Лекция 16.
Бинарное дерево

Дерево – это **связный граф** без циклов. **Граф** – это совокупность вершин и связывающих их ребер. В **связном графе** можно найти путь по ребрам от любой вершины к любой другой. Пример связного графа – схема Московского метрополитена. В нём много циклов, и речь не только о кольцевой линии.

Бинарное дерево – иерархическая структура данных, в которой у каждой **вершины** (кроме **корня дерева**) есть одна родительская вершина и у каждой вершины не более двух поддеревьев (**ветвей дерева**) – левое и правое. Корень, как ни странно, всегда изображается сверху, а **ветви** идут вниз или в стороны, и скорее напоминают разветвления корня. У **листьев** нет дочерних вершин, только родительская. Любая часть дерева в свою очередь тоже является деревом.

Мы уже сталкивались с деревьями – полными деревьями рекурсивных вызовов, когда изучали рекурсию. Там в каждой вершине дерева находился вызов рекурсивной функции с ее именем и параметрами.

Бинарные деревья

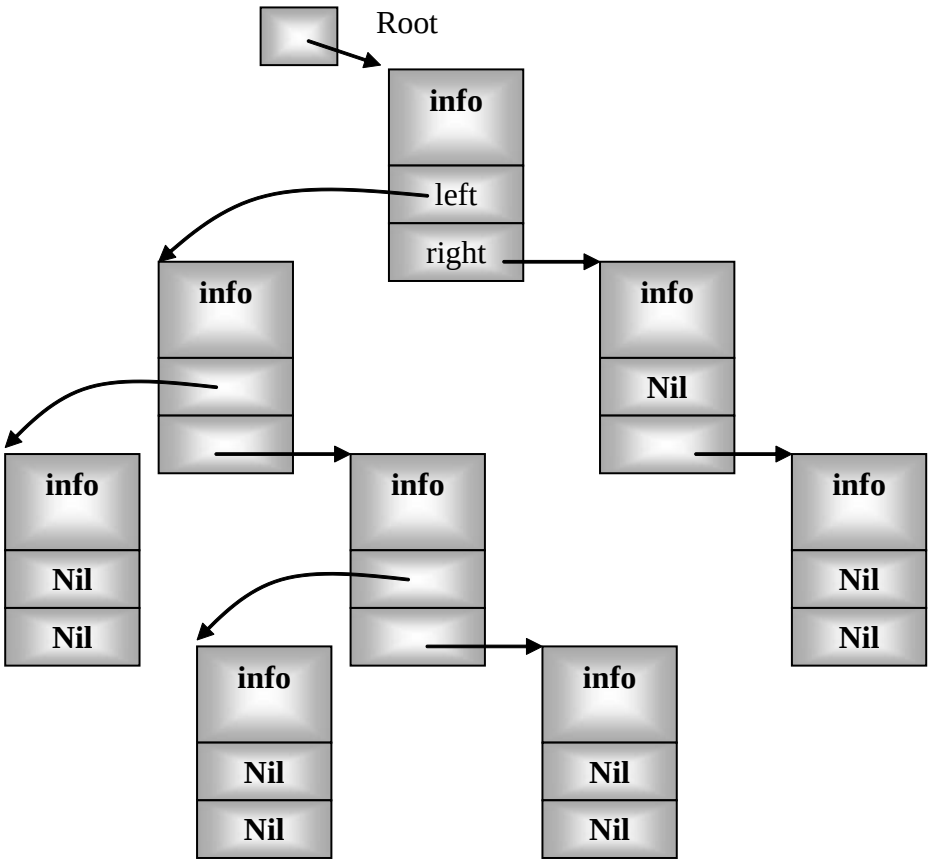


Рис.16.1

Данные:

Адрес корня дерева.

Допустимые операции:

Создать дерево из 1 листа (элемент, без поддеревьев);

Добавить лист/вершину/корень дерева (часто в соответствии с определенным на дереве порядком);

Удалить лист/вершину/фрагмент дерева;

Проверка на отсутствие элементов;

Очистить дерево (удалить все элементы);

Узнать значение информационной части текущего узла дерева.

Для работы с элементами дерева используется обход дерева в глубину, в ширину, симметричный... Например, дерево на рис 16.3 можно обойти следующими путями:

10, 5, 1, 7, 6, 9, 10, 13 (начиная с корня в глубину слева направо - ЦЛП (центр-левое-правое));

10, 5, 10, 1, 7, 13, 6, 9 (начиная с корня по ярусам – в ширину);

1, 6, 9, 7, 5, 13, 10, 10 (с самого левого листа, узлы после листьев - ЛПЦ);

1, 5, 6, 7, 9, 10, 10, 13 (с самого левого листа проходя через узлы - ЛЦП – видно **упорядоченность**)

Дерево может быть построено, начиная с любого листа, в зависимости от нужд задачи. Например, при синтаксическом разборе выражения $\frac{1}{6+9}+5$ строится дерево, показанное ниже.

