Программирование

Лекция 13
Динамические структуры данных

Петров Александр Владимирович

Фёдоров Станислав Алексеевич

(по материалам Веренинова Игоря Андреевича с изменениями и дополнениями на Fortran 08 и UML)

Октябрь 2013

Ссылки и адресаты (1)

Ссылка – переменная, связанная
с другой переменной, называемой адресатом.

При обращении к ссылке будет происходить
обращение к адресату и наоборот.

```
integer, pointer :: p  ! ссылка
integer, target :: a   ! адресат
```

Ссылки позволяют создавать динамические структуры данных -
списки, стеки, деревья, очереди.

Ссылки и адресаты (2)

Операция => прикрепление ссылки к адресату.

```
program  prog
   integer, pointer :: p
   integer, target :: a
   a = 100
   p => a       ! прикрепили ссылку к адресату
   write(*,*) p

   p = 100    ! a = 100
   a = 500    ! p = 500
end
```

Все изменения, происходящие с адресатом, дублируются в ссылке.

Ссылки на массивы

```
real, pointer :: pa(:)
real, target :: a(5) = [1, 2, 3, 4, 5]
pa => a
print *, pa
```

Результат

1.000000 2.000000 3.000000 4.000000 5.000000

Массивы ссылок

Явно не поддерживаются.

Выход?

```
type Cell
    real, pointer :: column(:)
end type Cell

type(Cell) :: matrix(:)
```

Ссылки и адресаты (3)

Функция **associated(pt, addr)** возвращает **.TRUE.**
если ссылка **pt** прикреплена к адресату **addr**.

```
program prog
   integer, pointer :: p1, p2, p3
   integer, target :: a,b
   a = 100;  b = 2; p1 => a;  p2 => a
   write(*, *) associated(p1, p2)  ! TRUE
   write(*, *) associated(p1)
   write(*, *) associated(p2, a)
   p1 => b
   write(*, *) associated(p3)      ! FALSE
   write(*, *) associated(p1, p2)
   write(*, *) associated(p1, a)
end
```

Ссылки и адресаты (4)

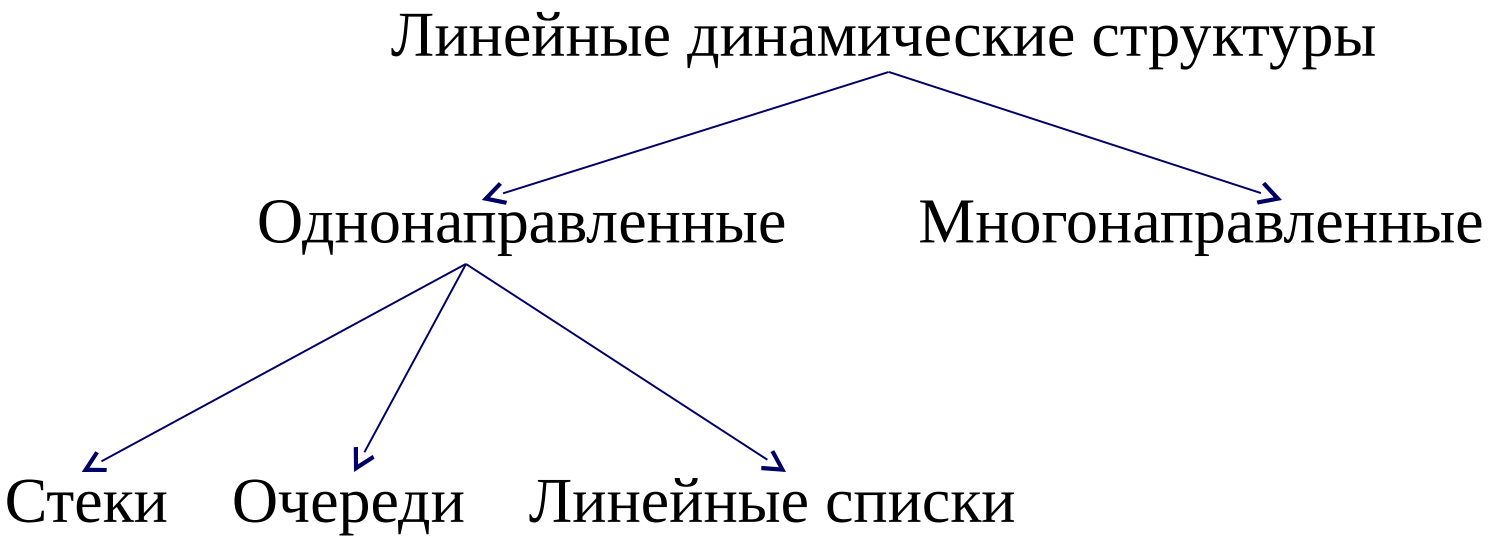
Оператор **nullify** - открепление ссылки от адресата.

```
program prog
   integer, pointer :: p1, p2
   integer, target :: a

   a = 1000;  p1 => a;  p2 => a
   ! если к адресату прикреплены две ссылки,
   ! то отсоединим последнюю

   if (associated(p1, p2)) nullify(p2)

   write(*,*) associated(p1), associated(p2) ! T, F
end
```

