

## 2.2. Карринг

### Частичная параметризация функций

```
plus      :: Integer -> Integer -> Integer
plus x y  =  x + y
```

```
plus2     :: Integer -> Integer
plus2 y   =  2 + y
```

```
plus2     :: Integer -> Integer
plus2 y   =  plus 2 y
```

```
map plus2 [5, 3, 8, 10]  =>  [7, 5, 10, 12]
```

```
plus2     :: Integer -> Integer
plus2     =  plus 2
```

```
map (plus 2) [5, 3, 8, 10]  =>  [7, 5, 10, 12]
```

```
plus      :: Integer -> (Integer -> Integer)
plus x    =  \y -> x + y
```

```
plus      :: Integer -> Integer -> Integer
plus     =  \x y -> x + y
```

## Различные формы записи уравнений

```
comp      :: (b -> c) -> (a -> b) -> (a ->> c)
comp f g x = f (g x) f (g x) x -> f (g x)
```

Все функции в *Haskell* – это функции с одним аргументом и одним результатом!

Haskell В. Curry – карринг.

«Карринговые» функции – это частично параметризуемые функции.

```
plus1      :: Integer -> Integer -> Integer  -- в карринговой форме
plus1 x y  =  x + y
```

```
plus2      :: (Integer, Integer) -> Integer  -- не в карринговой форме
plus2 (x, y) =  x + y
```

```
curry plus2      =>      plus1
uncurry plus1    =>      plus2
```

```
curry  :: ((a, b) -> c) -> (a -> b -> c)
uncurry :: (a -> b -> c) -> ((a, b) -> c)
curry f x y      =  f (x, y)
uncurry f (x, y) =  f x y
```

## Сечения

```
(+) :: (Num a) => a -> a -> a
```

```
(+) 5 8 -> 13
```

```
(+) 5 -> \n->5+n
```

```
(5 +)
```

```
raiseList :: (Num a) => [a] -> [a]
```

```
raiseList lst = map (1+) lst
```

```
(+) ? 8 -> \n->n+8
```

```
(+ 8)
```

```
searchList :: (Eq a) => a -> [a] -> Bool
```

```
searchList e = (foldr (||) False) . (map (== e))
```

```
searchList 5 [1,3,7,5,2]
```

```
((foldr (||) False) . (map (== 5))) [1,3,7,5,2]
```

```
foldr (||) False (map (== 5) [1,3,7,5,2])
```

```
foldr (||) False [1 == 5, 3 == 5, 7 == 5, 5 == 5, 2 == 5]
```

```
foldr (||) False [False, False, False, True, False]
```

```
True
```

## Еще раз о сортировке списка с помощью дерева

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

```
build list    = foldr insert Empty list
flatten tree  = foldTree (:) [] tree
```

## Фильтрация списка

```
filter      :: (a -> Bool) -> [a] -> [a]
filter f [] = foldr condCons [] []
filter f (x:ls) = if (filter f x) then x : filter f ls
                  | otherwise = filter f ls

quicksort   :: (Ord a) => [a] -> [a]
quicksort [] = []
quicksort (x:ls) = (quicksort (filter (< x) ls)) ++ [x] ++
                  (quicksort (filter (>= x) ls))

quicksort [] = []
quicksort (x:ls) = (quicksort [y | y<-ls, y < x]) ++ [x] ++
                  (quicksort [y | y<-ls, y >= x])

descartes ls1 ls2 = [(x, y) | x <- ls1, y <- ls2]
oddSqrns ls       = [x*x | x <- ls, x `mod` 2 == 1]
```

## Характеристическая функция множества

```
type IntSet = (Integer -> Bool)
empty      :: IntSet
empty e    = False

from2to100 :: IntSet
from2to100 e = (e >= 2) && (e <= 100)

odds      :: IntSet
odds e    = (e `mod` 2 == 1)

conj      :: IntSet -> IntSet -> IntSet
(s1 `conj` s2) e = (s1 e) && (s2 e)

disj      :: IntSet -> IntSet -> IntSet
(s1 `disj` s2) e = (s1 e) || (s2 e)

diff      :: IntSet -> IntSet -> IntSet
(s1 `diff` s2) e = (s1 e) && ! (s2 e)

addElem   :: Integer -> IntSet -> IntSet
addElem a s e = (e == a) || (s e)

remElem   :: Integer -> IntSet -> IntSet
remElem a s e = (e <> a) && (s e)

fromSet   :: IntSet -> [Integer] -> [Integer]
fromSet set ls = [x | x <- ls, set x]
```