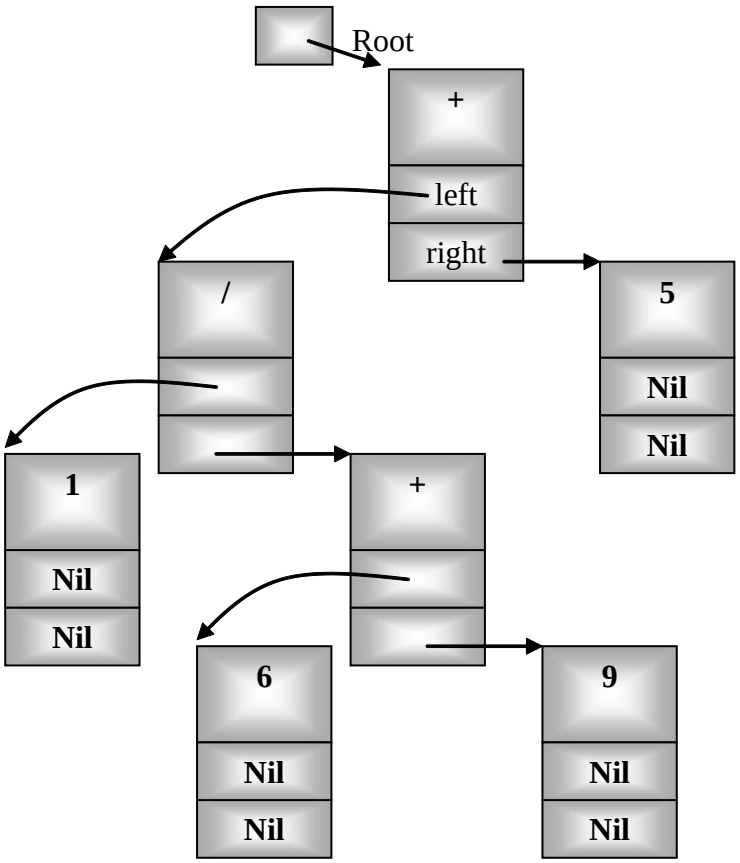


Рис. 16.2. Синтаксический разбор выражения

На следующем рисунке приведен пример дерева с упорядоченными по возрастанию вершинами.

