CSC 337, Fall 2013
Assignment 1
Due: Tuesday, September 17 at 22:00:00

The problems on this assignment have been assigned a total of 300 points to avoid the need for fractional points when grading. Those 300 points correspond to 50 points of the semester-long total of 600 assignment points. In other words, this assignment represents 5% of your final grade in this course.

Remember that late assignments are not accepted and that there are no late days but if circumstances beyond your control interfere with your work on this assignment, there may be grounds for an extension. See the syllabus for details.

About Google...

Things like Google and Stack Overflow are incredible resources for working professionals; but when used to directly search for the answer to a question on a class assignment, they can cause learning and/or creative opportunties to be forever lost. For example, one of the questions below asks, "What's a handy use for the `meta` element that's not mentioned on the slides?" My hope is that you'll take a few minutes and read the documentation for the meta element that's on MDN, W3C, or elsewhere, like W3Schools, and see if you can find something besides the `charset` attribute that looks like it could be put to good use. Once you've got an answer in mind, then maybe that's a good time to Google, to see if there are interesting uses that weren't evident from the documentation.

One of the problems below, `pattern`, essentially asks you to figure out a way to draw some carefully positioned triangles using only HTML elements, no CSS or JavaScript. I'd be a little surprised if you could Google up a solution but who knows. But if you do Google it up, you'll have lost a chance to do some out-of-the-box thinking. Once you've solved it, however, then see if you can Google up a solution. I did find a similar idea, albeit using CSS, that was very interesting.

If you'll trust me and take on these challenges as I intend, I think you'll learn a lot, and improve your ability to get and/or keep a great job.

But above all, don't let problems like `pattern` drive you crazy. My intention is to create a challenge for you; but if the challenge starts to turn into frustration, then it's time to ask for help. The TAs and I will explore what you understand and try to give you a tiny push to get you moving. If time's short or a tiny push doesn't work, we'll give you a bigger push. If a gap in your understanding is the issue, we'll help you fill it. Remember that our goal is for everybody to master the material and earn an "A".

Sample files and such

In this write-up the notation `$a1/x.txt` is used to indicate that a file can be found both at the URL `http://cs.arizona.edu/classes/cs337/fall13/a1/x.txt` and on the CS machines using the path `/cs/www/classes/cs337/fall13/a1/x.txt`.

A zip of the entire `a1` directory can be found at `$a1/all.zip`.

This convention will be used on all assignments.

Suggestion: Set up an "`a1`" search engine as shown on slide 28. Use this as the target: http://cs.arizona.edu/classes/cs337/fall13/a1/%s To download the zip just type (on Windows) Control-L a1 all.zip ENTER. On Macs, Cmd-L a1 questions.html will bring up the questions for Problem 7.

Programming problems

This assignment has some programming problems. The course prerequisites only require that you know how to program in some language but not in any particular language, like Java. Therefore, on this assignment you can use any language you want, as long as we've got a way to run it on lectura. Java, Python (2 or 3), Ruby (1.8.7), C++ (via g++) and Perl (5.14.2) are fine, for example. If you already know PHP, you can use it. C is fine, too, but writing these programs in C will require much more effort than I intend. **If C is your only option, let me know ASAP. If your only good option is a language that's not on lectura, like C#, let me know ASAP.** Based on the assignment 0 responses it looks like most of you will be using Java or Python.

The `cities` and `linkwords` programs both read from standard input. That's `System.in` in Java, `sys.stdin` in Python. `cities` creates two or more files. `linkwords` writes to standard output. To provide an example of doing those three things, here are Java and Python versions of a program called `numlines`. It reads lines on standard input and writes the lines, with line numbers prepended, to `out.txt`, a file it creates. It also prints a line on standard output using `System.out.println`.

