestNode;
          rightestNode:=rightestNode.rightNode;
        end;
        // копируем
        currentNode.info:=rightestNode.info;
        currentNode.key:=rightestNode.key;
        // удаляем ставший лишним узел
        if (parentNode=currentNode) then // никуда не уходили
        begin
          parentNode.leftNode:=rightestNode.leftNode;
          if (rightestNode.leftNode<>nil) then
rightestNode.leftNode.parent:=parentNode;
        end
        else
        begin
          parentNode.rightNode:=rightestNode.leftNode;
          if (rightestNode.leftNode<>nil) then
rightestNode.leftNode.parent:=parentNode.rightNode;
        end;
     end;
  end;


  //********************** рекурсивные *************************

  procedure TTree.R_insertLeafInTree(inf: string; k: integer);
  var
    currentNode: TNode;
  begin
    currentNode:=root;
    if (currentNode=nil) then // корень не существует - создаем его
    root:=TNode.create(inf, k, nil)
    else R_insert(inf,k,root);
  end;


  procedure TTree.R_insert(inf: string; k: integer; parent: TNode);
  var
    n: TNode;
  begin
    if (k<parent.key) then
    begin
      if (parent.leftNode=nil) then
      begin
        // если левое поддерево пусто
        n:= TNode.create(inf,k,parent);
        parent.leftNode:=n;
      end
      else R_insert(inf,k,parent.leftNode);
    end //k<parent.key
    else if (k>parent.key) then
    begin
      if (parent.rightNode=nil) then
      begin
        // если справа от узла ничего нет
        n:=TNode.create(inf,k,parent);
        parent.rightNode:=n;
      end
      else R_insert(inf,k,parent.rightNode);
    end  //k>parent.key
    else // k = parent.key
```

```
  parent.info:=inf;
end;


function TTree.R_search(n: TNode; k: integer): string;
begin
  if (n=nil) then
  begin
    result:='No such element!';
    exit;
  end
  else if (k<n.key) then R_search(n.leftNode,k)
  else if (k>n.key) then R_search(n.rightNode,k)
  else
  result:=n.info;
end;

function TTree.R_delFromTree(k: integer): string;
var
  currentNode: TNode;
begin
  currentNode:=root;
  result:=R_delete(k,currentNode);
end;

function TTree.R_delete(k: integer; parentNode: TNode): string;
begin
  if (parentNode= nil) then
  begin
    result:='No such element';
    exit;
  end;
  if (k<parentNode.key) then
  begin
    if ((parentNode.leftNode<>nil)and (k=parentNode.leftNode.key)) then
    result:=delThisNode(parentNode,LEFT)
    else result:=R_delete(k,parentNode.leftNode);
  end// k < parent.key
  else if (k > parentNode.key) then
  begin
    if ((parentNode.rightNode<>nil)and(k=parentNode.rightNode.key)) then
    result:=delThisNode(parentNode,RIGHT)
    else result:=R_delete(k,parentNode.rightNode);
  end // k > parent.key
  else // k == parent.key
  // а это возможно только в случае, когда parent = root
  Result:=delThisNode(parentNode,INROOT);
end;

function TTree.delThisNode(parent: TNode; direct: integer):string;
var
  deleteNode, rightestNode: TNode;
begin
  if (direct= LEFT) then deleteNode:=parent.leftNode
  else if (direct = RIGHT) then deleteNode:=parent.rightNode
  else // direct == INROOT
  deleteNode:=parent;
  Result:=deleteNode.info;

  if (deleteNode.rightNode=nil) then // нет правого поддерева
  begin
```

```
  if (direct=INROOT) then // это корень
  begin
    root:=deleteNode.leftNode;
    if (deleteNode.leftNode<>nil) then
    deleteNode.leftNode.parent:=nil;// у корня нет предка
  end
  else if (direct=LEFT) then
  begin
    parent.leftNode:=deleteNode.leftNode;
    if (deleteNode.leftNode<>nil) then
    deleteNode.leftNode.parent:=parent;
  end
  else // это был правый узел direct = RIGHT
  begin
    parent.rightNode:= deleteNode.leftNode;
    if (deleteNode.leftNode<>nil) then
    deleteNode.leftNode.parent:=parent;
  end;
end // нет правого поддерева
else if (deleteNode.leftNode=nil) then // нет левого поддерева
begin
  if (direct=INROOT) then
  begin
    root:=deleteNode.rightNode;
    if (deleteNode.rightNode<>nil) then
    deleteNode.rightNode.parent:=nil;
  end
  else if (direct=LEFT) then
  begin
    parent.leftNode:=deleteNode.rightNode;
    if (deleteNode.rightNode<>nil) then
    deleteNode.rightNode.parent:=parent;
  end
  else // direct == RIGHT
  begin
    parent.rightNode:=deleteNode.rightNode;
    if (deleteNode.rightNode<>nil) then
    deleteNode.rightNode.parent:=parent;
  end;
end // нет левого поддерева
else // есть оба поддерева - тогда ищем самый правый узел в левом поддереве
begin
  rightestNode:=rightest(deleteNode.leftNode);
  parent:=rightestNode.parent;
  // копируем
  deleteNode.info:=rightestNode.info;
  deleteNode.key:=rightestNode.key;
  // удаляем ставший лишним лист
  if (parent = deleteNode) then // никуда не уходили
  begin
    parent.leftNode:=rightestNode.leftNode;
    if (rightestNode.leftNode<>nil) then
    rightestNode.leftNode.parent:=parent;
  end
  else
  begin
    parent.rightNode:=rightestNode.leftNode;
    if (rightestNode.leftNode<>nil) then
    rightestNode.leftNode.parent:=parent;
  end;
end; // есть оба поддерева - тогда ищем самый правый узел в левом поддереве
```

