Homework 4: Scala Overview

Due: problems 1-4, November 3, 2005; remainder November 10, 2005.

In this homework you will learn: the basics of Scala, how Scala can be used as a functional programming language, how its features make it extensible, and how it can be used for XML processing.

As you work the problems in Scala, think about what about Scala makes it good or bad as a next programming language. See the problems in Section 5, which you should think about as you work on the other problems.

Since the purpose of this homework is to ensure skills in functional programming, this is an individual homework. That is, for this homework, you are to do the work on your own (not in groups).

1 Some Installation Tips

Scala is installed on the machines in 115 Atanasoff if you want to use these machines. If you want to use Scala on your own machine, the following may be useful.

The tips in this section pertain to Scala 1.4.0.2 and the Scala Eclipse Plugin version 1.0.3.

1.1 Using the Scala Eclipse Plugin

It seems that the Scala Eclipse plugin doesn't work with Eclipse 3.1, which is the latest version of Eclipse. Use Eclipse 3.0.2 instead, which you can install from `http://eclipse.org/downloads/index.php`.

To get the Eclipse plugin working, first follow the directions in the Scala Eclipse plugin web page `http://scala.epfl.ch/docu/eclipse/`. These have you run programs as an *external* tool. To do this follow the instructions in the "Test Example" section of the Scala Eclipse plugin web page.

To get the files for this homework into Eclipse, do the following. You should, after following the instructions described above, have a project, call it "ScalaDev," and a source directory in that project, `src`. To get the files from the homework zip file into Eclipse, copy them files by hand into the src directory, which you could do by unzipping from outside Eclipse and letting the files fall into `eclipse/workspace/ScalaDev`, which will put the packages inside src. Then start Eclipse and do a Refresh (from the File menu) on the ScalaDev project.

1.2 Using the Command Line

Another way to run the tools is by using the command line scripts `scalac` and `scala`. See the instructions for the individual problems for how to do this. But one example is to fist change into the package where you are working (i.e., into `src/scalahw`) and then execute the following.

```
scalac -classpath . -d ../bin scalahw/ListExamplesTest.scala
scala -classpath ../bin scalahw.ListExamplesTest
```

You can also use and adapt the `Makefile` supplied (even on a Windows machine, if you have cygwin installed).

2 Functional Programming in Scala

You will probably find it helpful to read *Scala By Example* [Ode05], and to use *The Scala Language Specification* [OAC^{+}05] as a reference. You can get all of these from `http://scala.epfl.ch/docu/`.

Make a package `scalahw`, and put your code for all of the programming problems below in that package.

Testing harnesses and other code for these problems are found on the web and on the department machines in `/home/course/cs541/public/homework/hw4/`. If you are using Eclipse, you can import this code into Eclipse using the "Import" function from the File menu.

1. (15 points) In Scala, write an `object` declaration `scalahw.ListExamples` (i.e., `ListExamples` in package `scalahw`) that satisfies the requirements of the following trait.

   ```
   // @(#)$Id: ListExamplesProblem.scala,v 1.1 2005/10/28 09:03:35 leavens Exp leavens $
   package scalahw;
   trait ListExamplesProblem {
     /** Delete all elements of a list matching e.
      * This is done as a for-comprehension. */
     def delete_all[T](e: T, xs: List[T]): List[T];

     /** Delete all elements of a list matching e.
      * Using library functions. */
     def delete_all_b[T](e: T, xs: List[T]): List[T];

     /** Delete all elements of a list matching e.
      * Using explicit recursion and pattern matching. */
     def delete_all_c[T](e: T, xs: List[T]): List[T];
   }
   ```

   That is, each of the three versions of `delete_all` is a polymorphic function that takes an item, `e`, of type `T` and a list `xs`, of type `List[T]`, and returns a list just like the argument list, but with the each occurrence of the item (if any) removed.

   As in the Haskell homework, you are to do this in three different ways: (a) using a for-comprehension, (b) by using standard functions available in Scala, and (c) by writing out the recursion yourself, using pattern matching to help.