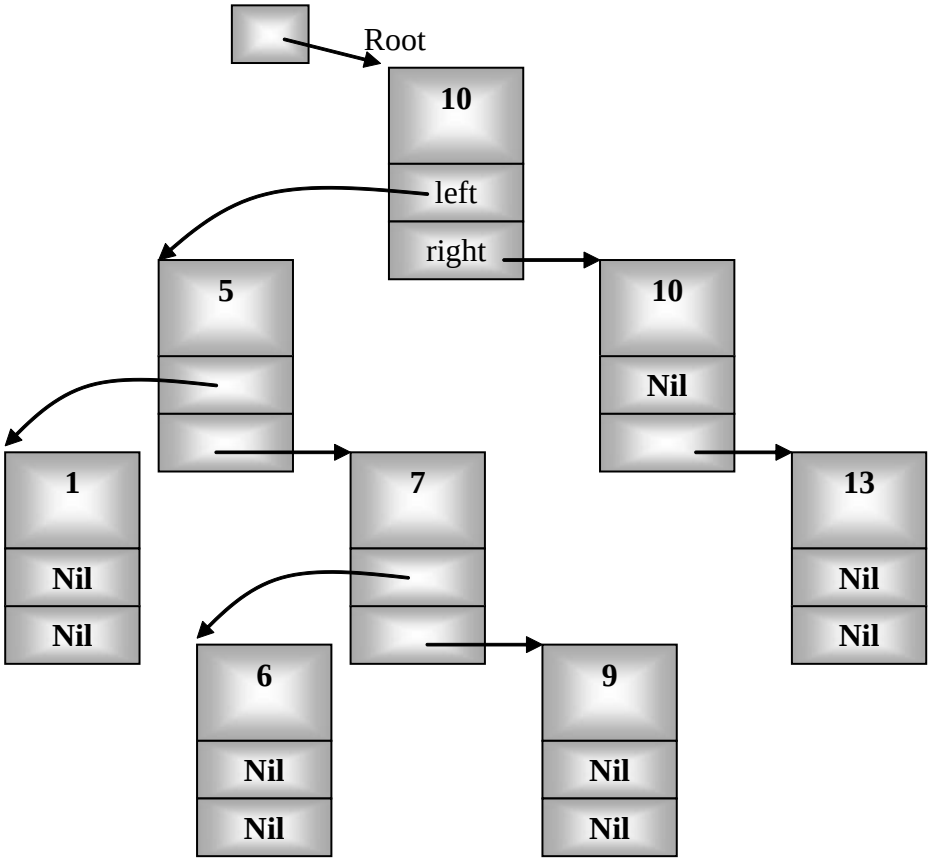
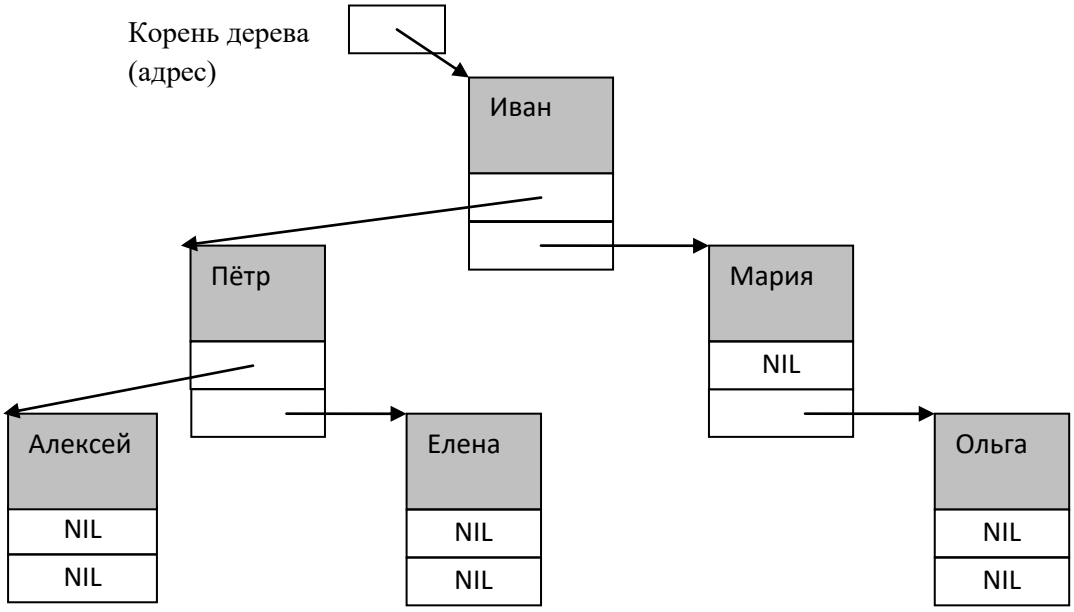


Рис. 16.3 Упорядоченные по возрастанию узлы (чем правее, тем больше)



Генеалогическое дерево Ивана.

Отец: Пётр, родителями которого являются Алексей и Елена (дедушка и бабушка Ивана);

