Программирование

Лекция дополнительная
Работа с деревьями

Петров Александр Владимирович

Фёдоров Станислав Алексеевич

Октябрь 2013

```
type Node
        integer Num = 0
        type (Node), pointer :: left  => Null()
        type (Node), pointer :: right => Null()
end type Node
```

```
recursive pure subroutine Put_tree( tree, elem )
    type (Node), intent (inout) :: tree
    integer, intent (out) :: elem

    if (elem < Tree%Num) then
        if (.not. Associated(Tree%Left)) then
            allocate (New)
            New%Num = elem
            Tree%Left => New
        else
            call Put_tree( Tree%Left, elem )
        end if
    else if (elem > Tree%Num) then
        if (.not. Associated(Tree%Right)) then
            allocate (New)
            New%Num = elem
            Tree%Right => New
        else
            call Put_tree( Tree%Right, elem )
        end if
    end if
end subroutine
```

```
recursive subroutine Output_tree( tree )
    type (Node), intent (in) :: tree

    ! Если есть есть куда идти налево, то двигаться туда.
    if (Associated(tree%left)) &
        call Output_tree( tree%left )

    ! Вывод текущего элемента.
    write (OUTPUT_UNIT, "(i0, 1x)", advance='no') tree%Num

    ! Двигаемся направо, если можно.
    if (Associated(tree%right)) &
        call Output_tree( tree%right )
end subroutine
```

```
recursive subroutine Output_tree_straight( tree )
    type (Node), intent (in) :: tree

     ! Вывод текущего элемента.
    write (OUTPUT_UNIT, "(i0, 1x)", advance='no') tree%Num

    ! Если есть есть куда идти налево, то двигаться туда.
    if (Associated(tree%left)) &
        call Output_tree( tree%left )

    ! Двигаемся направо, если можно.
    if (Associated(tree%right)) &
        call Output_tree( tree%right )
end subroutine
```

```
recursive subroutine Output_tree_reverse( tree )
    type (Node), pointer, intent (inout) :: tree

    ! Если есть есть куда идти налево, то двигаться туда.
    if (Associated(tree%left)) &
        call Output_tree( tree%left )

    ! Двигаемся направо, если можно.
    if (Associated(tree%right)) &
        call Output_tree( tree%right )

    ! Вывод текущего элемента.
    write (OUTPUT_UNIT, "(i0, 1x)", advance='no') tree%Num
end subroutine
```

```
recursive pure function Search_in_tree( tree, elem ) result (res)
    logical res
    type (Node), pointer, intent (in) :: tree
    integer, intent (in) :: elem

    if (elem == tree%Num) then
        res = .true.
    else if (elem < tree%Num)
        ! Если есть есть куда идти налево, то двигаться туда.
        if (Associated(tree%left)) then
            call Search_in_tree( tree%left )
        else
            res = .false.
        end if
    else
        if (Associated(tree%right)) then
            call Search_in_tree( tree%right )
        else
            res = .false.
        end if
    end if

end function
```