**PHY/CAM 381C: Computational Physics Homework 1 (Revised)**
**Due: Thursday, January 30, 12:30pm**

**Problem 1a:** Send me (`matt@infeld.ph.utexas.edu`) an e-mail message from *your own account* on any of the Linux machines in RLM 3.118. Use the system's "hostname" (`linux1, linux2, ...` etc.) for the subject of the message, and include *your* name in the body of the message. Incidentally, note that there is a Unix command `hostname`.

**Problem 1b:** Send me an e-mail message from *your own account* on one of the recommended Center for Relativity SGI machines (`einstein` or `godel`). Again, use the host name for the subject of the message, and include your name in the text of the message.

---

*Do the remaining problems using your account on the SGI machines. In particular, note that "∼" in the following refers to your home directory on those machines, and all file and directory references are to files on the SGIs. As always, send me mail immediately if you encounter problems using any of the machines, or if you have undue difficulties with any part of the assignment. Be sure to read all instructions carefully as your grade will depend on certain files being in the right places with the right names.*

---

**Problem 2:** I have created directories for each of you which you may use to use to "publish" Web pages (preferably related to this course) via the Relativity web server (`http://wwwrel.ph.utexas.edu`). Your personal directory (which I'll refer to as your Web directory) is `/Public/Students/<accountname>` where `<accountname>` is the name of your account on the SGIs. I have created a "template" homepage named `/Public/Students/phy381C/welcome.html`. Copy this HTML file into your Web directory (use the same name for the file—`welcome.html`) and modify the file to reflect your name, address, phone number etc. Use either `vi` or `emacs` to do the modifications. If you don't feel like publishing any specific piece of information, specify it as "unlisted". Below the horizontal rule (line) in the template file, add suitably labelled links to (a) the course home page (b) my home page and, if applicable, (c) your research group's page. Feel free to add links, now and throughout the course, to resources which you find relevant and/or interesting. Check your work by verifying that you can get to your home page by going to our Course page, selecting "Student Pages", and then your name.

---

**Problem 3:** Answer the following questions in a file called `~/hw1/a3/README`. Note that for this and following problems you will first have to create a subdirectory in `~/hw1`. If you have difficulties with any of the exercises here, first review the on-line Unix notes and the `man` pages for the commands given in the hints. In the `README` file, make sure that you clearly indicate where one answer ends and the other begins.

**Problem 3a:** Document and explain the output from the following sequence of commands which is to be executed within one of your own shells:

```
% set command='date'
% echo $command

% echo '$command'
% echo "$command"
% echo `$command`
```

Note that the quotes used in the last three commands are forward quotes, double quotes, and back-quotes, respectively.

**Problem 3b:** Give a Unix command or pipeline which will compute the total number of lines in all the `.f` files (files with an extension `.f`) contained in the file sub-hierarchy rooted at the current working directory. Test your construct on the sub-hierarchy rooted at `~phy329/f77`. Your command need not report *only* the total number of lines, but be sure to clearly state how the total is to be read off from the generic output. *Hint: You may find the* `find` *command useful.*

**Problem 3c:** Modify your previous command or pipeline so that it counts the total number of lines in all of the `.f` files (again in the hierarchy rooted in the current working directory) which have a `'c'` or `'C'` in column 1 (comment lines). Again, test your command on the hierarchy rooted at ~`phy329/f77`. Roughly what percentage of the lines of Fortran code contained in the hierarchy are comments? *Hint: Consider the* `find` *and* `grep` *commands.*

**Problem 3d:** The file `/usr/lib/dict/words` (hereafter called the dictionary) contains a list of words (mostly genuine English words), one per line. How many words does the dictionary contain? How many 3-character words does the dictionary contain? List all of the 17-letter words in the dictionary. How many words in the dictionary contain all five of the vowels 'a', 'e', 'i', 'o' and 'u'? *Hint: Use* `grep`.

---

**Problem 4:** Typeset the following passage from *Classical Electrodynamics*, by J. D. Jackson (2nd Edition, Copyright John Wiley & Sons), in LaTeX. Leave the LaTeX source (it should be a complete, short document) in ~`/hw1/a4/the.tex`. You need not use the same size of type in your document as that used here, but otherwise try to make your LaTeX'ed output resemble mine as closely as possible.

### 1.1 Maxwell Equations in Vacuum, Fields, and Sources

The equations governing electromagnetic phenomena are the Maxwell equations, which for sources in vacuum are

$$\begin{array}{rcl} \nabla \cdot \mathbf{E} & = & 4\pi\rho \\ \nabla \times \mathbf{B} - \dfrac{1}{c}\dfrac{\partial \mathbf{E}}{\partial t} & = & \dfrac{4\pi}{c}\mathbf{J} \\ \nabla \times \mathbf{E} + \dfrac{1}{c}\dfrac{\partial \mathbf{B}}{\partial t} & = & 0 \\ \nabla \cdot \mathbf{B} & = & 0 \end{array}$$

Implicit in the Maxwell equations is the continuity equation for charge density and current density,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \mathbf{J} = 0$$

---

**Problem 5:** As discussed in class, the sub-hierarchy rooted at ~`/ph329/f77` contains a variety of simple Fortran programs which illustrate various language features and programming techniques. First copy this entire hierarchy into ~`/hw1/a5` (thus creating a hierarchy rooted at ~`/hw1/a5/f77`) and remove subdirectory `f77/ex1` and its contents. Then, for each of the subdirectories `f77/ex2`, `f77/ex3`, $\cdots$, `f77/ex7`, build all of the demo programs contained therein (*Hint: You should be able to do this by simply typing* `make` *in each of the sub-directories*), and run them. You will have to figure out what command-line arguments and/or input must be supplied to each program by examining the source code. Ignore the warnings about multiply defined symbols when compiling the programs in `f77/ex3`. In each of the subdirectories create a file called `README` which gives a *brief* synopsis (no more than two or three sentences) of what each program does. Create a subdirectory called `f77/ex2a` which contains a demo program called `myfdemo1` which is identical to `f77/ex2/fdemo1`, except that after each feature is demonstrated it writes the prompt:

```
myfdemo1: Through <some string corresponding to specific feature>
myfdemo1: Enter anything & <CR> to continue
```

instead of

```
Through <some string corresponding to specific feature>
Enter anything & <CR> to continue
```