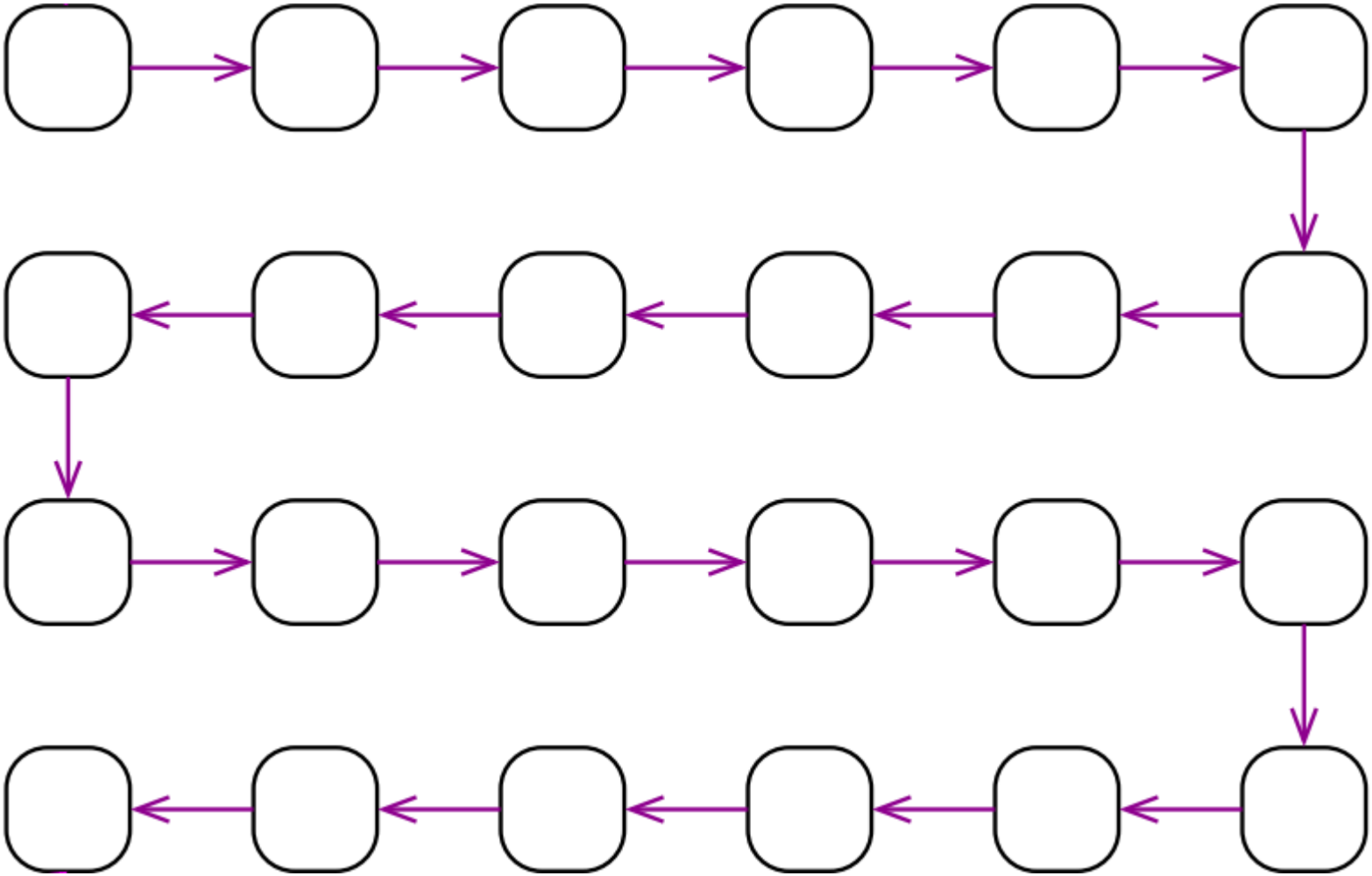
Пример

```
function select (switch, left, right)
   real, pointer :: select, left, right
   logical switch
   if (switch) then
      select => left
   else
      select => right
   end if
end function select

new_arrow => select(a > b,old_arrow,null())
```