   Your solution should use `scalahw.ListExamplesProblem` as a mixin, so that you do not have to repeat the type declarations in the above trait in your solution. That is your class `scalahw.ListExamples` should follow the outline below.

   ```
   package scalahw;
   /* ... */
   object ListExamples with ListExamplesProblem { /* ... */ }
   ```

   To test your code run the object `scalahw.ListExamplesTest`, which can be consulted for examples. (The file containing this object, and files containing the mixins and classes it uses, are available from the course directory, as described above.)

   Hand in a printout of the code your wrote and the output of your testing.

2. (30 points) Write, in Scala, a solution to the library database problem we worked in Haskell (problem 5.13 in the second edition on Thompson's book). That is you are to write an object `scalahw.LibraryDB` that satisfies the requirements of the trait below.

   ```
   // @(#)$Id: LibraryDBProblem.scala,v 1.2 2005/11/01 20:21:54 leavens Exp leavens $
   package scalahw;
   trait LibraryDBProblem {
     type Person = String;
     type Book = String;
   ```

```
  type Database = List[Pair[Person, Book]];

  /** "Given a book, find the borrower(s) of the book, if any." */
  def borrowers (db: Database)(bk: Book): List[Person];

  /** "Given a book, find out whether it is borrowed." */
  def borrowed (db: Database) : Book => Boolean;

  /** "Given a person, find out the number of books he or she has borrowed." */
  def numBorrowed (db: Database)(per: Person): Int;
}
```

Your solution should use `scalahw.LibraryDBProblem` as a mixin, so that you do not have to repeat the type declarations in the above trait in your solution.

To test your code run the object `scalahw.LibraryDBTest`, which can be consulted for examples. (The file containing this object, and files containing the mixins and classes it uses, are available from the course directory, as described above.)

Hand in a printout of the code your wrote and the output of your testing.

3. This problem relates to modularization of numeric code using functional techniques and lazy evaluation. As in the Haskell homework, the code explores the Newton-Raphson algorithm. This algorithm computes better and better approximations to the square root of a number `n` from a previous approximation `x` by using the function `next`, provided in the following trait.

```
// @(#)$Id: NumericCodeProblem.scala,v 1.1 2005/10/28 09:03:35 leavens Exp leavens $
package scalahw;
import Stream._;
trait NumericCodeProblem {
  /** Iteration of a function, starting with a given argument. */
  def iterate[T](f: T => T)(x:T): Stream[T] = cons(x, iterate(f)(f(x)));

  /** Next approximation for square root compuation. */
  def next(n: Double)(x: Double) = (x + n / x) / 2.0;

  /** Sequence of square root approximations */
  def approximations(n: Double): Double => Stream[Double];

  /** Find an approximation within epsilon. */
  def within(epsilon: Double)(str: Stream[Double]): Double;

  /** Compuate square roots to within epsilon. */
  def squareRoot(a0: Double)(epsilon: Double)(n: Double): Double;

  /** Infinite list of doubles, starting at n.
   * (Adapted from Scala by Example, by Odersky) */
  def from(n: Double): Stream[Double] = cons(n, from(n + 1.0));
}
```

That is you are to write an object `scalahw.NumericCode` that satisfies the requirements of the trait above.

Your solution should the above trait, `scalahw.NumericCodeProblem`, as a mixin, so that you do not have to repeat the provided functions (such as `iterate` and `next`).

Here are some more details about what you are to do.

