Computer Science 716 **Practical Test 2**
Functional Programming: Haskell **20/10/2016**

For this practical test the Haskell code in the files `parsing.hs` and `eval.hs` should be used, and the functionality of the `eval` function should be extended. `Parser`, defined in `parsing.hs`, is a monadic type.

The following basic parsers are provided.

- The parser `item` fails if the input is empty, and consumes the first character otherwise.
- The parser `failure` always fails.
- The parser `p +++ q` behaves as the parser `p` if it succeeds, and as the parser `q` otherwise.
- The parser `return v` always succeeds, returning the value `v` without consuming any input.

Useful functions:

- The function `parse` applies a parser to a string.
- `sat :: (Char -> Bool) -> Parser Char` - to parse a character that satisfies a predicate.
- `char :: Char -> Parser Char` to parse a specific character.
- `many' :: Parser a -> Parser [a]` - to apply a parser zero or more times

Remarks:

- A list with a tuple (instead of the `Maybe` type) is used to indicate the result of parsing.
- Failure of paring is indicated by the empty list.
- The second item in the tuple indicates the unprocessed string after parsing.

Examples:

-
  ```
  > parse item ""
  []
  ```

-
  ```
  parse item "abc"
  [('a',"bc")]
  ```

-
  ```
  > parse failure "abc"
  []
  ```

-
  ```
  > parse (return 1) "abc"
  [(1,"abc")]
  ```

-
  ```
  parse (item +++ return 'd') "abc"
  [('a',"bc")]
  ```

- `parse (failure +++ return 'd') "abc"`
  ```
  [('d',"abc")]
  ```
- `> parse (many' item) "abc"`
  ```
  [("abc","")]
  ```
- `> eval "2*(3+4)"`
  ```
  14
  ```

Attempt as many of the following extensions to the `eval` function as possible.

- Allow integers (or even doubles) in expressions and not only digits.
- Allow (optional) spaces between operators and operands.
- Add additional functions such as `log`, exponentiation and factorial.