Сложные динамические структуры данных



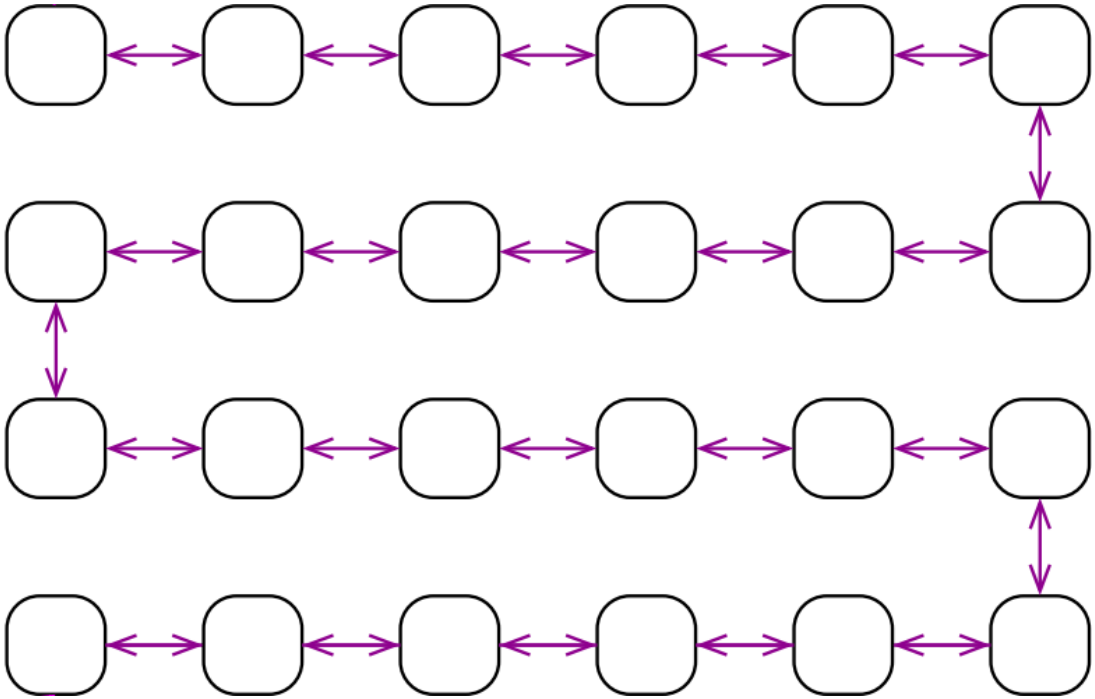
Однонаправленные списки

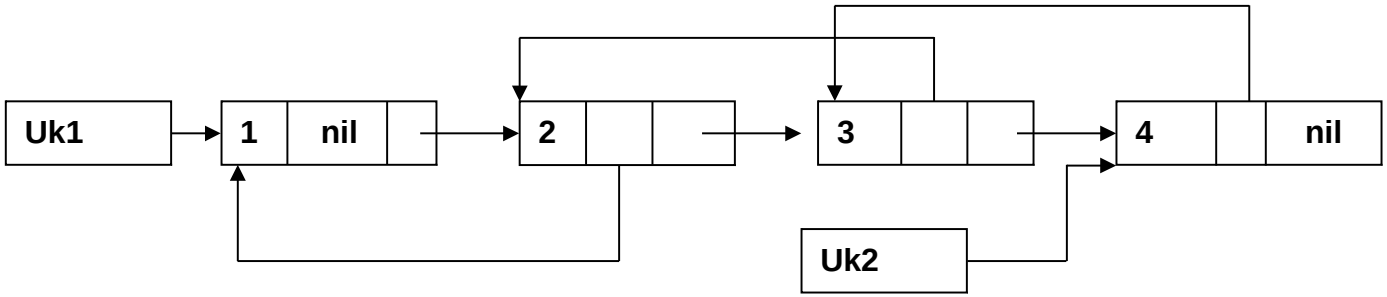


Пример

