11.4 Assignment exercises

Exercise 11.4.1 2p

Create a new parser `ws` for consuming whitespace. It should consume zero or more space `' '` or newline `'\n'` characters.

```
Haskell REPL

> runParser ws "abc"
Success "" "abc"
> runParser ws "  abc"
Success "  " "abc"
```

Exercise 11.4.2 2p

Create a new combinator `between :: Parser a -> Parser b -> Parser c -> Parser c` which takes as arguments 3 parsers: `between pHd pTl p`. It runs them in the order `pHd`, `p` and `pTl`, passing the remaining input from the one to the other. If all 3 parsers succeed, it discards the results from `pHd` and `pTl` and returns the result of `p`. If any parser fails the error is returned.

```
Haskell REPL

> runParser (between (char '[') (char ']') (string "abc")) "[abc]xyz"
Success "abc" "xyz"
```

Hints:
Think about how can you use `andThen` to chain 3 parsers. What is the type of the returned parser? How can you *transform* this type to match the required return type (`Parser c`)?
Is there any function that already does part of this work?

Exercise 11.4.3 2p

Create a new parser `ident :: Parser String` that parses an identifier, which:

1. Must contain as least one character
2. Can have as first character letters
3. The rest of the characters can be letters, digits, or a question mark ("?")

```
Haskell REPL

> runParser ident "abc abc"
Success "abc" " abc"
> runParser ident "1abc abc"
Error "Unexpected character '1'"
> runParser ident "abc1 abc"
Success "abc1" " abc"
> runParser ident "empty? abc"
Success "empty?" " abc"
```

Hints:
Think about the following questions:

- How can you ensure that the identifier has at least one character, without checking

the length of the input string?
- Take a look at the `some` combinator and the `string` parser. What do they have in common?

12.2 Assignment

Deadline: Monday, January 11, 23:55

12.3 Submission instructions

1. Unzip the `MiniLisp.zip` folder. You should find:
   - 2 files in the `src` folder:
     - `Parser.hs` - for the basic parsing library that you completed at the last lab
     - `MiniLisp.hs` - for the LISP parser that you partially completed at the last lab
2. Edit the first line of each of the source files as described in the comments.
3. Edit the source files in the `src` folder with your solutions.
4. When done, zip this `MiniLisp` folder and name the zip archive with the following format:

$$MiniLisp\_\langle FirstName\rangle\_\langle LastName\rangle\_\langle Group\rangle$$

   Examples of valid names:
   - `MiniLisp_John_Doe_30432.zip`
   - `MiniLisp_Ion_Popescu_30434.zip`
   - `MiniLisp_Gigel-Dorel_Petrescu_30431.zip`

   Examples of invalid names:
   - `Solutions.zip`
   - `MiniLisp.zip`
   - `Solutii_MiniLisp_Ion_Popescu.zip`

12.3.1 Preparation

Update your existing `MiniLisp.hs` file from the previous lab by adding the new definitions from the `MiniLisp.hs` file from this lab.

12.3.2 Exercises

Exercise 12.3.1 3p

Create a parser `sepBy sep p` with the signature `sepBy :: Parser a -> Parser b -> Parser [b]`

```
Haskell REPL

> runParser (sepBy (char ',') ident) "a,b,c"
Success ["a", "b", "c"] ""
> runParser (sepBy (char ',') ident) "abc"
Success ["abc"] ""
> runParser (sepBy (char ',') ident) "a,,b"
Success ["a"] ",,b"
> runParser (sepBy ws ident) "a  b c     d"
Success ["a", "b", "c", "d"] ""
```

Exercise 12.3.2 — 2p

Create a parser `lispAtom :: Parser lispAtom` which parses a LISP atom (either number or symbol).

```
Haskell REPL

> runParser lispAtom "abc"
Success (Symbol "abc") ""
> runParser lispAtom "123"
Success (Number 123) ""
```

Exercise 12.3.3 — 2p

Create a parser `lispList :: Parser [LispValue]` which parses a list of LISP values, using the `lisp` function (that you will write in Exercise 12.3.4).

```
Haskell REPL

> runParser lispList "(a b c)"
Success [Atom (Symbol "a"),Atom (Symbol "b"),Atom (Symbol "c")] ""
> runParser lispList "(a b c("
Error "Unexpected character '('"
```

Exercise 12.3.4 — 2p

Create a parser `lisp :: Parser LispValue` which parses a LISP value, using the `lispAtom` and `lispList` functions.

```
Haskell REPL

> runParser lisp "(a b c)"
Success (List [Atom (Symbol "a"),Atom (Symbol "b"),Atom (Symbol "c")]) ""
> runParser lisp "123"
Success (Atom (Number 123)) ""
> runParser lisp "abc1"
Success (Atom (Symbol "abc1")) ""
```