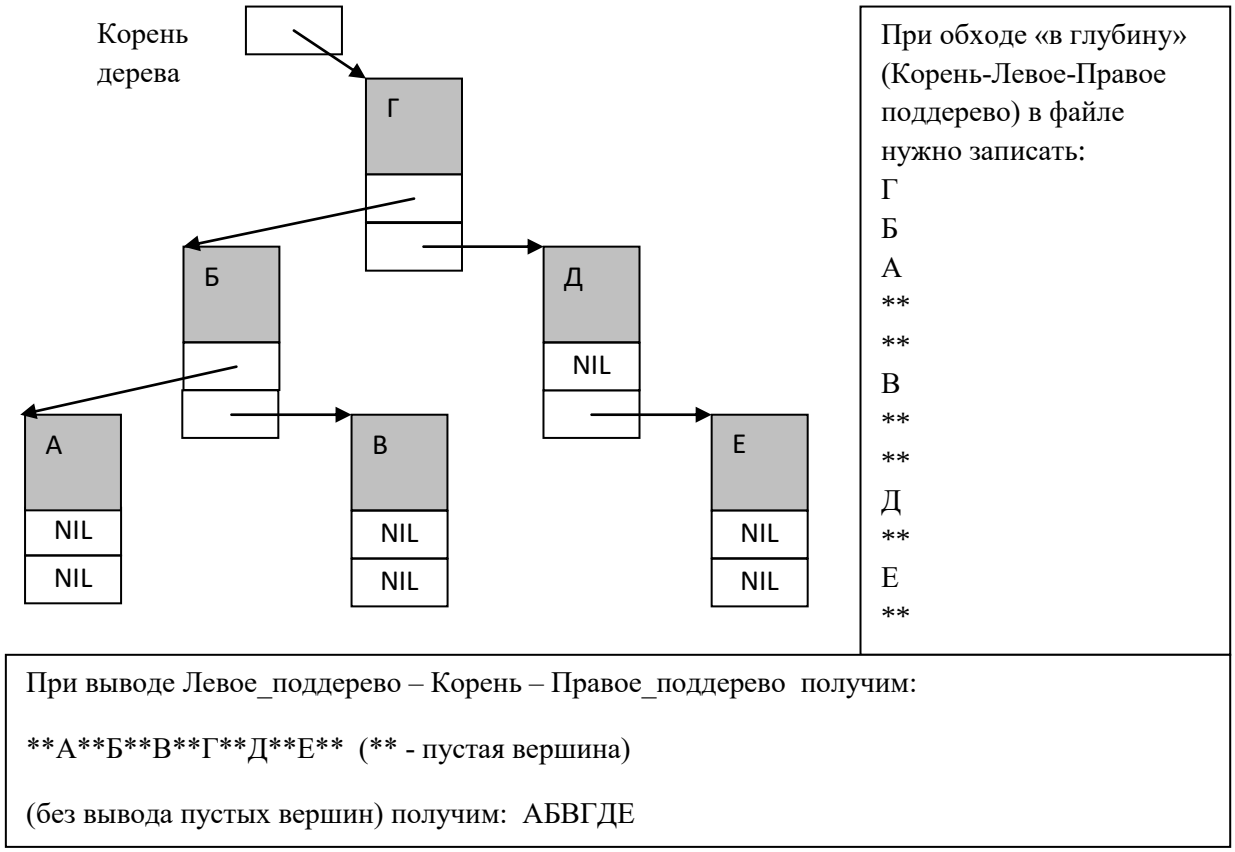
Мать: Мария, у которой отец не указан в дереве, а мать Ольга (бабушка Ивана).

Рис.16.4 Генеалогическое дерево

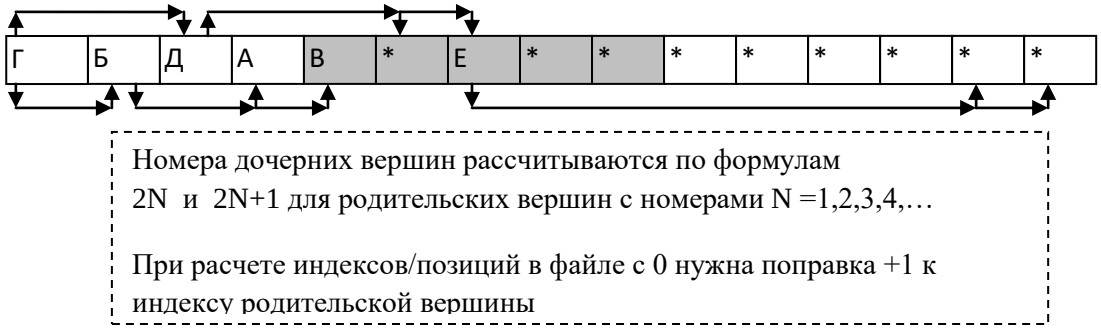
В задании «В» типового расчета надо создать **генеалогическое дерево**.

Для каждого человека в генеалогическом дереве указаны следующие сведения: Фамилия, Имя, Отчество, дата рождения, а также при необходимости дата смерти и место рождения. **Для каждой вершины левое поддерево отвечает за отца, правое – за мать.** Например, дерево Ивана (на рисунке 16.4 указаны для упрощения только имена):

Дерево предлагается **смоделировать двумя способами**:
1) на основе ссылочного типа «Нелинейный связный список»;



2) на основе динамического массива или типизированного/двоичного файла (на выбор студента).



При создании **упорядоченного дерева**, каждый элемент сразу надо добавлять в нужное место: Например, создадим упорядоченное дерево из чисел 6,3,4,2,6,9:

