Principles of Programming Languages 2017W, Functional Programming

Assignment 1: Recursive Functions in Haskell (4 Points)

Assignment 1.1: Tokenizer (2 Point)

Write a function `tokenize`

```
tokenize :: String -> [String]
```

to split an input string into a list of strings (tokens) as follows:

Input is a string which contains opening and closing rounded brackets and words (or single symbols) separated by white spaces or brackets. In fact, input strings are intended to represent expressions in prefix notation, e.g.,

```
"(+ (* x   y) 1234)"
```

The result list of tokens then contains all the opening and closing brackets and the words separated by white spaces. For example the above input string should give the following list of tokens

```
["(", "+", "(", "*", "x", "y", ")", "1234", ")"]
```

You can use the following functions (see program frame for Assignment 1):

Function `skipWhiteSpaces` skips leading white spaces from an input string

```
skipWhiteSpaces :: String -> String
skipWhiteSpaces (c:cs) | isSpace c   = skipWhiteSpaces cs
skipWhiteSpaces cs                   = cs
```

Function `nextWordAndRest` gets the next word and the rest of the input string. Note, that an empty token `""` means that no further tokens are available.

```
nextWordAndRest :: String -> (String, String)
nextWordAndRest text =
    nextWordAndRest' (skipWhiteSpaces text) ""
    where
        nextWordAndRest' :: String -> String -> (String, String)
        nextWordAndRest' (s:rest)     word | isSpace s   = (reverse word, rest)
        nextWordAndRest' rest@('(':_) word               = (reverse word, rest)
        nextWordAndRest' rest@(')':_) word               = (reverse word, rest)
        nextWordAndRest' []           word               = (reverse word, [])
        nextWordAndRest' (c:rest)     word               = nextWordAndRest' rest (c:word)
```

With those functions `tokenize` is simple and works as follows:

- First skip white spaces from the input string.
- Test if the input string is empty, then you have no more tokens (thus, an empty list of tokens).
- Then test if there is an opening or closing brackets. They give you a next token `"("` or `")"`.
- For any other case, use function `nextWordAndRest` to get the next word and the rest of the input.
- Recursively get the rest of the tokens from the rest of the input string.

Hints:

- A good design idea is to use a local recursive function that actually does the work.
- Use pattern matching and case expressions to distinguish the different cases.

Assignment 1.2: **(2 Points)**

Write a function `areBracketsBalanced`

```
areBracketsBalanced :: [String] -> Bool
```

which tests if the opening and closing brackets in a list of tokens as obtained from function `tokenize` are balanced, that means, if there is a closing bracket for every opening bracket.

Hint:

- Write a recursive function which uses a counter which is increased when encountering an opening bracket and decreased with a closing bracket. At the end of the input list, this counter should be 0 again.

Submit your solution electronically

- by Dec 14, 15:30
- through page http://www.ssw.uni-linz.ac.at/scripts/upload/upload_form.php?lecture=POPL&lang=EN