Reference sheet for the exam

November 16, 2021

This sheet will be attached to the examination paper, and lists functions and classes that may be required in answers to the examination questions.

Reference: selected standard functions

Basic functions

- `odd, even :: Integral a => a -> Bool`
  Test whether a number is odd or even
- `null :: [a] -> Bool`
  Test whether a list is empty
- `head :: [a] -> a`
  The first element of a non-empty list
- `tail :: [a] -> [a]`
  All but the first element of a non-empty list
- `last :: [a] -> a`
  The last element of a non-empty list
- `length :: [a] -> Int`
  The length of a list
- `reverse :: [a] -> [a]`
  the reversal of a finite list
- `(++) :: [a] -> [a] -> [a]`
  The concatenation of two lists.
- `zip :: [a] -> [b] -> [(a,b)]`
  List of pairs of corresponding elements of two lists, stopping when one list runs out.
- `take :: Int -> [a] -> [a]`
  The first n elements of the list if it has that many, otherwise the whole list.
- `drop :: Int -> [a] -> [a]`
  The list without the first n elements if it has that many, otherwise the empty list.
- `and :: [Bool] -> Bool`
  `and` returns `True` if all of the Booleans in the input list are `True`.
- `or :: [Bool] -> Bool`
  `or` returns `True` if any of the Booleans in the input list are `True`.
- `product :: Num a => [a] -> a`
  The product of a list of numbers.
- `sum :: Num a => [a] -> a`
  The sum of a list of numbers.
- `concat :: [[a]] -> [a]`
  The concatenation of a list of lists.

Higher order functions

- `map :: (a -> b) -> [a] -> [b]`
  `map f xs` is the list obtained by applying `f` to each element of `xs`:

$$\texttt{map f } [x_1, x_2, \ldots] = [\texttt{f } x_1, \texttt{f } x_2, \ldots\ ]$$

- `filter :: (a -> Bool) -> [a] -> [a]`
  `filter p xs` is the list of elements `x` of `xs` for which `p x` is `True`.

- `iterate :: (a -> a) -> a -> [a]`
  `iterate f x` is the infinite list of repeated applications of `f` to `x`:

$$\texttt{iterate f x} = [\texttt{x}, \texttt{f x}, \texttt{f (f x)}, \ldots\ ]$$

- `takeWhile :: (a -> Bool) -> [a] -> [a]`
  `takeWhile p xs` is the longest prefix of `xs` consisting of elements `x` for which `p x` is `True`.

- `dropWhile :: (a -> Bool) -> [a] -> [a]`
  `dropWhile p xs` is the rest of `xs` after removing `takeWhile p xs`.

Text processing

```
type String = [Char]
```

- `words :: String -> [String]`
  breaks a string up into a list of words, which were delimited by white space.

- `lines :: String -> [String]`
  breaks a string up into a list of strings at newline characters. The resulting strings do not contain newlines.

- `unwords :: [String] -> String`
  joins words, adding separating spaces.

- `unlines :: [String] -> String`
  joins lines, after appending a terminating newline to each.

Character functions

- `isAlpha :: Char -> Bool`
  tests whether a character is alphabetic (i.e. a letter).

- `isUpper :: Char -> Bool`
  tests whether a character is an upper case letter.

- `isLower :: Char -> Bool`
  tests whether a character is a lower case letter.

- `isDigit :: Char -> Bool`
  tests whether a character is a digit.

- `toUpper :: Char -> Char`
  converts lower case letters to upper case, and preserves all other characters.
- `toLower :: Char -> Char`
  converts upper case letters to lower case, and preserves all other characters.

Input/Output

```
type FilePath = String
```

- `readFile :: FilePath -> IO String`
  `readFile f` is an action that reads the contents of the file named `f`.
- `putStr :: String -> IO ()`
  `putStr s` is an action that writes the string `s` to the console.

Selected standard classes

```
class  Eq a  where
    (==), (/=) :: a -> a -> Bool

    x /= y     =  not (x == y)
    x == y     =  not (x /= y)

class  (Eq a) => Ord a  where
    (<), (<=), (>=), (>) :: a -> a -> Bool

class  Show a  where
    show             :: a -> String

class  (Eq a) => Num a  where
    (+), (-), (*)    :: a -> a -> a
    fromInteger      :: Integer -> a
```