

## Списки в языке Haskell.

```
[]           -- пустой список
[1, 2, 3]    -- список из заданных элементов
1:[2, 3]     -- присоединение головного элемента к списку
1:(2:(3:[])) -- создание списка с помощью конструктора ':'
[1..n]       -- создание списка с помощью арифметической прогрессии
[2, 4..20]   -- арифметическая прогрессия с заданной разностью
```

### Типы списков

```
[Integer]   -- список из целых чисел: [1..10]
[Char]       -- список из символов (строка: "List" == ['L','i','s','t'])
[(Char, Int)] -- список из кортежей: [('L', 1), ('i', 2), ('s', 3)]
[[Int]]      -- список из списков: [[1, 2], [3, 5..10], []]
```

### Функция суммирования элементов списка

```
sumList      :: [Integer] -> Integer
sumList []   = 0
sumList (x:s) = x + sumList s

sumList [1, 3, 6]
1 + sumList [3, 6]
1 + 3 + sumList [6]
1 + 3 + 6 + sumList []
10
```

## Еще один способ вычисления факториала.

```
factorial      :: Integer -> Integer
prodList'     :: [Integer] -> Integer -> Integer
factorial n   = prodList' [1..n] 1
prodList' [] p = p
prodList' (x:ls) p = prodList' ls (p*x)    -- концевая рекурсия
```

## Несколько стандартных операций над списком и их определения.

```
head          :: [a] -> a
head (x:ls)   = x
head []       = error "head: empty list"

tail          :: [a] -> [a]
tail (x:ls)   = ls
tail []       = error "tail: empty list"

length        :: [a] -> Int
length (x:ls) = 1 + length ls
length []     = 0

null          :: [a] -> Bool
null (x:ls)   = False
null []       = True
```

## Более сложные функции обработки списков.

```
last      :: [a] -> a
last []   = error "last: empty list"
last [x]  = x
last (x:ls) = last ls

init      :: [a] -> [a]
init []   = error "init: empty list"
init [x]  = []
init (x:ls) = x : init ls

 (!! )    :: [a] -> Int -> a
[] !! _  = error "(!!): empty list"
(x:ls) !! 0 = x
(x:ls) !! n = ls !! (n-1)

(++ )    :: [a] -> [a] -> [a]
[] ++ ls = ls
(x:l1) ++ l2 = x : (l1 ++ l2)

reverse  :: [a] -> [a]
reverse' :: [a] -> [a] -> [a]
reverse ls      = reverse' ls []
reverse' [] l   = l
reverse' (x:ls) l = reverse' ls (x:l)
```

## 1.3. Определение новых типов данных.

### *Определение синонимов для типов*

```
type String    = [Char]
type Coord     = (Double, Double)
type Pair a    = (a, a)
type Complex   = Pair Double
```

### *Использование синонимов*

```
find      :: String -> Char -> Int
find [] _ = -1
find (x:s) y | x == y      = 0
              | otherwise  = 1 + find s y

distance  :: Coord -> Coord -> Double
distance (x1, y1) (x2, y2) = sqrt ((x2-x1) * (x2-x1) + (y2-y1) * (y2-y1))

complexAdd :: Complex -> Complex -> Complex
complexAdd (r1, i1) (r2, i2) = (r1+r2, i1+i2)

swap      :: Pair a -> Pair a
swap (x, y) = (y, x)
```

## 1.3. Определение новых типов данных.

### *Определение конструкторов*

```
data WeekDay = Sun | Mon | Tue | Wed | Thu | Fri | Sat
data Bool    = False | True
```

### *Использование конструкторов*

```
weekend      :: WeekDay -> Bool
weekend Sun  = True
weekend Sat  = True
weekend _    = False
```

### *Конструкторы с параметром*

```
data Coord   = Coord Double Double
data Pair a  = Pair a a
```

### *Использование конструкторов с параметрами*

```
distance     :: Coord -> Coord -> Double
distance (Coord x1 y1) (Coord x2 y2) =
    sqrt ((x2-x1) * (x2-x1) + (y2-y1) * (y2-y1))

swap         :: Pair a -> Pair a
swap (Pair x y) = Pair y x
```