(a) (10 points) The function `approximations` is curried, and `approximations(n)(a0)` returns the infinite list of approximations to the square root of `n`, starting with `a0`, computed by iterating `next`. For example we would have the following (where the output has extra line breaks added for readability):

```
approximations(1.0)(1.0).take(5)
  ==> [1.0, 1.0, 1.0, 1.0, 1.0]
approximations(2.0)(1.0).take(5)
  ==> [1.0, 1.5, 1.4166666666666665, 1.4142156862745097,
       1.4142135623746899]
approximations(64.0)(1.0).take(7)
  ==> [1.0, 32.5, 17.234615384615385, 10.474036101145005,
       8.292191785986859, 8.005147977880979, 8.000001655289593]
```

(b) (20 points) The function `within` takes a tolerance, that is, a number `epsilon`, and an infinite stream of `Double`s, and looks down the stream to find two consecutive numbers in the stream that differ by no more than `epsilon`; it returns the second of these. (It might never return if there is no such pair of consecutive elements.) For example, given the following definitions,

```
  import Stream._;
  val all8s: Stream[Double] = cons(8.0, all8s);
  val myStream =
      cons(1.0, cons(32.5, cons(17.2346, cons(10.474,
           cons(8.29219, cons(8.00515, all8s))))));
```

we have the following examples of `within`:

```
within(1.0)(from(1.0))
      ==> 2.0
within(0.5)(myStream)
      ==> 8.00515
```

(c) (10 points) Using the two pieces above, the function `squareRoot` takes an initial guess, a tolerance `epsilon`, and a double, `n`, and returns an approximation to the square root of `n` that is within `epsilon`. The following are examples, if you ignore the insignificant digits in the first example's output.

```
squareRoot(1.0)(0.0000001)(2.0)
      ==> 1.414213562373095
squareRoot(1.0)(0.0000001)(64.0)
      ==> 8.0
```

(d) (15 points) For this problem we will have you write testing code yourself, both to give you the experience, and to avoid problems with the sensitivity of floating point computations to different algorithms. Write your testing code in an object, `scalahw.NumericCodeTest` to test the above parts.

Hand in a printout of the code your wrote, including the testing code, and the output of your testing.

4. (40 points) In this problem you will implement the following trait in Scala.

```
// @(#)$Id: FSEntityProblem.scala,v 1.1 2005/10/28 09:03:35 leavens Exp leavens $
package scalahw;

trait FSEntityProblem {

  type EName = String;
  type Contents = String;
  type Path = List[EName];
  type Map[A,B] = scala.collection.mutable.Map[A,B];

  /** File system entities */
  trait FSEntity {
    /** Says whether this FSEntity is okay. */
    def okFSEntity: Boolean;
  }

  /** Things with a fetch method (directories). */
  trait Fetchable {
    /** Fetch the FSEntity along p from this,
     * or throw an exception if can't do that. */
    def fetch(p: Path): FSEntity;
  }

  // You have to implement the above two methods
  // for each of the following two subtypes of FSEntity.
  // Your definition should also produce the functions required below.
  // Hint: Use case classes.

  /** Files, with contents c. */
  type File <: FSEntity;
  def File(c: Contents): File;

  /** Directories, with map from names to FSEntities m. */
  type Dir <: FSEntity with Fetchable;
  def Dir(m: Map[EName, FSEntity]): Dir;
}
```

Your solution, an object `scalahw.FSEntity`, should use `scalahw.FSEntityProblem` as a mixin, so that you do not have to repeat the type declarations in the above trait in your solution.

An `FSEntity` is either a *file*, if it has the form `File(`s`)`, or a *directory*, if it has the form `Dir(`m`)`.

You have to implement the types `File` and `Dir` as classes (make them case classes). After that, you must write the two methods, `okFSEntity` and `fetch` declared in the two nested traits of the above trait. The two methods are described as follows.

(a) (20 points) The method `okFSEntity` is available in both files and directories. It determines if the receiver is *ok* in the sense that, if the receiver is a directory of the form `Dir(`m`)`, then all of the following must hold:

- the map m is not null,
- one of the entity names of m is the special name `".."` (which names the directory's parent directory), and
- the value of the `FSEntity` associated with the name `".."` is a directory.

There are no other conditions on a file system entity being "ok". Thus, an file is always ok. Since a directory entity may refer to itself (the structure may be circular), it is *not* part of checking that an entity is ok to check that subdirectories contained in the directory are ok (as that could lead to infinite loops). Entity names in an ok directory may be empty and may contain arbitrary characters.