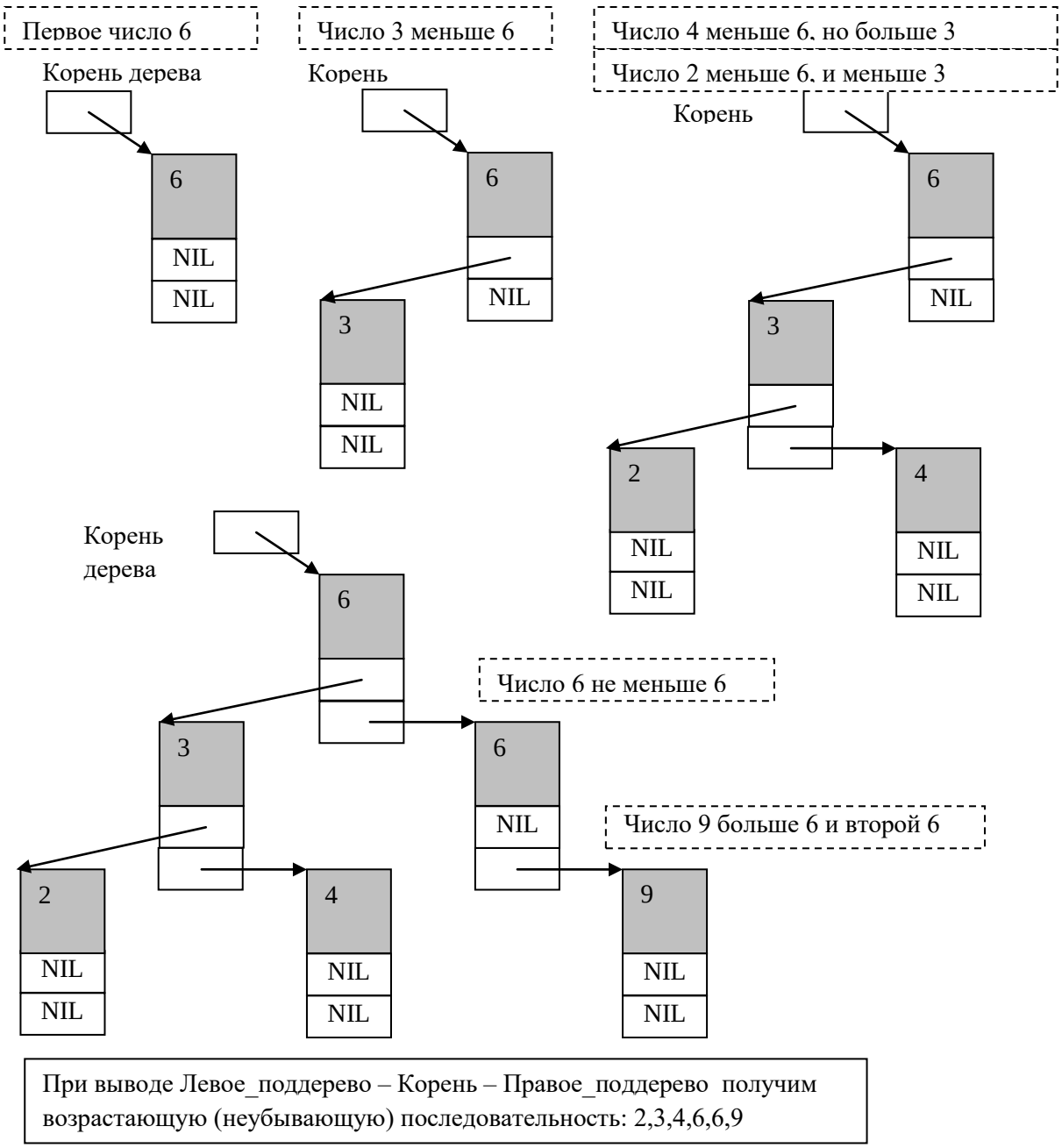


Рис.16.5 Порядок построения упорядоченного дерева

Пример консольного приложения
с вводом чисел для бинарного дерева из текстового файла
и выводом его (не более 5 ярусов) на экран

Тип бинарного дерева на основе нелинейного двусвязного списка в *Delphi*:

```
type
  Pder = ^Tder;
  Tder = record
    chislo: integer;   // значение в вершине дерева
    left, rigth: Pder;  // адреса корневых вершин левого и правого поддеревьев
  end;
```