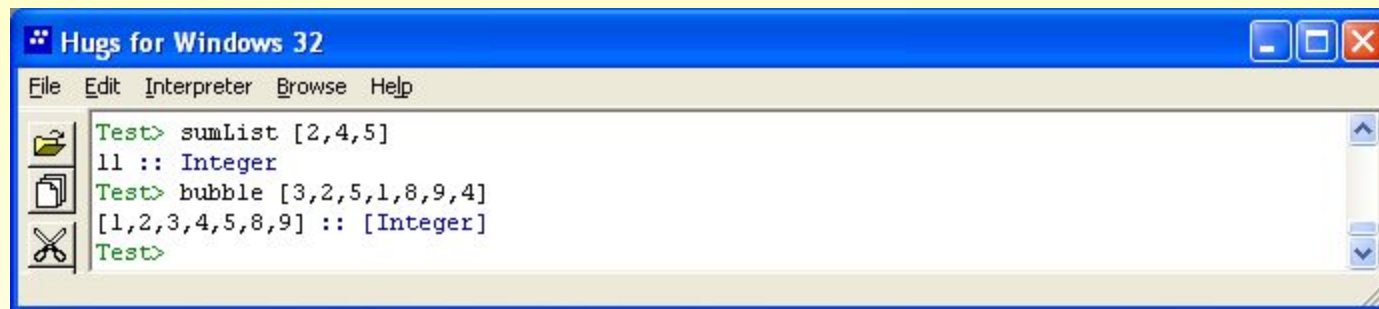
## Сложные типы данных.

```
data IntList = Nil | Cons Integer IntList
sumList      :: (Num a) => IntList -> a
sumList Nil  = 0
sumList (Cons e l) = e + sumList l
```

## Сортировка списка.

```
insert      :: (Ord a) => a -> [a] -> [a]
insert elem [] = [elem]
insert elem list@(x:s) | elem < x = elem:list
                       | otherwise = x:(insert elem s)

bubble      :: (Ord a) => [a] -> [a]
bubble []   = []
bubble (x:s) = insert x (bubble s)
```



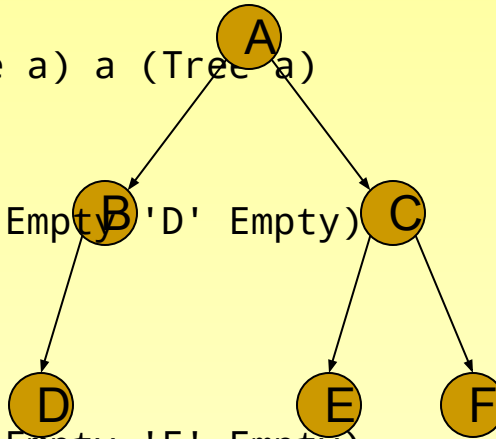
The screenshot shows a window titled "Hugs for Windows 32" with a menu bar (File, Edit, Interpreter, Browse, Help) and a toolbar. The main area contains the following text:

```
Test> sumList [2,4,5]
11 :: Integer
Test> bubble [3,2,5,1,8,9,4]
[1,2,3,4,5,8,9] :: [Integer]
Test>
```

