```
> module WinkelBon where
>
> import ParseLib
>
> parse_result p = fst . head . filter (null . snd) . p
>
```

Uitwerkingen Grammatica's en Ontleden ? Deeltentamen 1 (van 2)
Dinsdag 18 december 2007 (15:00-17:00)

1-4
2-2
3-3
4-3
5-2
6-4
7-1
8-4

9

```
1 W        ->  (Produkt Prijs "\n")*
  Produkt  ->  Identifier
  Prijs    ->  Nat "." Nat
```

2

```
> type AbstractBon  =  [(Produkt,Prijs)]
> type Produkt      =  String
> type Prijs        =  (Int,Int)
```

3

```
> spaces = many (symbol ' ')
>
> parseBon  :: Parser Char AbstractBon
> parseBon  =  many ((\produkt _ prijs _ -> (produkt,prijs)) <$>
>                       parseProdukt
>                   <*> spaces
>                   <*> parsePrijs
>                   <*> token "\n"
>                   )
>
>
> parseProdukt  :: Parser Char Produkt
> parseProdukt  =  identifier
>
> parsePrijs  :: Parser Char Prijs
> parsePrijs  =  (\euros _ centen -> (euros,centen))
>            <$> integer
>            <*> symbol '.'
>            <*> integer
>
> broodstring  :: String
> broodstring  =  "brood 3.55\nyoghurt 1.99\nkwark 1.49\n"
>
```

```
> ab  :: String
> ab  =  "melk 1.19\nboter 0.99\ncola 1.79\nkaas 7.49\n"
```

4

```
> totaal :: AbstractBon -> Prijs
> totaal =  (\p -> (p `div` 100, p `mod` 100))
>        .  sum
>        .  map ((\(a,b) -> 100*a+b) . snd)
```

5

```
> printBon      :: AbstractBon -> String
> printBon bon  =
>   let tot = totaal bon
>   in  concat (map (\(produkt,prijs) ->
>                    produkt ++ " " ++ showPrijs prijs ++ "\n")
>                   bon) ++ "\n" ++ "Totaal: " ++ showPrijs tot ++"\n"
>
> showPrijs                  :: Prijs -> String
> showPrijs (euros,centen)  = show euros ++ "." ++
>                             if centen < 10 then "0" else "" ++
>                             show centen
```