Pada pengerjaan Parser Combinator ini, saya mempelajari hal-hal berikut:

1. `newtype` ini hanya memiliki 1 konstruktor dengan satu fungsi untuk meng unwrap Parser menjadi `Maybe (a, String)`. Sebaliknya, tipe Parser ini menerima fungsi `String -> Maybe (a, String)` dan mengembalikan Parser (wrap).

```
newtype Parser a = Parser {parse :: String -> Maybe (a, String)}
```

2. `<$> (Functor)`. Terlihat bahwa <$> menerima fungsi dan menerapkan fungsi tersebut ke dalam parameter wrapper f. Ternyata map ini fmap yang khusus dilakukan kepada list, fmap lebih general, yaitu dapat map function over sebuah arbritary Functor. Agar dapat dilakukan <*> pada Parser, maka perlu dijadikan instance dari Functor,yaitu dengan mendefinisikan fungsi `fmap`.

```
instance Functor Parser where
  fmap f (Parser p) = Parser $ \inp -> do
    (x, input') <- p inp
    Just (f x, input')
jsonNull = (\_ -> JsonNull) <$> stringParser "null" -- Parser String
menjadi Parser JsonValue
```

3. `<*> (Applicative)`. Agar Parser kita dapat menjadi instance sebuah Applicative, dibutuhkan fungsi `pure` dan juga `<*>` atau liftA2. Mirip yah tipenya dengan fmap, bedanya hanya pada inputnya, yaitu ini `f (a -> b)` sedangkan fmap `(a -> b)`. <*> ini berguna karena sifat haskell yang menerapkan `currying`. Applicative dapat dilihat sebagai memasukkan input kepada parser pertama, lalu hasilnya dimasukkan kedalam parser kedua.

```
instance Applicative Parser where
  (Parser p1) <*> (Parser p2) = Parser $ \inp -> do
    (f, inp') <- p1 inp
    (a, inp'') <- p2 inp'
    Just (f a, inp'')
```

Dapat dilihat juga bahwa <*> memasukkan input pada parser pertama, lalu parser ke dua, seperti yang saya jelaskan di atas dimana ini menggunakan konsep `currying`. Pure dari definisinya yaitu a -> f a, digunakan untuk sebagai safety ketika sesuatu, pada konteks parsec ini, bukan sebuah Parser, sehingga diubah penjadi Parser a terlebih dahulu.

4. `sequenceA dan traverse`. Sekarang, sudah ada charParser, namun input kita akan berupa string. Perhatikan tipe berikut! `map charParser "bob" :: [Parser Char]` Yang diinginkan adalah Parser [Char], terbalik! Ternyata sudah ada fungsi untuk handle kasus seperti ini, sequenceA + map, atau traverse. Mereka disebut sebagai inside-out operator. Parser [Char] adalah tipe yang kita inginkan untuk stringParser.

```
sequenceA (map charParser "bob") :: Parser [Char]
traverse charParser "bob" :: Parser [Char]
stringParser = traverse charParse
```

5. `<|> (Alternative)`. Berikut, untuk menerapkan jsonBool kita akan menerapkan interface `Alternative`. Intinya alternative ini dipanggil jika suatu parser fail. Cukup intuitif, karena hanya cek apakah string merupakan salah satu dari dua value, true atau false.

```
jsonBool = f <$> (stringParser "true" <|> stringParser "false") where | f
"true" = JsonBool True | f "false" = JsonBool False
```

6. `<* dan *>`. Mirip dengan <*>, namun perbedaannya adalah dilakukan pembuangan elemen tergantung arah.Ini sangat berguna untuk penerapan `JsonString` dan `JsonArray`, karena mereka berdua dibatasi oleh suatu hal (quotes, brackets possibly whitespace, respectively). Disebut juga sebagai discardLeft dan discardRight. Perhatikan pada jsonString, kita melakukan discardRight ketika ada parser untuk '"' di sebelah kiri, dan sebaliknya discardLeft jika " parser berada di kanan.

```
prefixParser predicate = Parser $ \inp -> Just (span predicate inp)
literalParser = prefixParser (/= '"')
jsonString = JsonString <$> (charParser '"' *> literalParser <* charParser
'"')
```

7. `many`. selama ini hanya dapat melakukan parsing untuk 1 parser saja, namun pada implementasi JsonArray dan JsonObject, dibutuhkan kemampuan untuk parse semua kemungkinan parser pada sebuah array. Ternyata yang dicari adalah fungsi `many :: f a -> f [a]`.

```
*Main Data.Char> parse (many jsonValue) "nulltruefalse\"hello\"" -- parse
sampai fail
Just ([JsonNull,JsonBool True,JsonBool False,JsonString "hello"],"")
split sepParser json = (:) <$> json <*> many (sepParser *> json) <|> pure
[]
```

Kesimpulan

Saya mempelajari cara membuat parser dengan newtype. Cara mengextend interface Functor, Applicative, Alternative. Dilakukan sedemikian agar dibuatnya fungsi-fungsi kecil yang pada awalnya cukup sulit, seperti charParser, prefixParser, dan pluralParser. Namun pada akhirnya sadar bahwa parser-parser kecil ini sangat membantu dalam membuat parser yang lebih besar lainnya. charParser digunakan untuk stringParser, prefixParser untuk digitParser, dan pluralParser untuk jsonArray dan jsonObject.

Selain itu, pembelajaran setengah semester ini membekali saya, yaitu konsep currying dan partial evaluation pada Applicative dan higher order functions pada Functor.