(Note that, unlike the problem on the Haskell exam, there is no distinct names requirement, as maps automatically enforce that. However, you need to check that in `Dir(`m`)`, m is not null.

To test your code run the object `scalahw.FSEntityTest`, which can be consulted for examples. (The file containing this object, and files containing the mixins and classes it uses, are available from the course directory, as described above.)

(b) (20 points) The method `fetch` is available only for directories (and cannot be called on a file). When writing the `fetch` method, you should assume that each directory that is a receiver of this method satisfies the predicate `okFSEntity` from the previous problem.

The directory receiving the `fetch` method call can be thought of as the current directory. The path argument, `p`, is a list of strings representing, from left to right, a path through the directory structure, starting at the current directory. The `fetch` method either returns an `FSEntity` that it finds by following the path, or it throws an `IllegalArgumentException`. You can assume that the list `p` is non-empty.

To test your code run the object `scalahw.FSEntityTest`, which can be consulted for examples. (The file containing this object, and files containing the mixins and classes it uses, are available from the course directory, as described above.)

Note that if you do testing on your own for this problem, you have to be careful to avoid infinite loops, as the directories (that are ok) always contain a recursion in their data. For example, the root directory will have itself as its parent directory. Thus, you have to avoid printing these directories and also you have to avoid structural comparisons.

3 Extensibility Problems

One of the goals of a significant body of work on programming languages has been to solve the "expression problem" [Coo91, KFF98, Rey78, Wad98, ZO05], which has two conflicting goals. The first goal is to allow extension of the functionality of a program to handle new types of data, even data that may recursively extend existing data types, but without editing the old program. The second goal is to do this in a way that can be statically type checked.

Programming language researchers often discuss the expression problem in the context of extending an interpreter for a programming language with new features. (See, for example, the 1992 POPL paper by Wadler [Wad92], which Steele responded to in POPL 1994 [Ste94], and which Liang, Hudak, and Jones improved upon in POPL 1995 [LHJ95].)

The expression problem has practical implications also for teaching programming languages; for example, the book *Essentials of Programming Languages* (EOPL), by Friedman, Wand, and Haynes [FWH01] takes students through a series of programming language interpreters for successively more complex languages. Although the book concentrates on the differences between each interpreter and the previous one, the actual code is produced by editing previous interpreters, instead of by just writing down the differences. (This is true despite the lack of type checking in the implementation language, Scheme.)

5. (50 points) In this problem you will extend the simplest of the EOPL interpreters (see above) with three simple features:

   (a) A larger initial environment, which includes a binding from the symbol `zero` to the number 0. Thus the meaning of the variable reference expression `zero` should be the number 0.

   (b) A division primitive, with concrete syntax `/`. This primitive allows the user to divide one expressed value (i.e., a number) by another expressed value. For example, the value of a primitive application expression such as `/(10,2)` is 5, since 5 is 10 divided by 2.

   (c) An `if` expression, with concrete syntax `if` E_1 `then` E_2 `else` E_3, where E_1, E_2, and E_3 are expressions. The value of `if` E_1 `then` E_2 `else` E_3, is the value of E_2 if E_1 has a value that is different from 0, and is the value of E_3 otherwise. For example the meaning of `if zero then 3 else add1(4)` is 5.

We have provided the base interpreter that you are to extend in the package `interp` (which you can obtain from the course homework directory). The main code is in a class `interp.Interpreter`. Its its tests are in `interp.InterpreterTest`, which also uses classes from the `testing` package. The parser for this initial language is in `interp.EOPLParsers`. The abstract syntax is contained in `interp.Program`, `interp.Expression`, and `interp.Primitive`. You should read that code, including its tests to get an idea of how the interpreter works.

Your task is to extend this base interpreter with the three features described above, without changing any code in the package `interp`. To enforce this, you are to solve this problem by only writing code in a new package, `interp2`.

You should write only the minimum amount of code in `interp2`, without duplicating any of the code in `interp`. (Hint, use `super` calls.) You should write files in this package that have the same name as those in the package `interp`, counting on the package names for disambiguation where necessary.