**Problem 6:** In directory ~/hw1/a6 create a Fortran 77 source file rw2d.f and corresponding executable rw2d, which simulates and analyzes random walks of a particle on a two dimensional integer lattice. Specifically, if at step $n$ the current position of the particle performing the walk is $(x, y) = (i, j)$, where $i$ and $j$ are integers, then, with equal probability (i.e. $p = 1/4$), the position at step $n + 1$ is either $(i + 1, j)$, $(i - 1, j)$, $(i, j + 1)$ or $(i, j - 1)$. rw2d should accept two integer arguments as illustrated in this usage message:

```
usage: rw2d <nsteps> <nwalks>
```

where <nsteps> is the number of steps to take per walk, and <nwalks> is the number of separate walks to be generated. Note that you can use the integer function i4arg discussed in class and available in the p329f library to parse the command-line arguments. For each walk, start the particle at $(0, 0)$ and use real*8 values for the particle coordinates, even though the coordinates are restricted to integer values. When your program is finished simulating the random walks, it must write on standard output the average of the distance-squared of the particle from its starting point ($< r^2 >$; use a simple average, and average over all walks) after each step, as a function of step number. Specifically, the output should consist of two numbers per line, generated using code such as the following:

```
do i = 1 , nsteps
   write(*,*)  i,   rsqavg(i)
end do
```

Document and test your program thoroughly, then make runs of 10, 100, and 1000, 5000-step walks, and save the output in files out10, out100 and out1000 respectively. Use gnuplot or sm (Supermongo) to produce a *single* plot of average-distance-squared versus step number for all three data-sets. Save a postscript version of the plot in distance.ps. What can you say about the behaviour of $< r^2 >$ versus step number as the number of walks gets large? Answer in a file called README. Note than on the SGIs, the built-in real*8 function rand can be used to generate a random number between 0 and 1—it is up to you to figure out how to use rand to generate the particular randomness required by the simulation. Be sure that you declare the function properly in the main program, and note that since it has no arguments, it should be invoked with an empty argument list:

```
real*8      rand

write(*,*) 'This is a pseudo-random number ', rand()
```

**Problem 7:** *The Chaos Game*
Let

$$V_i \equiv (\tilde{x}_i, \tilde{y}_i) \qquad i = 1, 2, 3$$

be the vertices of an equilateral triangle with unit side length, lying in the $xy$ plane. The orientation and position of the triangle in the plane is not important but, for concreteness, you may wish to "center" the triangle at the origin $(x, y) = (0, 0)$. The game is initialized by choosing, *at random*, some point in the $xy$ plane, such that the point lies outside of the triangle. Call this randomly chosen location the *active point*. One then *randomly* chooses one of the three triangle vertices, $V_i$, draws an imaginary line between the vertex and the active point, bisects the line and makes the point of bisection the active point. This last sequence of randomly choosing a vertex, bisecting the line between it and the active point and updating the active point with the point of bisection is then repeated some specified number of times. If one plots all the active points which are generated, an interesting pattern appears after a sufficiently large number of iterations.

In directory ~/hw1/a7, write a well-documented Fortran program called chaos2d which implements this algorithm. As with the previous problem, you will probably find use for the real*8 random number function, rand. The program should take a single integer command-line argument which specifies the number of iterations to perform. Again, I suggest that you use the i4arg function discussed in class and available in the p329f library to parse the argument. The program should write to standard output the $x$ and $y$

coordinates of all of the active points which are generated (including the original randomly chosen one). Specifically, use a `write` statement like the following

```
write(*,*) xactive, yactive
```

in your iteration loop, so that the coordinate pairs appear one per line on standard output. You may optionally output (also to standard output) the $(x, y)$ coordinates of the three triangle vertices $V_i$. If you choose to do so, output them *once only*, presumably at the beginning of the game. Finally, although the initial active point is to be chosen randomly, ensure that its distance from the center of the triangle does not exceed 2.5 (recall that the sides of the triangle are of length 1.0). Run your program for 10, 100, 1000 and 10000 iterations and redirect the output to files `out10, out100, out1000,` and `out10000` respectively. Use `gnuplot` or `sm` to plot the points defined by each of the output files and save the plots as postscript files `out10.ps, out100.ps, out1000.ps,` and `out10000.ps`. When you make your plots, you should use an aspect ratio of 1 (same range of $x$ as $y$) and should not be afraid to clip a *few* points in order to focus on the region of interest (i.e. the base triangle and its interior). Also be sure that the plotting symbols you use are not so large as to obscure the (interesting) detail in the plot: in fact, you should figure out a way to make the plotting symbols as small as possible (but still visible, of course).

**Optional**: Generalize the game in various ways such as

1. Adding an option which controls the number of fixed vertices used (i.e. so that you can play with 4, 5, 6, ... fixed vertices, rather than just 3).
2. Adding an option which controls the placement of new points: new points should continue to be placed along a line connecting the previous point and the randomly chosen vertex, but need not be at the mid-point of the line-segment.
3. Implementing the game in three dimensions.

If you do chose to generalize the game, be sure to create a new source file and executable—`chaos2d` must execute precisely as specified above.