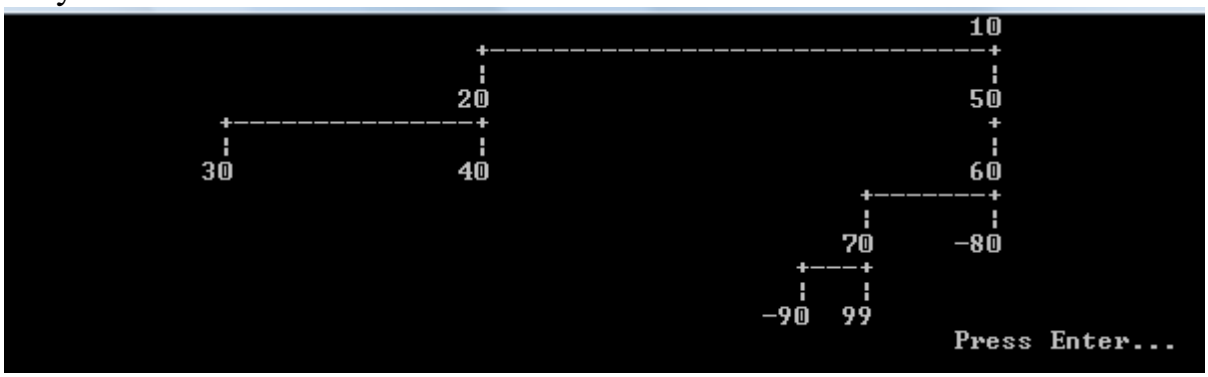
Обратите внимание, что в примере все процедуры для работы с деревом **рекурсивны**, например, освобождение памяти:

```
procedure free_tree(var root: Pder);
begin
  if root<> nil then
  begin
    free_tree(root^.left);  // рекурсивный вызов освобождения левого поддерева
    free_tree(root^.rigth);  // рекурсивный вызов освобождения правого поддерева
    dispose(root);  /// только после этого освобождаем и обнуляем (nil) текущую
    root:=nil;                          // вершину ведь в ней хранятся адресе поддеревьев
  end;
end;
```

Пример текстового файла Tree.txt
с описанием (Корень – Левое поддерево - Правое поддерево):

```
10
20
30
**
**
40
**
**
50
**
60
70
-90
**
**
100
**
**
-80
**
**
```

Результат



Добавлена корректировка:

Если число меньше -99, то оно исправляется на -99.

Если число больше 99, то оно исправляется на 99, например, 100 в рассматриваемом примере.

Аномалии не рассмотрены.

Программный код на Delphi

```
program Project2;

{$APPTYPE CONSOLE}

uses SysUtils;

type
  Pder = ^Tder;
  Tder = record // моделирование на основе двусвязного нелинейного списка
    chislo: integer;
    left, rigth: Pder;
  end;

var
  derevo: array [3..15] of string;  // изображение дерева, не более 5 ярусов

procedure create_tree(var root: Pder; var f: TextFile; i:byte; var kol:byte);
var r: integer; str: string[3];
begin
  readln(f, str);
  if TryStrToInt(str, r) then
  begin
    if r>99 then r:=99;
    if r<-99 then r:=-99;
    if i>kol then kol:=i;

   new(root);
   root^.chislo:=r; {-99..+99}
   create_tree(root^.left, f, i+1, kol);
   create_tree(root^.rigth, f, i+1, kol);
  end
  else
   root:=nil;
end;

procedure free_tree(var root: Pder);
begin
  if root<> nil then
  begin
    free_tree(root^.left);
    free_tree(root^.rigth);
    dispose(root);
```

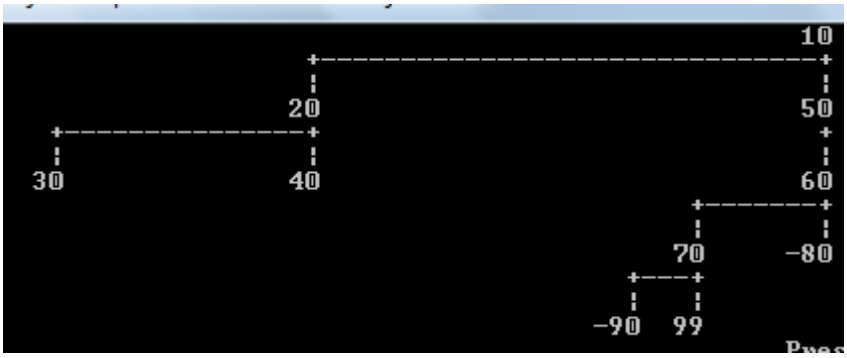
```
    root:=nil;
  end;
end;

procedure make_tree(root: Pder; i,k,kol: byte; r: boolean);
var j:byte;
begin
  if i<=kol then
  if root <> nil then
  begin
   if i>1 then
    begin
     if r then
       for j:=1 to 64 div k -1 do derevo[3*i-2]:=derevo[3*i-2]+'-'
     else
       for j:=1 to 64 div k -1 do derevo[3*i-2]:=derevo[3*i-2]+' ';
     derevo[3*i-2]:=derevo[3*i-2]+'+';
     for j:=1 to 64 div k -1 do derevo[3*i-1]:=derevo[3*i-1]+' ';
     derevo[3*i-1]:=derevo[3*i-1]+'|';
    end;
   for j:=1 to 64 div k - 3 do derevo[3*i]:=derevo[3*i]+' ';

   //знак
   if root^.chislo<0 then derevo[3*i]:=derevo[3*i]+'-'
   else derevo[3*i]:=derevo[3*i]+' ';
   //две цифры
   derevo[3*i]:=derevo[3*i]+Chr(48+abs(root^.chislo) div 10)+ Chr(48+abs(root^.chislo)
mod 10);

   //аналогично с поддреревьями
   make_tree(root^.left, i+1, 2*k, kol, false);
   make_tree(root^.rigth, i+1, 2*k, kol, root^.left<>Nil);
  end
  else
    begin  // пустое поддерево - пробелы
     if r then
       for j:=1 to 64 div k do derevo[3*i-2]:=derevo[3*i-2]+'-'
     else
       for j:=1 to 64 div k do derevo[3*i-2]:=derevo[3*i-2]+' ';
     for j:=1 to 64 div k do derevo[3*i-1]:=derevo[3*i-1]+' ';
     for j:=1 to 64 div k do derevo[3*i]:=derevo[3*i]+' ';

     make_tree(nil, i+1, 2*k, kol, false);
     make_tree(nil, i+1, 2*k, kol, false);
    end;
end;
```

```
var
  root: Pder;
  kol,j: byte;
  f: TextFile;

begin
  AssignFile(f, 'tree.txt');
  Reset(f);

  kol:=0;
  create_tree(root, f, 1, kol);

  CloseFile(f);

  if kol>5 then kol:=5;

  for j:=3 to kol*3 do derevo[j]:='';
  make_tree(root, 1, 1, kol, false);
  for j:=3 to kol*3 do writeln(derevo[j]);

  free_tree(root);
  write('Press Enter...':75); readln;
end.
```