We have provided tests in `interp2.InterpreterTest` for you. Note that the class `ExpTest` referred to in `interp2.InterpreterTest` is the class `interp2.ExpTest`, you will need to write that class.

(The files for the packages `interp` and the start of `interp2` are available from the course directory, as described above.)

You should turn in a printout of all the code you write in the `interp2` package and the results of running our tests. You don't need to print out the code in the `interp` package, since you shouldn't be changing that.

A large part of this problem is figuring out what to do (i.e., how to use Scala's mechanisms to make the extension work). You can get some hints from the earlier problems in this homework. However, if you have trouble understanding the parsing, let the staff know, and we can give more hints on that.

6. Here are some possibilities for more exploration about the expression problem.

   (a) (50 points extra credit) Are there any barriers to continuing such a sequence of extensions to further levels? In this problem you will see by making more extensions on top of the extensions of the previous problem. However, work in in yet another new package, `interp3`. For this problem, add `let`-expressions and anonymous functions. Write your own tests and print out the results of testing and the contents of the package `interp3`.

   (b) (100 points extra credit) Several authors have noted that it's more difficult to combine independently-developed extensions to a base program into a program that contains all the extensions. Explore this in Scala by making an independent extension to the `interp` package (that is, one that does not rely on `interp2` or `interp3`). Call the new extension package `interp2b`. In `interp2b`, add support for `case`-expressions (or `cond`-expressions). Then develop a package `interpboth` that combines the extensions in `interp2` and `interp2b`. What problems arise, if any? Do you have to change any existing code to make things work?

4 XML Problems

For additional material on Scala and XML programming, we recommend reading Burak Emir's projects page, `http://lamp.epfl.ch/~emir/projects/`, and in particular his book draft on Scala and XML (`http://lamp.epfl.ch/~emir/projects/scalaxbook.docbk.html`).

7. (30 points) Write a Scala program (i.e., an application), `scalaxml.ListDir`, when called with a path to a directory as an argument, produces a web page in XHTML 1.0 Strict format on the program's standard output that lists each file and directory (other than `.` and `..`) in a bulleted list.

   For example, when my solution is run from my `src/scalaxml` directory as follows:

   ```
     scalac -classpath .. -d ../../bin *.scala
     scala -classpath ../../bin scalaxml.ListDir .. > listdir.html
   ```

   passing it the string `..` as an argument, it produces the following output, which the above command captures in the file `listdir.html`.

   ```
   <?xml version="1.0" encoding="iso-8859-1"?>
   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
   <html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
     <head>
       <title>Directory ..</title>
     </head>
     <body>
       <h2>Directory ..</h2>
       <ul>
         <li>interp</li>
         <li>interp2</li>
         <li>scalahw</li>
         <li>scalaxml</li>
         <li>testing</li>
         <li>xhtml1_strict.dtd</li>
       </ul>
     </body>
   </html>
   ```

   Your solution must use the XML syntax feature of Scala, and should print from an instance of (some subtype of) `scala.xml.Node` that you form. You'll find it's much easier if you do that (as opposed to creating objects for the XML tree or directly outputting Strings).

   You must structure your solution in (at least) two files. The first file must define a class `scalaxml.DirectoryList`, which has a constructor that takes a `java.io.File` argument. This object should be instantiated by the object, `scalaxml.ListDir`, which will be defined in the second file. This object contains the main program that reads the command line argument and uses the class to do the listing. The object should be responsible for formatting the output and printing it to standard output, the class should have a method that assembles the XML for the web page's html element as an object; i.e., everything between `<html>` and `</html>` in the output is just formatted from Scala object constructed in a method of the `DirectoryList` class.

   Note that your solution must make the name of the directory (`..`) appears in both the title and in the heading of the generated XHTML output.

   You can assume that the program is given exactly one argument. It's okay to have the program die (e.g., with an unhandled exception) if the argument is not a directory.

