Homework 10: Parsing, Printing, and Evaluation

For this assignment, you will write a parser, printer, and evaluator for arithmetic expressions. To do so, you will (1) learn how to read a *BNF grammar*, (2) learn how to use Scala's parser-combinator library, and (3) use property-based testing, using Scalatest.

1 Preliminaries

You should create a directory-tree that looks like this:

```
./parsing
├── build.sbt
├── project
│   └── plugins.sbt
└── src
    ├── main
    │   └── scala ..... Your solution goes here
    └── test
        └── scala ..... Yours tests go here
```

Your `build.sbt` file must have exactly these lines:

```
resolvers += "PLASMA" at "https://dl.bintray.com/plasma-umass/maven"
libraryDependencies += "edu.umass.cs" %% "compsci220" % "1.1.0"
```

The `project/plugins.sbt` file must have exactly this line:

```
addSbtPlugin("edu.umass.cs" % "cmpsci220" % "3.0.1")
```

The support code for this assignment is in the package `hw.parsing`.

2 Overview

For this assignment, you will work with a language of arithmetic expressions: numbers, addition, subtraction, multiplication, and division. Here are some examples of the *concrete syntax* of the language:

- `1 + 2`
- `10`
- `2 * 3 + 5 * -10`
- `2 * (3 + 5) * -10`

$$
\begin{array}{rcl}
\textit{number} & ::= & \texttt{-?[0-9]+(.[0-9]+)?} \\
\textit{atom} & ::= & \textit{number} \\
 & | & \texttt{(}\textit{expr}\texttt{)} \\
\textit{exponent} & ::= & \textit{atom} \\
 & | & \textit{exponent}\texttt{\^{}}\textit{atom} \\
\textit{mul} & ::= & \textit{exponent} \\
 & | & \textit{exponent}\texttt{*}\textit{mul} \\
 & | & \textit{exponent}\texttt{/}\textit{mul} \\
\textit{add} & ::= & \textit{mul} \\
 & | & \textit{mul}\texttt{+}\textit{add} \\
 & | & \textit{mul}\texttt{-}\textit{add} \\
\textit{expr} & ::= & \textit{add}
\end{array}
$$

Figure 33.1: Grammar of arithmetic expressions

- `2 * (3 + 5) ^ 2 * -10`

More formally, the concrete syntax of the language is defined using the grammar in fig. 33.1.

Your first task is to implement a parser that parses strings to the `Expr` type. For example, `parse("1 + 2")` should produce `Add(Num(1), Num(2))`. To do so, you will use Scala's parser combinator library with Packrat parsing.

Your second task is to implement a printer, which returns strings that represent arithmetic expressions. An important property of the printer is its relationship with the parser:

`parseExpr(print(e)) == e`, for all expressions e.

It is tedious to write test cases for this property, since there are so many different kinds of expressions. Instead, we will use *ScalaCheck* to test this property on randomly generated expressions.

Finally, for completeness, you'll write an evaluator for arithmetic expressions.

3 Programming Task

You should use the template in fig. 33.2 for your solution.

We suggest proceeding in the following order:

1. Implement `ArithEval`. This is a simple recursive function.
2. Implement `ArithParser` by translating the grammar provided above to Scala's parser combinators.
3. Implement `ArithPrinter`.

We suggest using ScalaCheck to test these functions. (You'll have to define generators as part of your test suite.)

4 Check Your Work

Figure 33.3 is a trivial test suite that simply ensures that you've defined the parser, printer, and evaluator with the right types.

```
import hw.parsing._
import scala.util.parsing.combinator._

object ArithEval extends ArithEvalLike {
  def eval(e: Expr): Double = ???
}

object ArithParser extends ArithParserLike {

  // number: PackratParser[Double] is defined in ArithParserLike

  lazy val atom: PackratParser[Expr] = ???

  lazy val exponent: PackratParser[Expr] = ???

  lazy val add: PackratParser[Expr] = ???

  lazy val mul: PackratParser[Expr] = ???

  lazy val expr: PackratParser[Expr] = ???
}

object ArithPrinter extends ArithPrinterLike {
  def print(e: Expr): String = ???
}
```

Figure 33.2: Template for the parser, printer, and evaluator.

```
class TrivialTestSuite extends org.scalatest.FunSuite {

  test("several objects must be defined") {
    val parser: hw.parsing.ArithParserLike = ArithParser
    val printer: hw.parsing.ArithPrinterLike = ArithPrinter
    val eval: hw.parsing.ArithEvalLike = ArithEval
  }

}
```

Figure 33.3: Your solution must pass this test suite with no modifications.

5 Hand In

From the `sbt` console, run the command `submit`. The command will create a file called `submission.tar.gz` in your assignment directory. Submit this file using Moodle.

For example, if the command runs successfully, you will see output similar to this:

```
Created submission.tar.gz. Upload this file to Moodle.
[success] Total time: 0 s, completed Jan 17, 2016 12:55:55 PM
```

Note: The command will not allow you to submit code that does not compile. If your code doesn't compile, you will receive no credit for the assignment.