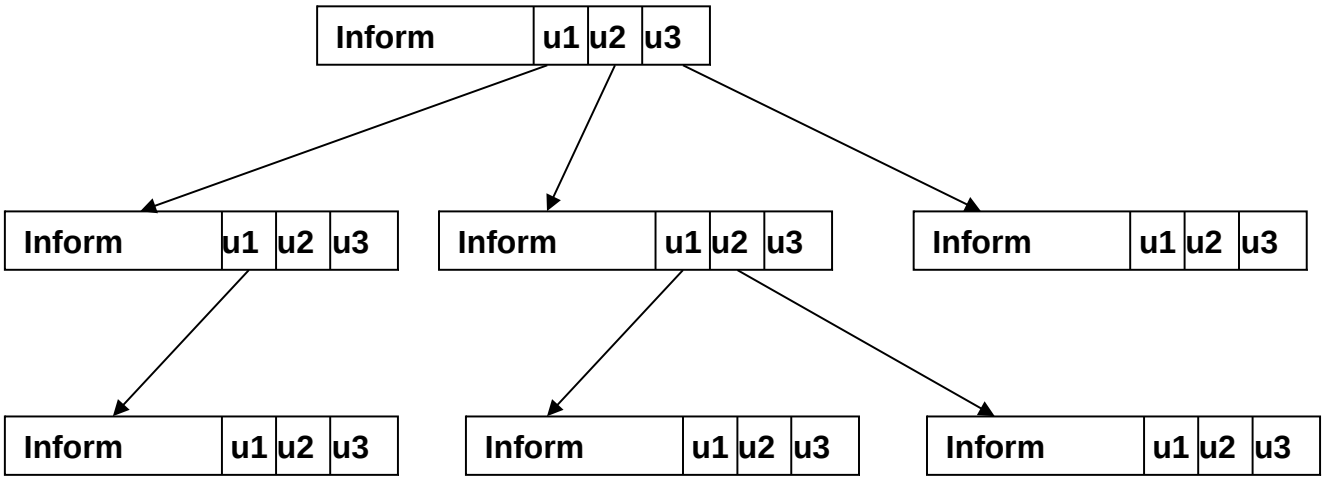
```
type Cell
    character(len = 20) :: node_name
    real :: node_weight
    type(Cell), pointer :: next, last, first_child, last_child
end type Cell
```