   Although the items in the directory in the output shown above seem to be in sorted order, this is not required. (Perhaps the JDK on Windows does this automatically, so you might see something different on a different operating system.) Also the formatting of the output, with indentation and helpful line breaks, doesn't have to appear as above (although that is helpful for debugging), since web browsers don't care about that.

Some hints: note that this problem has no relation to the `FSEntity` problem above: in this problem we are using real files and directories, as presented by Scala (through Java or .NET), not `FSEntity` objects.

In my solution, I used the `java.io.File` methods `getName()` and `list()`.

You will find that you have to output the first 3 lines of the output directly from strings, instead of from objects; currently the `scala.xml.dtd.DocType` class doesn't print properly according to the W3C validator. I also wasted a bunch of time trying to make a XML document with type `scala.xml.Document`, to attach the dtd and document type, but these don't seem to affect printing, and hence should be ignored.

You can use the class `scala.xml.PrettyPrinter` to make the output be formatted (to help debugging). However, this class seems to have a bug that affects lines of a certain length; it causes an out of memory error. If that happens to you, the workaround is to adjust the size of the lines to be longer until it doesn't affect you. (I filed a bug report on this.)

For more about XHTML 1.0 Strict format, `http://www.w3.org/TR/xhtml1/`. The formal definition or DTD is `http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd`.

You should hand in a printout of all your code, including a printout of the XHTML generated as part of your testing. However, you may want to hold off printing your code until you solve the next problem (see below).

Use the free Validator service `http://validator.w3.org/` to check the output of your program conforms to the standard for XHTML 1.0 Strict documents.

8. (30 points) Write a Scala program (i.e., an application), `scalaxml.ListImages`, when called with a path to a directory as an argument, produces a web page in XHTML 1.0 Strict format on the program's standard output that contains an `<img src="..."></img>` element for each image file in the given directory, where the `src` attribute is a path to the image. Aside from the title and heading, nothing else should appear in the web page produced.

   Image files are defined, for purposes of this problem, as those with a file name suffix of `.jpg`, `.JPG`, `.gif`, or `.GIF`.

   For example, when I run the program from my `src/scalaxml` directory as follows:

```
  scalac -classpath .. -d ../../bin *.scala
  scala -classpath ../../bin scalaxml.ListImages "../../MyPictures" >listimages.html
```

   passing it a path to the "MyPictures" directory as an argument, it produces the following output, which the above command captures in the file `listimages.html`.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Images from MyPictures</title>
  </head>
  <body>
    <h2>Images from MyPictures</h2>
    <p>
      <img alt="Cahokia_Pan.jpg" src="../../MyPictures/Cahokia_Pan.jpg"></img>
      <img alt="Endurance_br2.jpg" src="../../MyPictures/Endurance_br2.jpg"></img>
      <img alt="hills.jpg" src="../../MyPictures/hills.jpg"></img>
      <img alt="LionKingPan.JPG" src="../../MyPictures/LionKingPan.JPG"></img>
      <img alt="pan-A074R1_br.JPG" src="../../MyPictures/pan-A074R1_br.JPG"></img>
    </p>
  </body>
</html>
```

Your solution must use the XML syntax feature of Scala, and should print from an instance of (some subtype of) `scala.xml.Node` that you form.

Note that the title and heading are slightly different from the previous problem.

You should make sure that, even if your program is run on a Windows machine, where the result of calling the `getPath()` method on a `File` object comes out with backslashes (\), that your output does not contain backslashes inside the `<img src="..."></img>` elements.

To make this problem more interesting, you must implement your solution using a class and an object, as in the previous problem. Furthermore, the class should extend the class you used in the previous problem. (Of course, there is no extension relation among the objects.) Try to make the minimal changes to the previous problem's code to make that work. The object, of course, will be `scalaxml.ListImages`. Call the class `ImagesList`, and hence it should start as follows.

