1 GHCI Commnads

Useful GHCI commands

- :(e |exit) - closes the interpreter;
- :(l |load) file - loads the file;
- :(r |reload) - reloads the last loaded file;
- :(i |info) function - writes information about given function;
- :(t |type) expression - writes information about the type of given expression;
- :(? |help) - prints help for GHCI commands.

2 Basic data types

- Basic comparison operands: `==, /=, <, >, <=, >=`
- Numbers operators: `+, -, *, ^, **`
- `Int`

  ```
  div, mod, abs, negate
  fromIntegral -- converts Integral to Num
  ```

- `Char`

  ```
  ord, chr
  ```

- `import Data.Char -- toUpper, isDigit`
- `Bool` `&&, ||, not, ==`
- `Double, Float`

  ```
  abs, acos, asin, sin, cos, pi,negate
  celing, floor, round, truncate, exp, log,
  ```

3 User-defined types

- Synonyms: `type String = [Char]`
- New type:

  ```
  data Color = Black | White | Red
  ```

  ```
  data Point = Point Float Float
  data Tree1 a = Leaf a
               | Branch (Tree1 a) (Tree1 a)
  ```

4 Function's definition

- Pattern matching - several definitions (equations) with different *patterns*, and *expressions* on the right side.

  ```
  f  pat11  pat12  ...  = rhs1
  f  pat21  pat22  ...  = rhs2
  ...

  factorial 1 = 1
  factorial n = n * factorial (n-1)

  length [] = 0
  length (x:xs) = 1 + length xs
  ```

- Guards

  ```
  max x y | x > y = x
          | otherwise = y
  ```

- Expressions:

  ```
  max x y = if x > y then x else y

  describeList xs = "The list is "++case xs of
         [] -> "empty."
         [x] -> "a singleton list."
         xs -> "a longer list."

  cylinder r h = let sideArea = 2 * pi * r * h
                     topArea = pi * r ^2
                 in  sideArea + 2 * topArea
  ```

- Local definitions:

  ```
  initials first last = [f]++". "++[l]++ "."
      where f = head first
            l = head last
  ```

5 Functions and operators working with lists.

- Accessing list elements

```
head [5,4,3,2,1] -- 5
tail [5,4,3,2,1] -- [4,3,2,1]
last [5,4,3,2,1] -- 1
init [5,4,3,2,1] -- [5,4,3,2]
[1,2,3] !! 2  -- 3
length [5,4,3,2,1] -- 5
null [1,2,3] -- False
null [] -- True
```

- Merging lists

```
[1,2,3] ++ [4,5] --[1,2,3,4,5]
concat [[1,2],[3],[4,5]] -- [1,2,3,4,5]
zip [1,2] [3,4,5] -- [(1,3),(2,4)]
zipWith (+) [1,2] [3,4] -- [4,6]
```

- Lists of numbers

```
minimum [8,4,2,1,5,6] -- 1
maximum [1,9,2,3,4] -- 9
sum [5,2,1,6,3,2,5,7] -- 31
product [6,2,1,2] -- 24
```

- Taking a part of a list

```
take 3 [5,4,3,2,1] -- [5,4,3]
drop 3 [8,4,2,1,5,6] -- [1,5,6]
takeWhile (> 0) [1,3,0,4] -- [1,3]
dropWhile (> 0) [1,3,0,4] -- [0,4]
filter (> 0) [1,3,0,2,-1] -- [1,3,2]
```

- Transforming a list

```
reverse [5,4,3,2,1] -- [1,2,3,4,5]
map (*2) [1,2,3] -- [2,4,6]
```

- Selected *nice* functions

```
4 `elem` [3,4,5,6] -- True
replicate 3 10 -- [10,10,10]
-- cycle and repeat returns infinite  list
take 7 (cycle [1,2,3]) -- [1,2,3,1,2,3,1]
take 7 (repeat 5) -- [5,5,5,5,5,5,5]
foldl (+) 10 [1,2,3] -- ((10+1)+2)+3 = 16
scanl (+) 10 [1,2,3] -- [10, 11, 13, 16]
foldr (-) 10 [1,2,3] -- 1-(2-(3-10)) = -8
scanr (-) 10 [1,2,3] -- [-8,9,-7,10]
foldl1 (-) [1,2,3,4] -- -8
scanl1 (-) [1,2,3,4] -- [1,-1,-4,-8]
foldr1 (-) [1,2,3,4] -- -2
scanr1 (-) [1,2,3,4] -- [-2,3,-1,4]
```