Двунаправленные списки



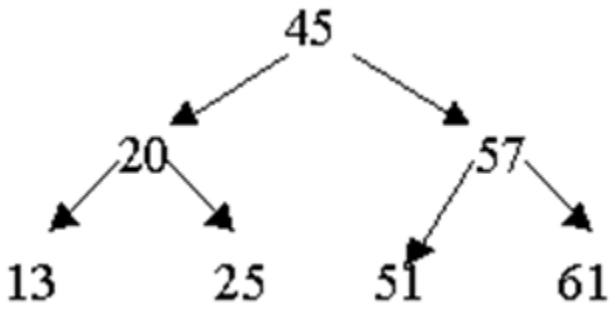


Троичное дерево: корень, листья, поддеревья



Двоичные деревья поиска

Надо записать в дерево поиска последовательность целых чисел:
45,57,20,13,61,25,51



Пример (1)

```
type :: Leaf
    integer num = 0
end type Leaf

type :: Branch
    type(leaf), allocatable :: leaves(:)
end type Branch

type :: Trunk
    type(branch), allocatable :: branches(:)
end type Trunk
```

Пример (2)

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

Интерактивные примеры

- Построение дерева поиска
- Вывод дерева поиска
- Прямой обход дерева поиска
- Обратный обход дерева поиска
- Поиск заданного элемента в дереве

Построение дерева поиска

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

        type (Node), pointer, intent (inout) :: New

        if (.not. Associated(tree)) then
                Allocate(New)
                New%Num = elem
                Tree => New
        else 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
```

Вывод дерева поиска

```
recursive subroutine Output_tree(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)

   ! 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
```

Прямой обход дерева поиска

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

Обратный обход дерева поиска

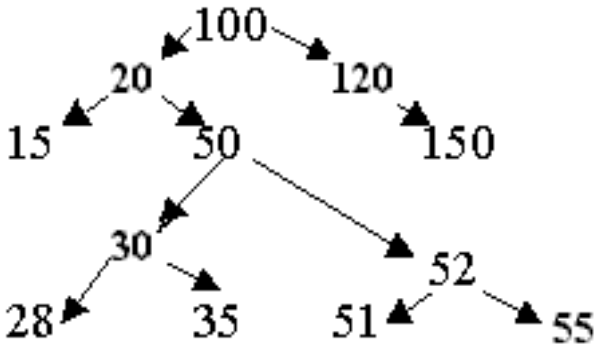
```
recursive subroutine Output_tree_reverse(tree)
   type(Node), pointer, intent (inout) :: tree

   ! 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
```

Поиск заданного элемента в дереве