```
class ImagesList(file: File) extends DirectoryList(file) /* ... */
```

Again, you can assume that the program is given exactly one argument, which names a directory.

Although the output above seems to list the images in sorted order, this is not required. Also the formatting of the output, with indentation and helpful line breaks, doesn't have to appear as above.

Some hints: In my solution, I found it helpful to use the `matches` and `replace` methods of `java.lang.String`. Also, see the previous problem for notes about the class `PrettyPrinter` and for more about XHTML.

You should hand in a printout of all your code, including a printout of the XHTML generated as part of your testing. (You should test with directories that contain some files or directories that are not images.)

Use the free Validator service `http://validator.w3.org/` to check the output of your program conforms to the standard for XHTML 1.0 Strict documents.

9. "Slide shows" are a more friendly way to examine pictures, as they don't require a browser to load all pictures at once.

   (a) (50 points extra credit) Implement a slide show as a Scala program that shows the first image in a directory (using `<img src="..."></img>` for the image), and has a link to a web page showing the next image, etc.

   (b) (50 points extra credit) Implement the slide show as an application that shows each image directly, and doesn't need to generate web pages. You will have to hook into your platform's GUI technology (e.g., Swing in Java) to do this.

10. (70 points) There is now an RSS 2.0 file for the course web page. Your task in this problem is to write a Scala program that, when called with one argument, the URL of the course web page (`http://www.cs.iastate.edu/~cs541/index.shtml`), generates a file that closely resembles the RSS feed for the course, which is found in the file `http://www.cs.iastate.edu/~cs541/rss.xml`. or `rss.xml` in the course directory `/home/course/public/cs541/`.

    In more detail, your program `scalaxml.Web2RSS`, should take the URL for a web page as a command line argument. It can assume that this web page is in XHTML format, and thus can be loaded as an XML file. It should search for the (first) `<table>` element in that web page, and convert that into a RSS file. See the source for the `rss.xml` file for the format of this file.

To convert the web page to the RSS file, one has to convert each non-header row in the web page's `<table>` into an `<item>` in the RSS file. The RSS file also contains some surrounding elements, which are always the same, except for the `<lastBuildDate>`.

To convert a row in the web page into an item in the RSS file, one does the following. The first element in each row (an `img` element) is ignored. The second element in the row, the date, is used as part of the `<description>` in the item. The third element in the row, some text with a one or more links, is placed in the `<description>` following the date, but with all links (`<a href="...">` and `</a>`) removed. The text in each link is left in the description. Moreover, the text of the first link in the description is used to form the `<title>` of the item, and the `href` attribute contained in that first link (i.e., the `a` element) is used as the `<link>` in the news item.

For example, the table row

```
<tr>
  <td valign="top">
    <img src="images/new.gif" alt="new news item" />
    </td>
  <td valign="top">10/31/2005</td>
  <td><a href="index.shtml">This web page</a>
     now has a <a href="rss.xml">RSS feed</a>.
    </td>
  </tr>
```

is converted into the following item.

