Project 2b

In this project, you have to build the graph generation software in Smalltalk.

Setting up

To start:

```
svn export http://unbox.org/wisp/var/timm/09/310/lib/proj2 proj2b
```

If that works, then following code should work:

```
cd proj2b
make score
10 FAILED
 4 PASSED
```

How you will be assessed

1. For ten marks, can you write the Smalltalk code to generate the eg/*.want files from the eg/*.dat files?
2. For five marks, can you wrote code to address the issues mentioned "XXX" in the files:

```
     1  1classes.st:4:      "XXX does Graph need anymore attributes?"
     2  1classes.st:9:      "XXX define subclasses of Tag to handle every tag"
     3  1classes.st:11:     "XXX Note that 'Tags' should run by calling 'tag'
     4  GraphReader.st:1:   "XXX find an implement any empty methods in the following"
     5  GraphReader.st:32:  "XXX delete all commentChars and anything to their right.
     6  GraphReader.st:33:   XXX After that, skip any blank or empty lines"
     7  Graph.st:1:         "XXX find an implement any empty methods in the following; e.g.
     8  Graph.st:5:         "XXX as per assignment 2a, there are bugs in Graph that you have tc
     9  Range.st:4:         "XXX"
    10  Range.st:9:         "XXX"
    11  Range.st:13:        "XXX"
    12  Range.st:17:        "XXX"
    13  Range.st:21:        "XXX"
    14  Range.st:25:        "XXX"
    15  Range.st:29:        "XXX" "should return a number 0 <= n <= 1"
    16  Tag.st:1:           "XXX find an implement any empty methods in the following"
    17  Tag.st:7:           "XXX if not class handles tag, then instantiate  TagError
    18  Tag.st:11:          ^nil "XXX once TagError is running, get rid of this return" !
    19  Tag.st:23:          "XXX need to complain when the first item is not an integer"
```

What to hand in

1. As before, all the code in your sub-directories.
2. A file proj2/XXX.st with 19 numbered sections (corresponding to the above list) showing what you did or could not do to handle the above "XXX" points. Note: in this document you must be very specific. We won't understand what you have unless you explain it to use vveeerryyy sslllowwwwllyyy with lots of code examples dropped in.

What you Will Have to Do

Write the actual graph layout code inside Graph.

Write subclasses of "Tag" to handle all the tags you see on input. For example, here's TagWidth class that broadcasts it handles 'width' and processes a set of 'words' representing the line:

```
! TagWidth class methods !
```

```
handles
        ^'width'
!!
! TagWidth methods !
line: words for: this at: line
        words first asInteger oo.
        words size > 1
                ifTrue: [^self error: (line,')  expected 1 argument ')].
        this width: words first asInteger.
        "XXX need to complain when the first item is not an integer"
! !
```

Find some fun way to handle the lines without tags (the numerical points). Hint: Smalltalk supports regular expressions, see http://smalltalk.gnu.org/wiki/regular-expressions

About the New Code

This code base is not quite the same as before. Read the following notes, carefully.

What has not changed

- You still create tests by adding some file *eg/testname.want*.
- You still run test code using

  ```
  make X=something run
  ```

- Once you like the output, you can cache it using

  ```
  make X=something cache
  ```

- You still run text code and compare its output to a .want file using

  ```
  make X=something test
  ```

- You still run all tests using

  ```
  make tests
  ```

- You still generate a final report of all the tests using

  ```
  make score
  ```

- You still submit by changing the variable names in Makefile then typing

  ```
  make submit
  ```

What has changed

1. Before: the magic file *proj2/st* looked like this:

   ```
   gst lib.st classes.st methods.st go.st $*
   ```

   Now, it looks like this:

   ```
   gst `ls *.st`  $*
   ```

   which means (1) find all the ".st" files in the current directory and (2) sort them alphabetically and (3) load them in that order. So, before, my Smalltalk tricks were in lib.st but now they are in 0lib.st (so they get loaded first). And the class definitions are in 1classes.st (so they get loaded second).

2. Before: you ran tests by their number; e.g. "make X=3 test".

   Now: the tests are a file name (one of the ".st" files in eg); e.g."make X=0lib.st test" And what is this "0lib.st" thing?

Welll...

3. Before: the *eg* directory contained dead data files.

   Now: it contains Smalltalk code (the *.st) files:

   ```
   eg/0lib.st  eg/10.st  eg/1.st  eg/1tag.st  eg/2.st  eg/3.st  eg/4.st
   eg/5.st  eg/6.st  eg/7.st  eg/8.st  eg/9.st  eg/GraphReader.st  eg/Range.st
   ```

   and some *.dat files:

   ```
   eg/10.dat  eg/1.dat  eg/2.dat  eg/3.dat  eg/4.dat  eg/5.dat
   eg/6.dat  eg/7.dat  eg/8.dat  eg/9.dat  eg/GraphReader.dat
   ```

   You'll note that there is a similarity in the names. For example, 10.st uses the graph specification in 10.dat But right now, the .st files are just stupid. All they do is write the input graph spec to the output (you'll have to fix that).

Hints

Use my "oo" method to show data structures.

To understand my code, have a good long look at:

- "eg/0lib.st" and "eg/0lib.st.want"
- "eg/GraphReader.st" and "eg/GraphReader.st.want"

And if you don't understand what is going on... come and ask me! I'm in my office 12-1, Mon-Wed-Fri, waiting for you. I sit myself, sad, alone, in the class consultation times and no one comes to visit me. Why? Why? Is it my poor hygiene? My horrid taste in music? My obsessive need to discuss esoteric programming language details with everyone I see? Oh if only there was someway I could work with a large group of people who have some need, some desire, to learn programming language details. If only...