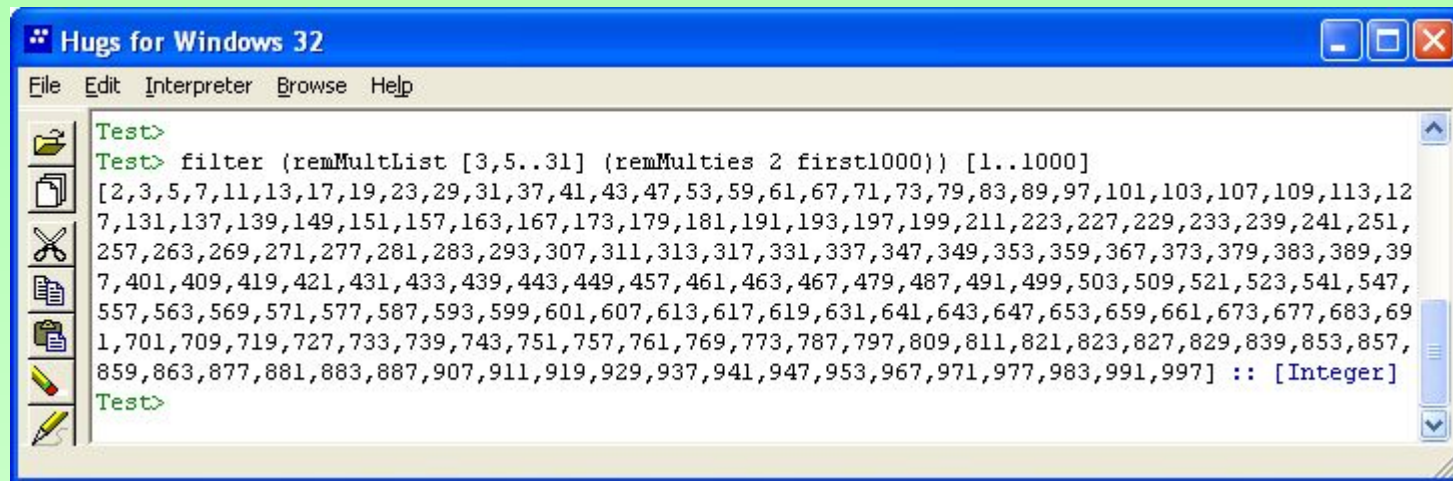
## Программирование с использованием множеств

```
remMulties      :: Integer -> IntSet -> IntSet
remMulties a s e = (s e) && ((e == a) || (e `mod` a /= 0))
```

```
remMultList     :: [Integer] -> IntSet -> IntSet
remMultList ls s = foldr remMulties s ls
```

```
first1000      :: IntSet
first1000 e    = (e >= 2) && (e <= 1000)
```

```
filter (remMultList [3,5..31] (remMulties 2 first1000)) [1..1000]
```



```
Hugs for Windows 32
File Edit Interpreter Browse Help
Test>
Test> filter (remMultList [3,5..31] (remMulties 2 first1000)) [1..1000]
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,12
7,131,137,139,149,151,157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,
257,263,269,271,277,281,283,293,307,311,313,317,331,337,347,349,353,359,367,373,379,383,389,39
7,401,409,419,421,431,433,439,443,449,457,461,463,467,479,487,491,499,503,509,521,523,541,547,
557,563,569,571,577,587,593,599,601,607,613,617,619,631,641,643,647,653,659,661,673,677,683,69
1,701,709,719,727,733,739,743,751,757,761,769,773,787,797,809,811,821,823,827,829,839,853,857,
859,863,877,881,883,887,907,911,919,929,937,941,947,953,967,971,977,983,991,997] :: [Integer]
Test>
```