Solution Review: Read CSV File

This lesson discusses the solution to the challenge given in the previous lesson.

main.go

products.txt

```go
package main

import (
        "bufio"
        "fmt"
        "log"
        "io"
        "os"
        "strconv"
        "strings"
)

type Book struct {
        title   string
        price   float64
        quantity        int
}

func main() {
        bks := make([]Book, 1)
        file, err := os.Open("products.txt")
        if err != nil {
                log.Fatalf("Error %s opening file products.txt: ", err)
        }
        defer file.Close()

        reader := bufio.NewReader(file)
        for {
                // read one line from the file:
                line, err := reader.ReadString('\n')
                if err == io.EOF {
                        break
                }
                // remove \r and \n so 2 in Windows, in Linux only \n, so 1:
                line = string(line[:len(line)-2])
                //fmt.Printf("The input was: -%s-", line)

                strSl := strings.Split(line, ";")
                book := new(Book)
                book.title = strSl[0]
                book.price, err = strconv.ParseFloat(strSl[1], 32)
                if err!=nil {
```

```
                fmt.Printf("Error in file: %v", err)
            }
            //fmt.Printf("The quan was:-%s-", strSl[2])
            book.quantity, err = strconv.Atoi(strSl[2])
            if err!=nil {
                fmt.Printf("Error in file: %v", err)
            }
            if bks[0].title == "" {
                bks[0] = *book
            } else {
                bks = append(bks, *book)
            }
        }
        fmt.Println("We have read the following books from the file: ")
        for _, bk := range bks {
            fmt.Println(bk)
        }
}
```

Reading CSV File

Look at the file **products.txt**. The *first* field of each line in the file is a **title**, the *second* is a **price**, and the *third* is a **quantity**. Whereas, the columns are separated by a `;`.

Now, look at the file **main.go**. First, we define a struct of type `Book` at **line 13**. It contains the fields according to the specifications of the data in our file: `title`, `price`, and `quality`.

At **line 20**, we make a slice `bks` of `Book`, with a length of **1**. At **line 21**, we open the **products.txt** file. The usual error-handling is done from **line 22** to **line 24**. **Line 25** makes sure the file is closed at the end of the function. At **line 27**, a buffered reader called `reader` is created. This is used in the infinite for loop (see implementation from **line 28** to **line 33**). Here, we read in a line (**line 30**) and jump out of the for loop when the end of the file is reached.

At **line 38**, the line that is read in, is split on the character `;`. The result is an array `strSl`. Let's see how the fields are assigned some values:

- The *first* element of `strSl` is the book's `title`, which we assign at **line 40**.
- The price (which is a floating-point number) is in the field `strSl[1]`, so we have to convert the string input to a float (see **line 41**). The `ParseFloat` method can return an error when the field is not a number

format, and this is handled from **line 41** to **line 44**.

- In the same way, the book's `quantity` (which is an integer number) is in the field `strSl[2]`, so we have to convert the string input to an integer (see **line 46**). The `Atoi` method can return an error when the field is not an integer number, and this is handled from **line 47** to **line 49**.

At **line 50**, we test if there is a `title`. If that's ok (else clause), we append the `book` struct to the `bks` slice. If not ok (if clause), we put that data at the start of the slice, perhaps for review. Then, the slice `bks` is printed out via a for-range loop at **line 58**.

That's it about the solution. In the next lesson, you'll be solving another challenge.