```
  end;

   function TTree.rightest(n: TNode):TNode;
   begin
    result:=n;
    if (Result.rightNode<>nil) then
    Result:=rightest(Result.rightNode);
   end;

//основная программа

var
  st: TTree;
  inf, s: string;
  choice, order: char;
  k: integer;
begin
  st := TTree.create;

  writeln('Create tree? yes/no:');
  while (true) do
  begin
    readln(s);
    if (s='no') then break;
    writeln('Enter info field:');
    readln(inf);
    writeln('Enter key');
    readln(k);
    st.insertLeafInTree(inf,k);
    writeln('Continue? yes/no:');
  end;

  writeln('Work with tree? yes/no:');
  while (true) do
  begin
    readln(s);
    if (s='no') then break;
    writeln('Enter i - to insert');
    writeln('      d - to delete');
    writeln('      s - to search');
    writeln('      p - to print');
    readln(choice);

    case choice of
      'i': begin
             writeln('Enter info field:');
             readln(inf);
             writeln('Enter key');
             readln(k);
             st.insertLeafInTree(inf,k);
           end;
       'd': begin
             writeln('Enter key');
             readln(k);
             inf:=st.delFromTree(k);
             writeln('Node ',inf,' is deleted.');
            end;
       's': begin
             writeln('Enter key');
             readln(k);
             s:=st.search(st.getRoot,k);
```

```
            writeln(s);
          end;
      'p': begin
            writeln('Enter u - to go from up to down');
            writeln('      d - to go from down to up');
            writeln('      i - to go from left to right');
            readln (order);

            case order of
                'u': st.preOrder(st.getRoot);
                'd': st.postOrder(st.getRoot);
                'i': st.inOrder(st.getRoot);
                else writeln('Unknown order!');
            end;
          end; // case 'p'
    end; // case choice
    writeln('Continue? yes/no:');
  end;//while
  readln;
end.
```

Задачи

Во всех задачах подразумевается, что функция является методом либо определенного выше класса

1. Опишите логическую функцию same(T), определяющую, есть ли в дереве T хотя бы два одинаковых элемента.
2. Напишите процедуру удаления (из существующего бинарного дерева) всех отрицательных элементов.
3. Доработайте класс таким образом, чтобы можно было:
    a. формировать дерево из текстового файла (формат файла предложите самостоятельно)
    b. переформировывать дерево на основе команд, описанных в текстовом файле (формат файла предложите самостоятельно – например, можно в каждой строке записывать символ, обозначающий операцию, а через пробел – данные, над которыми эта операция выполняется).
4. Дан текстовый файл, в котором присутствуют только буквы и пробельные символы. Назовем словом последовательность символов, внутри которой не содержится пробельных символов. Опишите дерево, подходящее для определения частоты вхождения (отношение количества вхождений к общему количеству слов в тексте) каждого слова в текст (подумайте, какого типа должно быть ключевое поле, какого – информационное). Считайте, что длина слова не может превосходить 255 символов. Опишите функцию, которая:
    a. определяет частоту вхождения для заданного слова
    b. определяет количество вершин в дереве, содержащих слова, начинающиеся на одну и ту же букву
    c. определяет количество вершин в дереве, содержащих слова, являющиеся палиндромами
    d. определяет количество вершин в дереве, содержащих слова, в состав которых входят все гласные буквы английского языка.
5. Переопределите функцию добавления элемента в дерево таким образом, чтобы в нем могли содержаться элементы с одинаковыми ключами (например, при равенстве добавляемого ключа ключу некоторого узла происходил бы переход к левому поддереву этого узла). Опишите рекурсивную функцию, подсчитывающую число вхождений в дерево элемента с заданным ключом.

6. Считайте, что информационное поле элемента дерева – вещественное число. Опишите рекурсивную функцию, вычисляющую сумму информационных полей всех элементов дерева (если дерево пусто, выводите сообщение об этом).
7. Считайте, что информационное поле элемента дерева – вещественное число. Опишите рекурсивную функцию, которая отыскивает максимальное значение информационного поля (если дерево пусто, выводите сообщение об этом).
8. Опишите рекурсивную функцию, выводящую на печать информационные поля и ключи всех листьев дерева.
9. Опишите рекурсивную функцию, определяющую число ветвей в самом длинном из путей от корня дерева до листьев.
10. Опишите рекурсивную функцию, подсчитывающую число вершин на заданном уровне (корень считается находящимся на нулевом уровне).
11. Опишите рекурсивную функцию, проверяющую на равенство два дерева
12. Опишите нерекурсивную функцию, проверяющую на равенство два дерева
13. Опишите функцию, которая строит копию данного дерева
14. Опишите логическую функцию, определяющую, есть ли в дереве хотя бы два элемента с одинаковыми ключами (см. задачу 4).
15. Двоичное дерево считается идеально сбалансированным, если для каждой его вершины количество вершин в левом и правом поддеревьях различается не более чем на 1. Нужно написать функцию проверки идеальной сбалансированности двоичного дерева.
16. В текстовом файле заданы N целых чисел в двоичной системе счисления (M бит каждое). Построить двоичное дерево, в котором числам соответствуют листья дерева, а путь по дереву определяет значение информационного поля этого листа. Информационные поля внутренних узлов желательно оставить пустыми. Указание: сначала попробуйте нарисовать эту структуру.