```
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
```

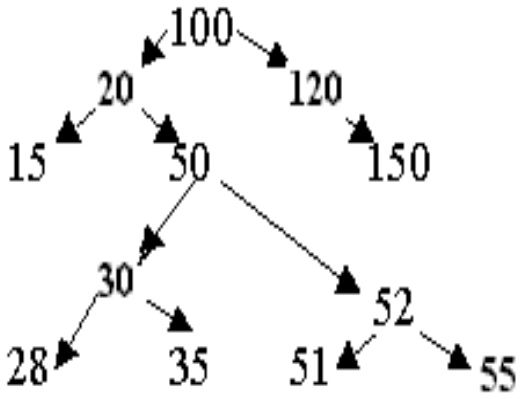
Удаление элемента из дерева

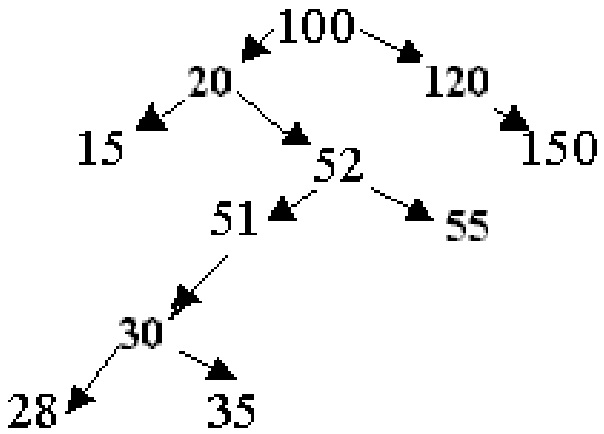
1. Заданного элемента в дереве нет
2. Заданный элемент является листом (не имеет подчиненных)
3. К элементу подключен только 1 подчиненный
4. К элементу подключены два подчиненных (два поддерева)
5. Исключаемый элемент – это корень



Удаление элемента из дерева (первая стратегия)

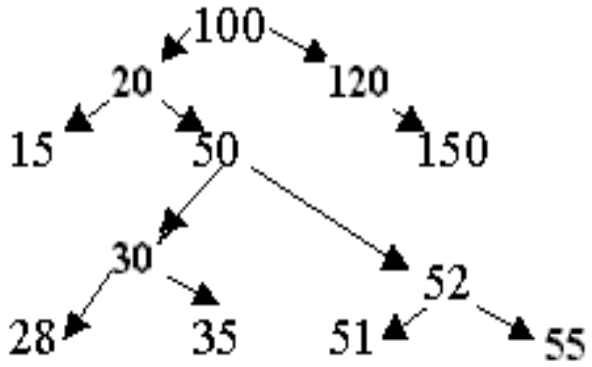
Пусть требуется удалить число 50 из дерева

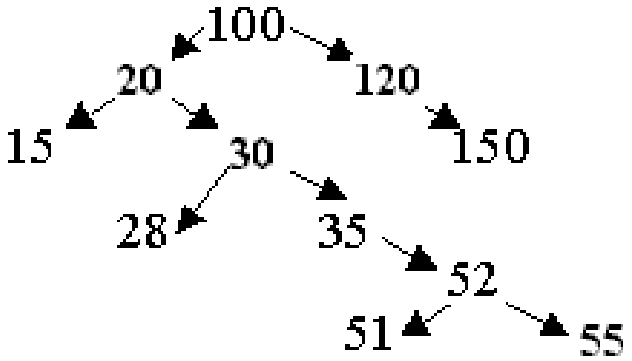




Удаление элемента из дерева (вторая стратегия)

Пусть требуется удалить число 50 из дерева





Удаление элемента из дерева

```
recursive subroutine Delete_in_tree(tree, elem)
   type (Node), pointer, intent (in) :: tree
   integer, intent (in) :: elem

   if (elem == tree%Num) then

   else if (elem < tree%Num)
      ! Если есть есть куда идти налево, то двигаться туда.
      if (Associated(tree%left)) &
         call Search_in_tree(tree%left)
   else
      if (Associated(tree%right)) &
         call Search_in_tree(tree%right)
   end if
end function
```

*Целочисленные указатели (1)

Целочисленный указатель – переменная целого типа, содержащая адрес некоторой переменной, называемой адресной переменной.

```
real a           ! базируемая переменная
pointer (p,a)    ! p – целочисленный указатель
                 ! на переменную типа real

character ch      ! ch – базируемая переменная
pointer (pc,ch)   ! pc – целочисленный указатель
                  ! на тип character
```

Целочисленный указатель и базируемая переменная используются совместно.

Целочисленный указатель часто используется для обращения к функциям языка С.

*Целочисленные указатели (2)

Функция **LOC** вычисляет адрес переменной.

```
program arrow
integer a    ! базируемая переменная
pointer(p,a) ! указатель на целый тип

integer :: b = 100

  p = loc(b)       ! вычислили адрес переменной b
  a = 500          ! базируемой переменной поместим в b
                   ! значение 500

  write(*,*)"address = ",p, &  ! 5038080
            "  value = ",a, &  ! 500
            "  b     = ",b     ! 500
end
```