## Определение и обработка двоичного дерева.

```
data Tree a = Empty |
            Node (Tree a) a (Tree a)

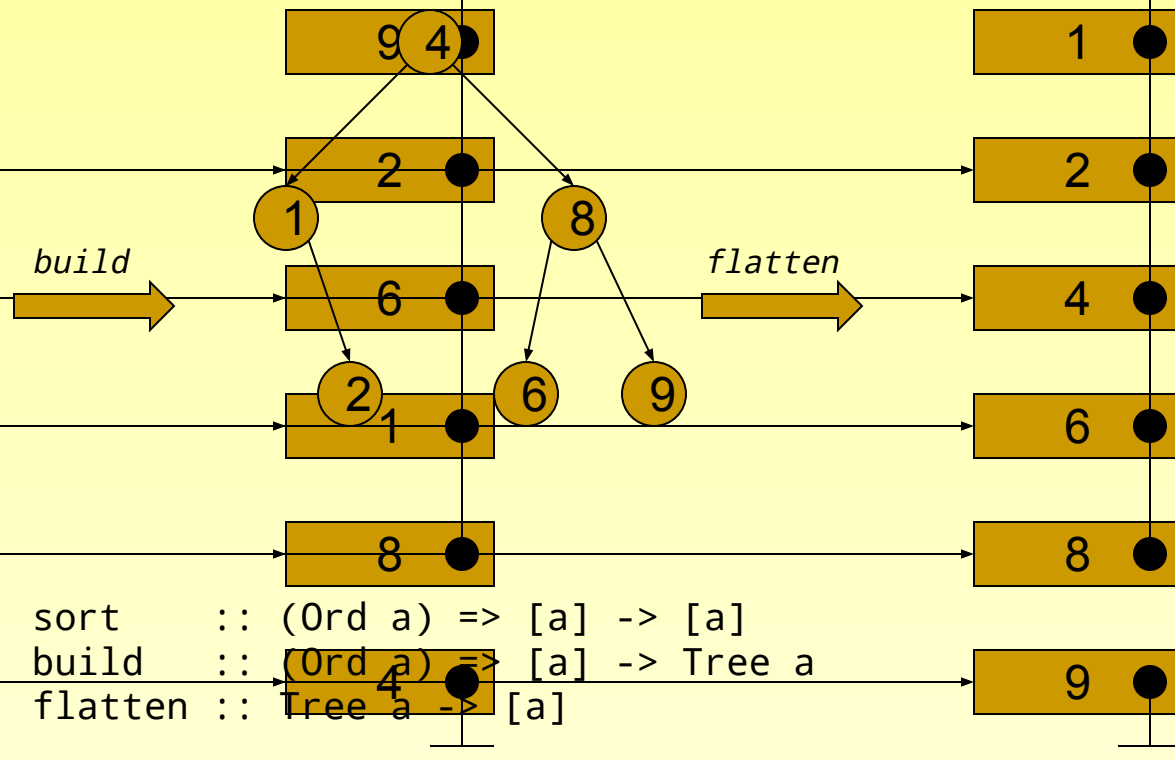
myTree :: Tree Char
myTree = Node (Node
              (Node Empty 'D' Empty) 'B'
              Empty)
            'A'
            (Node
              (Node Empty 'E' Empty) 'C'
              (Node Empty 'F' Empty))
```



```
height :: Tree a -> Int
height Empty = 0
height (Node t1 _ t2) = 1 + max (height t1) (height t2)
```

```
Hugs for Windows 32
File Edit Interpreter Browse Help
Test>
Test>
Test> height myTree
3 :: Int
Test>
```

# Сортировка с помощью двоичного дерева.





## Программа сортировки с помощью двоичного дерева.

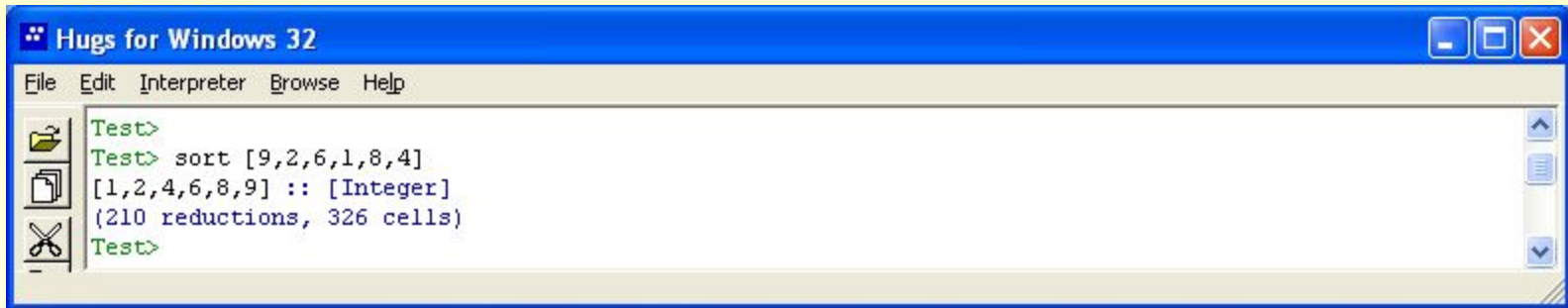
```
data Tree a = Empty |
            Node (Tree a) a (Tree a)

sort      :: (Ord a) => [a] -> [a]
build     :: (Ord a) => [a] -> Tree a
insert    :: (Ord a) => a -> Tree a -> Tree a
flatten   :: Tree a -> [a]

sort ls    = flatten (build ls)
build []   = Empty
build (e:ls) = insert e (build ls)

insert e Empty = Node Empty e Empty
insert e (Node t1 n t2) | e < n = Node (insert e t1) n t2
                        | e >= n = Node t1 n (insert e t2)

flatten Empty = []
flatten (Node t1 n t2) = (flatten t1) ++ (n : (flatten t2))
```



The screenshot shows a window titled "Hugs for Windows 32" with a menu bar (File, Edit, Interpreter, Browse, Help) and a toolbar. The main area displays the following text:

```
Test>
Test> sort [9,2,6,1,8,4]
[1,2,4,6,8,9] :: [Integer]
(210 reductions, 326 cells)
Test>
```