МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Тульский государственный университет»

Институт прикладной математики и компьютерных наук

Кафедра вычислительной техники

Отчет по лабораторной работе № 7

«Деревья»

по дисциплине «Функциональное и логическое программирование»

Вариант 28(1)

Выполнил студент группы 220681:

Шайхаттаров Д.В.

Проверил:

Асс. Савенков П. А.

Тула 2020

**1. Цель работы**

Получить навыки создания базы знаний на языке Пролог с помощью дерева.

**2. Задание на лабораторную работу**

Создайте предикат, удаляющий из двоичного справочника максимальный элемент левого поддерева.

**3. Постановка задачи**

Выделяем один левую часть нашего дерева в отдельное, далее рекурсивными методом ищем элемент с максимальным значением и затем удаляем его, объединяем дерево обратно.

**4. Тестирование**

При запуске программы выводятся следующие результаты (Рисунок 1).

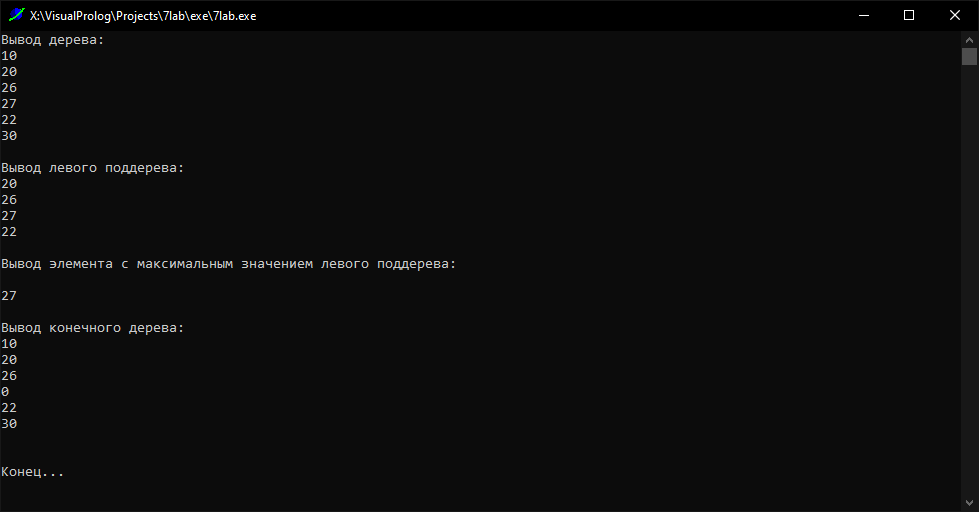


Рисунок 1. Результаты тестирования.

**5. Заключение**

В ходе данной лабораторной работы были получены навыки создания базы знаний на языке Пролог с помощью дерева.

**Приложение**

**Исходный код программы задания**

goal  
    mainExe::run(main::run).  
  
implement main  
    open core, console  
  
domains  
    treetype =  
        tree(integer, treetype, treetype);  
        empty.  
  
class predicates  
    print\_tree : (treetype).  
    left\_tree : (treetype, treetype [out]).  
    max\_root : (integer, treetype, integer [out]).  
    check\_root : (integer, treetype, integer [out]).  
    check\_value : (integer, integer, integer [out]).  
    check\_eq : (integer, integer, integer [out]).  
    del\_element : (integer, treetype, treetype [out]).  
    new\_left\_part : (treetype, treetype, treetype [out]) determ.  
    start : () determ.  
  
clauses  
    start() :-  
        Tree = tree(10, tree(20, tree(26, tree(27, empty, empty), empty), tree(22, empty, empty)), tree(30, empty, empty)),  
        write("Вывод дерева:"),  
        nl,  
        print\_tree(Tree),  
        nl,  
        left\_tree(Tree, L\_Tree),  
        write("Вывод левого поддерева:"),  
        nl,  
        print\_tree(L\_Tree),  
        nl,  
        write("Вывод элемента с максимальным значением левого поддерева:\n"),  
        nl,  
        max\_root(0, L\_Tree, L\_Out),  
        write(L\_Out),  
        nl,  
        write("\nВывод конечного дерева:"),  
        nl,  
        %print\_res(L\_Out, Tree),  
        del\_element(L\_Out, L\_Tree, L\_Out\_Tree),  
        new\_left\_part(Tree, L\_Out\_Tree, End\_Tree),  
        print\_tree(End\_Tree),  
        nl,  
        write("\nКонец...").  
  
    new\_left\_part(empty, empty, empty) :-  
        !.  
    new\_left\_part(tree(Root, \_, Right), New\_Left\_Tree, Out\_Tree) :-  
        Out\_Tree = tree(Root, New\_Left\_Tree, Right).  
    del\_element(\_, empty, empty) :-  
        !.  
    del\_element(Value, tree(Root, Left, Right), Out\_Tree) :-  
        check\_eq(Root, Value, Out\_Value),  
        del\_element(Value, Left, L\_Value),  
        del\_element(Value, Right, R\_Value),  
        Out\_Tree = tree(Out\_Value, L\_Value, R\_Value).  
    max\_root(\_, empty, 0) :-  
        !.  
    max\_root(Value, tree(Root, Left, Right), Out) :-  
        check\_root(Value, tree(Root, Left, Right), Out\_Root),  
        max\_root(Out\_Root, Left, Out\_Left),  
        max\_root(Out\_Root, Right, Out\_Right),  
        check\_value(Out\_Root, Out\_Left, L\_Res),  
        check\_value(L\_Res, Out\_Right, R\_Res),  
        Out = R\_Res.  
  
    check\_eq(Value\_1, Value\_2, Out) :-  
        if Value\_1 = Value\_2 then  
            Out = 0  
        else  
            Out = Value\_1  
        end if.  
    check\_root(\_, empty, 0) :-  
        !.  
    check\_root(Value, tree(Root, \_, \_), Out) :-  
        if Value < Root then  
            Out = Root  
        else  
            Out = Value  
        end if.  
  
    check\_value(Value\_1, Value\_2, Out) :-  
        if Value\_1 < Value\_2 then  
            Out = Value\_2  
        else  
            Out = Value\_1  
        end if.  
    print\_tree(empty) :-  
        !.  
    print\_tree(tree(Root, Left, Right)) :-  
        write(Root),  
        nl,  
        print\_tree(Left),  
        print\_tree(Right).  
  
    left\_tree(empty, empty) :-  
        !.  
    left\_tree(tree(\_, Left, \_), OutTree) :-  
        OutTree = Left.  
  
clauses  
    run() :-  
        init(),  
        if start() then  
        end if,  
        \_ = readLine().  
  
end implement main