```
<item>
  <title>This web page</title>
  <link>http://www.cs.iastate.edu/~cs541/index.shtml</link>
  <description>10/31/2005 - This web page now has a RSS feed.
  </description>
  <source url="http://www.cs.iastate.edu/~cs541/rss.xml">
    Home Page of Com S 541</source>
</item>
```

Note that the formatting won't necessarily come out exactly as shown in your output.

Hint: it may be helpful to use the Java classes: `java.util.Date`, `java.text.SimpleDateFormat`, and `java.net.URI`. In particular, I used the `resolve` method in `java.net.URI` (N.B., the class in question is named URI, not URL[1]), for resolving the URL against the web page's URL, to get an "absolute" URL from what might be a relative one in the link.

See `http://blogs.law.harvard.edu/tech/rss` for information about RSS 2.0.1.

Hand in a printout of your program's files and the output generated by running it on the 541 web page (`http://www.cs.iastate.edu/~cs541/index.shtml`).

5 Evaluation of Scala as a Language

11. (20 points) Compare and contrast Scala and Haskell. What advantages does Haskell have over Scala? What kinds of programming tasks are better served in Haskell versus Scala?

12. (10 points) Compare Scala's support for code reuse to that in Haskell.

[1] A URL is a particular kind of URI, although the Java classes in the JDK aren't directly related to each other in a subtype relationship.

13. (30 points) Based on your answer to the previous problem, list (all) the features of Scala's design that you would like to include in the design of "the next programming language." For each feature, briefly state why it would be helpful to include that in a language that would be a successor to Java.

14. (30 points; extra credit) What features of Scala's design would you omit from the design of "the next programming language?" For each feature, briefly explain your reasons.

15. (20 points; extra credit) What features of Scala do you think would be difficult for the average programmer to understand and use on a daily basis (even with reasonable tutorial materials and training)?

6 Extra Credit Research Problems

16. (50 points total; extra credit) Find and read a research article about Scala or about programming language designs that explore some of the type system or language design issues related to Scala. For example, you might find a paper about Scala's type system, for example on the Scala related resource pages (`http://scala.epfl.ch/docu/related.html`).

    Write a short (1 or 2 page maximum) review of the article, stating:

    - (10 points) what the problem was that the article was claiming to solve,
    - (20 points) the main points made in the article and what you learned from it,
    - (20 points) what contribution it make vs. any related work mentioned in the article.

    In your writing, be sure to digest the material; that is, don't just select various quotes from the article and string them together, instead, really summarize it. If you quote any text from the paper, be sure to mark the quotations with quotation marks (" and ") and give the page number(s).

    Please hand in a copy of the article with your review.

References

[Coo91] William R. Cook. Object-oriented programming versus abstract data types. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages, REX School/Workshop, Noordwijkerhout, The Netherlands, May/June 1990*, volume 489 of *Lecture Notes in Computer Science*, pages 151–178. Springer-Verlag, New York, NY, 1991.

[FWH01] Daniel P. Friedman, Mitchell Wand, and Christopher T. Haynes. *Essentials of Programming Languages.* The MIT Press, New York, NY, second edition, 2001.

[KFF98] Shriram Krishnamurthi, Matthias Felleisen, and Daniel P. Friedman. Synthesizing ojbect-oriented and functional design to promote re-use. In Eric Jul, editor, *ECOOP '98—Object-Oriented Programming, 12th European Conference, Brussels, Belgium, Proceedings*, volume 1445 of *Lecture Notes in Computer Science*, pages 91–113, New York, NY, 1998. Springer-Verlag.

[LHJ95] Sheng Liang, Paul Hudak, and Mark Jones. Monad transformers and modular interpreters. In *Conference Record of POPL '94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California*, pages 333–343. ACM, January 1995.

[OAC$^+$05] Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. The Scala language specification version 1.0. `http://scala.epfl.ch/docu/files/ScalaReference.pdf`, October 2005.

[Ode05] Martin Odersky. Scala by example. `http://scala.epfl.ch/docu/files/ScalaByExample.pdf`, October 2005.

[Rey78] John C. Reynolds. User defined types and procedural data structures as complementary approaches to data abstraction. In David Gries, editor, *Programming Methodology, A Collection of Articles by IFIP WG2.3*, pages 309–317. Springer-Verlag, New York, NY, 1978. Reprinted from S. A. Schuman (ed.), *New Directions in Algorithmic Languages 1975*, Inst. de Recherche d'Informatique et d'Automatique, Rocquencourt, 1975, pages 157-168.

[Ste94] Guy L. Steele, Jr. Building interpreters by composing monads. In *Conference Record of POPL '94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Portland, Oregon*, pages 472–492. ACM, January 1994.

[Wad92] Philip Wadler. The essence of functional programming. In *Conference Record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 1–14. ACM, January 1992.

[Wad98] Philip Wadler. The expression problem. Email to the Java Genericity mailing list, December 1998.

[ZO05] Matthias Zenger and Martin Odersky. Independently extensible solutions to the expression problem. In *The 12th International Workshop on Foundations of Object-Oriented Languages (FOOL 12)*, Long Beach, California, 2005. ACM.