NE 155
2D Transport Final Project Suggestions

Write a 2D transport solver that has vacuum boundaries on the bottom and left faces and reflecting boundaries on the top and right boundaries. Adapt the advice below as applicable based on whether you use a compiled language like C++ or an interpreted language like Python.

Good programming practices (There are some tutorials for all of these things available through Berekeley's The Hacker Within.):

- Please version control your code on something like github.
- Consider writing code comments with something like Doxygen (http://www.stack.nl/ dimitri/doxygen/) to document the API (application programming interface).
- Consider writing tests for your code using something like gtest (https://code.google.com/p/googletest/) - this might be a good extension for a two person project in particular.

Components that you'll need to include with the code:

- A README that states
  - How to compile the code (if applicable)
  - How to execute the code
  - Status of the code (as applicable): operational/compiles but doesn't run/doesn't compile/known bugs
  - Describe the problem solved by your code, expected input, resulting output, and any limitations or restrictions
- A sample input file and corresponding output file produced by your code.
- Things you'll want to include in the input file:
  - Number of x and y cells
  - Cells size in each dimension (you can choose if you want it to be uniform or non-uniform)
  - number of angles per octant
  - angular quadrature values, μ_i, η_i, w_i
  - Number of materials
  - A way to assign materials
  - Physical constants for each material: Σ_t, Σ_s

  - Fixed source term. You need to decide things like whether you want a source everywhere only in some cells, whether it must be constant or can be a function of space, etc. and how to express those choices in the input file.
  - If you want to allow for different boundary condition choices, how to specify those.
  - Flux convergence tolerance.

The general framework for your code should be:

- *Main Module:* calls subroutines listed in specific sequence, prints execution time, and terminates execution
- *Version data subroutine:* write code name, version number, author name(s), date and time of execution to an output file (perhaps also print to screen).
- *Input data subroutine:* read and/or process the input data. Check all input values for correctness/sensibility, e.g., all values are positive, array dimensions are correct, etc. Print an error message and terminate if one or more errors occur, otherwise print notification of successful input checking.
- *Input echo subroutine:* print the input data for each cell to the output file (this is good for reproducibility). Please format this in some useful way.
- *Transport solver subroutine:* implement the discretized transport solver here. While building the surrounding structure you can just have this print "will solve TE here". This will be comprised of several other subroutines (this is a suggested but not required implementation, but is a decent strategy):
  - *Inner iteration subroutine:* this will compute the source term (right hand side of fixed source and scattering), call the sweep subroutine, and check for convergence.
  - *Sweep subroutine:* "sweep" the mesh along each angle in the octant (each plus and minus combination for each angle set). This is a directional marching through cells. In each cell call the spatial solver method.
  - *Spatial solver subroutine:* solves the flux values in each cell. The diamond difference method is likely a good choice here.

I have reference solutions for a few cases if you are interested.