В задаче из ТР также требуется вывести дерево 2-4мя разными способами: (Корень – Левое_поддерево – Правое_поддерево, Левое_поддерево – Корень – Правое_поддерево, Левое_поддерево – Правое_поддерево – Корень; графически (по желанию). ** – признак пустого поддерева);

Можно также **при графическом выводе** в текстовый файл изображать дерево не вертикально (как в приведенном выше коде), а горизонтально, с помощью отступов показывая расположение ветвей дерева. Количество отступов равно текущей глубине вызовов (см. дальше)

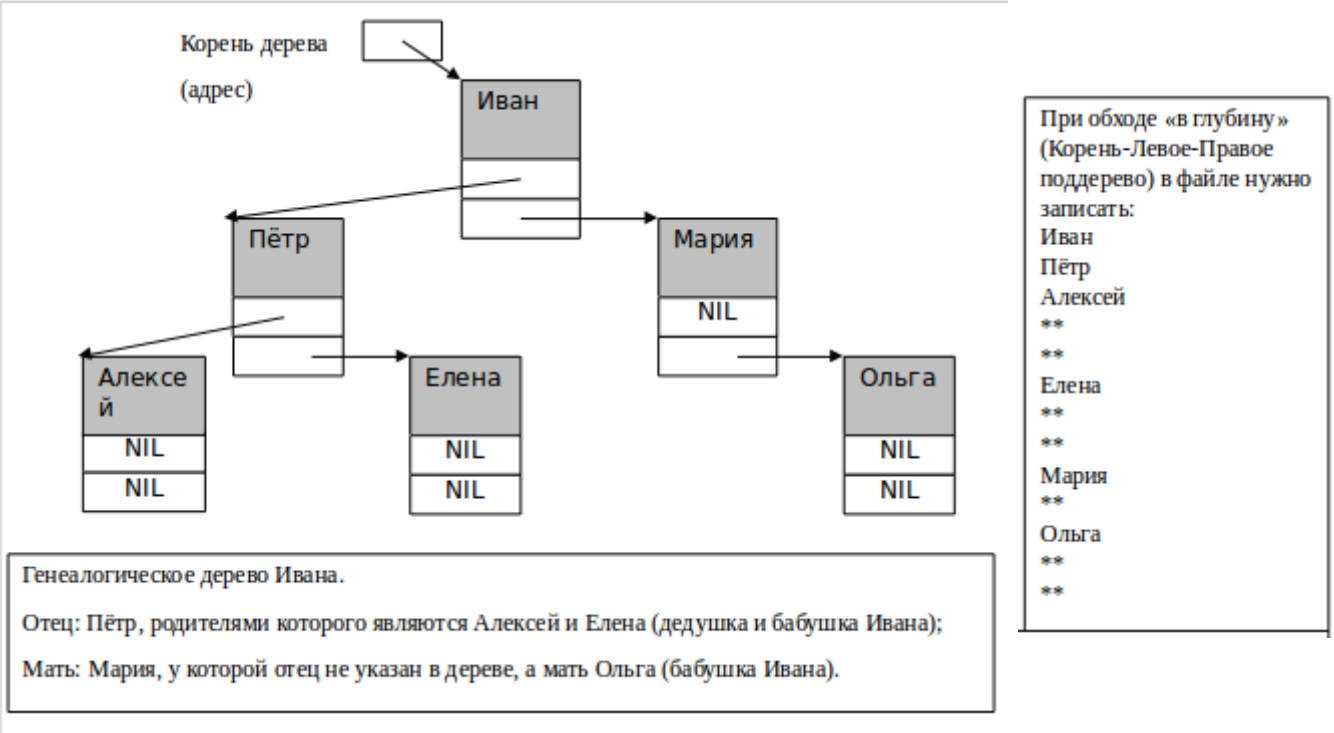
При вертикальном изображении не будет ограничения на количество ярусов, как в описанном примере, та как нет необходимости уместить дерево на экран и сделать запас слева для вывода <u>всех</u> возможных вершин.




```
10
--------- 20
          --------- 30
                    ---------**
                    ---------**
          --------- 40
                    ---------**
                    ---------**
--------- 50
          ---------**
          --------- 60
                    --------- 70
                              --------- -90
                                        ---------**
                                        ---------**
                              --------- 99
                                        ---------**
                                        ---------**
                    --------- -80
                              ---------**
                              ---------**
```