Here's the Java version:

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class numlines {
   public static void main(String[ ] args) throws IOException {
      System.out.println("Running numlines in Java..."); // goes to standard output

      File output = new File("out.txt");

      Scanner scanner = new Scanner(System.in);
      FileWriter writer = new FileWriter(output);

      writer.write("Generated by the Java version of numlines\n");
      int lineNumber = 1;
      while (scanner.hasNextLine()) {
         writer.write(String.format("%d\t%s%n", lineNumber++, scanner.nextLine()));
                                          // %n is platform independent newline
      }

      writer.close();
      scanner.close();
   }
}
```

On Windows, use the `javac` command to compile it, producing `numlines.class`:

```
W:\337\a1>javac numlines.java
```

`tobe.txt` is a three-line input file that's a test case for `linkwords`. Let's look at it using the `type` command:

```
W:\337\a1>type tobe.txt
To be
or not
to be.
```

Let's run `numlines` and use a redirection operator (<) to indicate that we want the contents of `tobe.txt` to be available for reading from standard input:

```
W:\337\a1>java numlines < tobe.txt
Running numlines in Java...  (produced by System.out.println)
```

Let's look at `out.txt` to see the numbered lines:

```
W:\337\a1>type out.txt
Generated by the Java version of numlines
1       To be
2       or not
3       to be.
```

Let's run it again and use a redirection operator to capture standard output in a file named `x`:

```
W:\337\a1>java numlines < tobe.txt >x
```

Because output generated by `System.out.println` goes to standard output, the file `x` contains the data written by `System.out.println`:

```
W:\337\a1>type x
Running numlines in Java...
```

The same sequence of commands would work on UNIX except that you'd use `cat` or `more` instead of `type` to examine `out.txt` and `x`.

Here's the Python version, `numlines.py`:

```
import sys

print("Running numlines in Python...");

output = open("out.txt", "w")

output.write("Generated by the Python version of numlines\n");

lineNumber = 1
for line in sys.stdin:
   output.write("{}\t{}".format(lineNumber, line))
   lineNumber += 1

output.close()
```

Usage:

```
W:\337\a1>python numlines.py < tobe.txt
Running numlines in Python...

W:\337\a1>type out.txt
Generated by the Python version of numlines
1       To be
2       or not
3       to be.

W:\337\a1>python numlines.py < tobe.txt >x

W:\337\a1>type x
Running numlines in Python...
```

The files used in the examples above are all in `$a1/all.zip`.

Problem 1. (25 points) `pattern`

Create an HTML file `pattern.html` that produces this pattern: (in `$a1/pattern.png`)

Your solution need not be a pixel for pixel match but the triangles should be about twice as wide as they are high, all five pairs should be centered on a vertical axis to within a few pixels, and there should be a small amount of vertical space between them.

You may use any HTML elements you wish but `$a1/blackpixel.png` is the only media file you may use. You may not use the `canvas` element or create an SVG-based solution. And, of course, no CSS or JavaScript yet! Hint: think about letting your computer do the work of creating `pattern.html`, like I did.

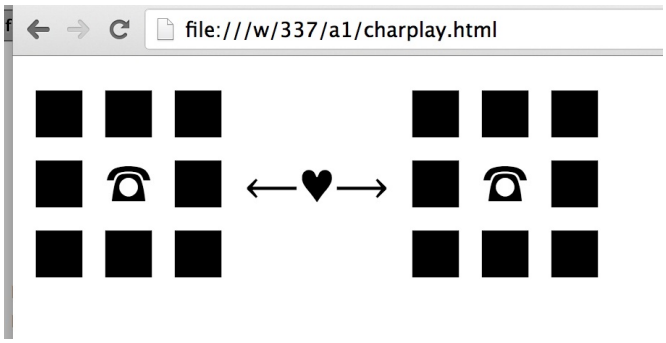
I've found that my solution doesn't look very good in Chrome but looks fine in Firefox.

If you can produce the pattern but want to do something more interesting just for fun using the same idea,

do it too! Put both in `pattern.html`, one after the other. Sorry—no extra credit for this; just a chance to impress.

Problem 2. (15 points) `charplay.html`

The idea of this problem is to play around with character references, which are mentioned on slide 23. http://www.w3.org/TR/html5/syntax.html#named-character-references has the full set. The catch: the glyphs vary between browsers. I came up with this using Safari on my Mac: (in $a1/charplay.png)



The above is shown with several steps of zoom using View>Zoom In.

Due to cross-machine differences, instead of saying "Match it!" I'd like you to simply play around with some character references and see if you can come up with a creation you like. If you've got a Mac, I challenge you to match mine but feel free to do whatever you'd like.

You'll get full credit on this problem if you simply submit a `charplay.html` that represents at least 30 minutes of work and includes (1) a comment like <!-- I worked on this for at least 30 minutes --> and (2) a comment that tells where you found it looks best, like <!-- best on Win 8 and latest Opera -->. If you can come up with something great in less in than 30 minutes, that's fine, too.

If you already know CSS and want to use it to spruce up your creation for fun, go ahead!

Problem 3. (55 points) `linkwords`

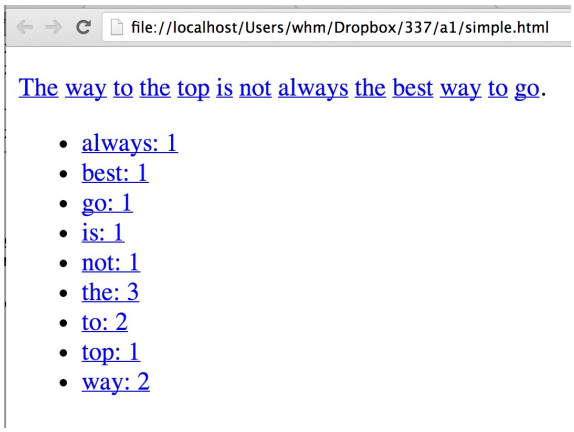
For this problem you are to write a program that reads plain text on standard input and produces an HTML file where each occurrence of a word is a hyperlink to the next occurrence of that word.

A Java version of `linkwords` might be run like this on Windows:

```
W:\337\a1>cat simple.txt
The way to the top is not always the best way to go.

W:\337\a1>java linkwords < simple.txt >simple.html
```

Here's a shot from the browser:



The contents of `simple.txt` are shown at the top, followed by a list of words and occurrence counts. Case is preserved in the text but the counts are case insensitive, with words shown in lowercase. The text and the list are both children of paragraph elements. Assume the input won't contain characters such as "<" that need to be escaped. Words are considered to be a sequence of consecutive letters; don't worry about contractions or hyphenated words.

It's important to test your solution with Chrome, Internet Explorer, or Chromium (on Linux). In those browsers when you click on a link a box is shown around the target of the link. In the above when you click on "The" you'll get a box around the second "the". Clicking on the last occurrence of a word takes you to the entry for that word in the list. Clicking on the list entry wraps you around to the first occurrence of that word.

Here's a video demo: http://www.youtube.com/watch?v=Dp5GcyZY2dE

Suggestion: Work out a small prototype HTML file manually before you start programming.

Let us know ASAP if you can't find a browser that shows the box around targets like in the video.

$a1/tamee.{txt,png} provides another example, albeit static.

An element of this problem is breaking up input lines into words and non-words (which should not be links). I encourage you to work it out yourself; but I don't want that to soak up a lot of time, so you can find some examples of splits using regular expressions for Java, Python and PHP in `$a1/regexpex.html`.

To build up the table of word counts, you'll probably want to use a HashMap<String,Integer> in Java, a dictionary in Python, or an array in PHP.

Problem 4. (60 points) `cities`

For this problem you are to write a program that reads a data file and generates a collection of HTML files for a simple, static website on Great Cities of the World.

The input file is read from standard input. The information for each city is specified as a series of six lines:

Name
Nickname

Official website (minus `http://`)
Images
Events
A line containing just a period

The images line consists of one or more entries separated by slashes, like this:

```
A:p1.jpg/B:i1.jpg/A picture:file 1.jpg
```

The line above represents three images. The title of the first image is "`A`". It can be found in the file named "`p1.jpg`". Note that a colon separates the title and the file name, and both can contain blanks. No provision is made for titles or file names that contain a slash or colon—they simply can't be represented. Process this line by first splitting on slashes and then split each of the parts on colons. (Use `String.split` in Java and `str.split` in Python.) There is no provision for specifying a path to a file, like `x/y.jpg`.

The events line consists of one or more entries separated by slashes, like this:

```
Spring Festival/Fourth of July Parade
```

No provision is made for events with a slash in their name.

`$a1/cities.txt` is a sample input file. A Java version would be run like this:

```
W:\337\a1>java cities < cities.txt
```

No visible output is produced but driven by the content of `$a1/cities.txt`, four files are created:

```
cities.html
Paris.html
Walkertown.html
Kernersville.html
```

Take a look at `cities.txt` and the four files above that were produced from it. You'll see that `cities.html` is mostly links to the per-city HTML files. The per-city HTML files display images and text about the city. The images are in `$a1/images`.

You needn't match what my version generated; but you must meet these requirements: (1) all the input needs to be presented in some way, (2) you must generate multiple pages, (3) per-city pages must be reachable from a top-level `cities.html`, and (4) every page must have a "home" image that links back to `cities.html`. If you already know CSS and want to add some style, go ahead, but that won't have any bearing on your grade. If you come up with something nice, maybe we'll have everybody else match it in assignment 2!

If you'd like to add to the collection of cities and images, include in your D2L submission a file named `mycities.txt` and add your images to the `images` directory. Submissions might be shared with the class, and/or used on later assignments, so don't supply anything you don't want the world to see.

Problem 5. (85 points) `markup.html`

The files `markup.png` and `markup.pdf` show rendering of my copy of `markup.html`. Your task on this problem is to figure out and add the HTML markup required to produce exactly the same rendering. You'll see that `markup.pdf` is a little sharper but, for example, it doesn't show the yellow

marks and those are important.

Pay close attention to the text—some is **bold**, some is *italics*, some is underlined, some is ***all three***, and more. Some text is in a fixed width font. It might be good to start by printing `markup.png` or `markup.pdf` and use a red pen to circle where work is needed.

`$a1/markup.txt` has the text, copied from the browser. `$a1/redcircle.png` is the PNG for the circle.

I'm aware of one cross-browser difference at present: the bullet styles on that descending procession of numbers at the top varies. For example, in Opera I see solid round bullets at all levels. To minimize headaches use Chrome when checking against my samples.

Near the bottom of the text is a list of five rules that you must heed. The last is the most important: if your `markup.html` fails validation, you'll get a zero on the problem. I recommend that you periodically validate as you're creating the file but don't worry about the prohibition of apostrophes, double quotes, and the character X until you're otherwise done.

Problem 6. (30 points) `deepest.html`

Create a file `deepest.html` that creates the deepest possible nesting of the following elements: `a`, `article`, `blockquote`, `body`, `br`, `em`, `h1`, `head`, `header`, `footer`, `html`, `li`, `meta`, `p`, `pre`, `strong`, `ul`. You may use each element at most once. Your file must validate. **Partial credit will be awarded if you don't reach the maximum, but failure to validate will earn a zero.**

The breadcrumbs at the bottom of the Chrome Developer view can be used to check your depth:

The above shows a depth of four: html>body>p>strong.

Problem 7. (30 points) `questions.html`

`$a1/questions.html` is an HTML file with a list of free-response questions. Copy the file and answer the questions. Put your response in the `<div class=answer>` element that's nested inside the `<div class=question>` element for each question.

Be sure that the file still renders properly after your edits.

Turning in your work

Use the D2L Dropbox named `a1` to submit a zip file that contains all your work. Use any name you wish for the file, perhaps `a1.zip`. If you submit more than one zip we'll grade your final submission.

The `python` command on `lectura` runs Python 2. If you use Python 3, add a comment "`# Python 3`" somewhere in your `.py` files.

Here's the full list of deliverables:

```
cities.* (e.g, cities.java)
charplay.html
deepest.html
linkwords.*
markup.html
pattern.html
questions.html
```

It's fine if your zip includes other files, too.

Remember that to be worth any points at all your `deepest.html` and `markup.html` must pass at `http://validator.w3.org/#validate_by_input`. Validation is NOT required for the other HTML deliverables, including the HTML files generated by `cities` and `linkwords`.

If you want to contribute to the Cities of the World compilation, include `mycities.txt` and whatever images are referenced. Don't worry about the potential of duplicated city or image names.

This assignment does require you to explore the HTML documentation on the web to find some useful elements we haven't discussed in class. Other than that, if you find yourself wanting to use something we haven't yet covered, you're probably overlooking something and/or making the problem more complicated than I intend.

Error handling and implicit limits

In the workplace you'll find that some programs need a lot of error handling and some don't need any at all. In this class we want to focus on the course material and not get consumed by writing and meeting detailed specifications covering errors and various odd cases. Unless otherwise specified in the write-up for a problem, your code in this class need only work on the "Happy path", where all input is well-formed, no errors occur, and all the things you should handle are described in the write-up or are exemplified in test files.

For example, consider the images line in `cities`. Here are some things you don't need to worry about:

An image entry that doesn't have a colon
Two consecutive slashes or a trailing slash
An empty title or file name
An image file that doesn't exist
Leading or trailing whitespace
An empty line

Thinking more broadly about `cities`, here some other things you don't need to worry about:

A duplicate city name
A city's entry having too many or two few lines
A missing period at the end of the block for a city
Creation of an `.html` file failing

If any of those things occur, then *behavior is undefined*. That phrase, "behavior is undefined" is something you'll see throughout your career. It means that the implementor does not need to be concerned in any way with that case. The program might terminate with an error, go into an infinite loop, or work just fine. Of course, it should never be viewed as a license to do something malicious.

Unless otherwise stated no specific limits on quantities should be assumed. Consider this line from the `cities` write-up: "The images line consists of one or more entries...". That means you can assume you'll have at least one image but there might be a million images, or a billion. If no upper limit is specified, it should be assumed that the number is only limited by the resources the operating system can provide.

Nothing is said in the cities write-up about how long an image's title or filename can be. A thousand-character title should work just fine. A thousand-character file name might be too long for some operating systems and cause file creation to fail but in this class you don't need to write code to workaround an operating system limit like that. It might save you a puzzle at some point to add code to check for file creation failure; but unless a write-up says to handle that case, you can assume we won't test that case.

Feel free to post questions about behavior on Piazza. If I say "the behavior in undefined in that case", you can read that as "it's ok if that blows up". If I say "no limit should be assumed", you can read that as "whatever the operating system limits it to".

Miscellaneous

My estimate is that it will take a typical student between five and fifteen hours to complete this assignment. If you're a CS senior, you'll probably do it in less. If you've taken only a single programming class, you might be on the high end, or beyond.

For each assignment I'll put up a Piazza poll that asks you to report how many hours you spent on the assignment. You don't need to participate in that but if you want to, make a little effort to track your time.

Keep in mind the point value of each problem; don't invest an inordinate amount of time in a problem or become incredibly frustrated before you ask for a hint or help. Remember that the purpose of the assignments is to build understanding of the course material by applying it to solve problems. If you reach the ten-hour mark, regardless of whether you have specific questions, it's probably time to touch base with us. Give us a chance to speed you up! Our goal is that everybody gets 100% on this assignment, and gets it done in whatever is a reasonable time for them.

Feel free to use comments to document your code as you see fit, but note that no comments are required, and no points will be awarded for documentation itself. (In other words, no part of your score will be based on documentation.) Exception: Those comments mentioned for `charplay.html`.

Remember that I award Bug Bounty points; if you find problems, there's a potential reward for reporting them. I prefer that bugs be reported by mail to me. I'll put up an "Assignment 1 FAQs and Corrections" post on Piazza and update it as things come in.

Use the `a1` folder for any Piazza posts about the assignment.

Note that this assignment has an inordinately long period before it's due. That's because I'll be doing a couple of substitute lectures in 453 the week of Sep 9. Were it not for that I'd make this due on Sep 12; but I'm afraid that with the extra work for 453 I might not be able to be as responsive as I should when answering questions and such. (Translation: this assignment may have the lowest work:time ratio you'll see all semester.)