Такое дерево напоминает дерево папок.

Пример на Си++ рекурсивного графического вЫвода бинарного дерева, созданного на основе массива, для генеалогического дерева:



В нижеприведенном коротком примере для краткости дерево уже "введено" - задано константой (массив из 16 строк, нулевой элемент не задействован). А в ТР (пункт б.) надо вводить его (не массив поэлементно, а генеалогическое бинарное дерево по вышеуказанному файлу в массив или нелинейный список) :

```c
#include <stdio.h>
struct TTree{
    char**mas;
    int cur, len;
};

void go_left(TTree &T) { T.cur*=2; return;}
void go_right(TTree &T) { T.cur*=2; T.cur++; return;}
void go_up(TTree &T) { T.cur/=2; return;}
bool is_curtree_empty(TTree &T) { if  (T.mas[T.cur]==NULL) return true; return false;}
char* get_cur(TTree &T){ return T.mas[T.cur];}

void vert_tree(TTree &T, int lair = 1){

    if (is_curtree_empty(T)) printf("%*s%s\n",lair*5,"-----","**");
    else { char ch=0;
        printf("%*s%s\n",lair*5,"-----", get_cur(T));
        go_left(T); vert_tree(T, lair+1); go_up(T);
        go_right(T); vert_tree(T, lair+1); go_up(T);

    }
}
int main()
{
    printf("TP Tree\n");
    char* treeArray[]={NULL, "Иван",
                       "Петр", "Мария",
                       "Алексей", "Елена", NULL, "Ольга",
                       NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL};
    TTree T1 = {treeArray, 1, 16};
    vert_tree(T1);
    return 0;
}
```

TP-Tree2

```
TP Tree
-----Иван
     -----Петр
          -----Алексей
               -----**
               -----**
          -----Елена
               -----**
               -----**
     -----Мария
          -----**
          -----Ольга
               -----**
               -----**

Process returned 0 (0x0)   execution time : 0.004 s
Press ENTER to continue.
```

Программный код в виде текста для копирования:

```
#include <stdio.h>

struct TTree{
 char**mas;
 int cur, len;
};

void go_left(TTree &T) { T.cur*=2; return;}
void go_right(TTree &T) { T.cur*=2; T.cur++; return;}
void go_up(TTree &T) { T.cur/=2; return;}
bool is_curtree_empty(TTree &T) { if (T.mas[T.cur]==NULL) return true; return false;}
char* get_cur(TTree &T){ return T.mas[T.cur];}

void vert_tree(TTree &T, int lair = 1){
 if (is_curtree_empty(T)) printf("%*s%s\n",lair*5,"-----","**");
 else { char ch=0;
      printf("%*s%s\n",lair*5,"-----", get_cur(T));
      go_left(T); vert_tree(T, lair+1); go_up(T);
      go_right(T); vert_tree(T, lair+1); go_up(T);
 }
}


int main(){
 printf("TP Tree\n");
 char* treeArray[]={NULL, "Иван",
                         "Петр", "Мария",
                         "Алексей", "Елена", NULL, "Ольга",
                         NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL};
 TTree T1 = {treeArray, 1, 16};
 vert_tree(T1);
 return 0;
}
```