PHY/CAM 381C: Computational Physics
Homework 1 Key

Problem 1:

The minimalist answer (assuming that `mail` is aliased to `Mail`, which it should be), is:

```
%  echo "Your name" | mail -s "`hostname`" \
        matt@infeld.ph.utexas.edu
```

which will work on either the SGIs or the linux machines. Note the use of the backslash here and below to denote the continuation of a command across more than one line.

Problem 2:

No key required. Feel free to continue to update your page as the course progresses; your accounts will remain active at least through the summer.

Problem 3a:

The command sequence

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

sets the value of the shell variable `command` to the *string* `date`; thus the `echo command` produces

```
date
```

Then,

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

produces the string

```
$command
```

since the forward quotes inhibit *all* evaluation by the shell;

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

produces

```
date
```

since shell variables and back-quote substitutions are evaluated within double quotes; and

```
% echo `$command`
```

produces (something like)

```
Mon Feb 10 10:25:34 CST 1997
```

since the back-quotes *first* cause evaluation of shell-variables within their scope, then cause the string they enclose to be executed as a Unix command, and finally cause the standard output from the command to be substituted for the back-quoted construction itself.

Problem 3b:

Assuming that the working directory is `~phy329/f77`, the command

```
% wc `find . -name '*.f' -print`
```

produces output

```
     8         12        101 ./ex1/greeting.f
     6         12        119 ./ex1/sayhello.f
               .
               .
               .
    62        185       2047 ./ex7/tdm.f
  2088       6808      68957 total
```

from which the total number of lines (2088) of Fortran source can be easily read off.

Problem 3c:

Again assuming that the working directory is `~phy329/f77`, the command

```
% find . -name '*.f' \
         -exec grep -i '^c' {} \; | wc -l
```

reports that a total of 861 lines, or about 41% of the lines are comments. Note the use of the `-l` option to the `wc` command to echo *only* the number of lines in the input.

Problem 3d:

There are 23788 words, 766 three-character words, 5 17-character words and 108 words which contain all 5 English vowels in `/usr/lib/dict/words`. Sample commands are:

```
% wc -l /usr/lib/dict/words

% grep '^...$' /usr/lib/dict/words | wc -l

% grep '^.................$' /usr/lib/dict/words

contradistinction
contradistinguish
electrocardiogram
indistinguishable
spectrophotometer

% grep -i a /usr/lib/dict/words | grep -i e | \
       grep -i i |  grep -i o | grep -i u | wc -l
```

Problem 4:

Here is one acceptable LaTeX source file:

```
\documentstyle [11pt] {article}

\begin{document}

\begin{flushleft}
{\bf 1.1 Maxwell Equations in
        Vacuum, Fields, and Sources}\\
\end{flushleft}

The equations governing electromagnetic
phenomena are the Maxwell equations, which
```

```
for sources in vacuum are
%
\begin{eqnarray*}
  \nabla \cdot {\bf E} &=& 4 \pi \rho \\
  \nabla \times {\bf B} -
    \frac{1}{c}
    \frac{\partial {\bf E}}{\partial t} &=&
    \frac{4\pi}{c}{\bf J} \\
  \nabla \times {\bf E} +
    \frac{1}{c}
    \frac{\partial {\bf B}}{\partial t} &=&
    0 \\
  \nabla \cdot {\bf B} &=& 0 \\
\end{eqnarray*}
%
Implicit in the Maxwell equations is the
continuity equation for charge density and
current density,
%
\[
 \frac{\partial \rho}{\partial t} +
       \nabla \cdot {\bf J} = 0
\]
%
\end{document}
```

Again, note that I am assuming that you will pick up sufficient LaTeX on your own to enable you to use the system to typeset your term project. Please see me if you are having undue difficulties doing so.

Problem 5:

No key required. Refer to your personal Grading sheets for any specific comments.

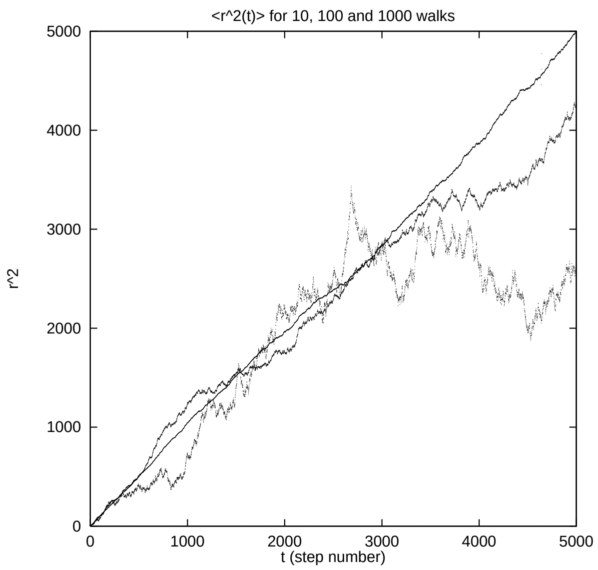
Problem 6:

Sample source code:

```
c-----------------------------------------------------------
c     PHY/CAM 381C Homework 1, Problem 6
c
c     rw2d: Simulates random walks on 2-d integer lattice.
c
c     Performs specified number of walks with specified
c     number of steps and outputs <r^2(step number)>
c     where the expectation value <...> is taken over
c     walks.
c
c     Walk restricted to horizontal and vertical steps.
c     Uses integer function 'roll' (based on built-in
c     'rand') to pick randomly among {1,2,3,4}.
c-----------------------------------------------------------
      program         rw2d

      implicit        none

      integer         iargc,       i4arg,      roll

      integer         nsteps,      nwalks

c-----------------------------------------------------------
c     Algorithm maintains running computation of (unnormal-
c     ized) expectation value of rsqavg.  The penalty for
c     this straightforward approach is the need to specify
c     the maximum number of steps in advance.
c-----------------------------------------------------------
      integer         maxsteps
      parameter     ( maxsteps = 100 000 )
      real*8          rsqavg(maxsteps)

      real*8          x,           y,          d
      integer         iwalk,       i,          randir
c-----------------------------------------------------------
c     For peace-of-mind on slow computers, if 'ltrace' is
c     .true., then progress message will be written to
c     standard-error every 50 (hard-wired below) steps.
c-----------------------------------------------------------
      logical         ltrace
      parameter     ( ltrace = .false. )

c-----------------------------------------------------------
c     Argument parsing and checking ...
c-----------------------------------------------------------
      if( iargc() .ne. 2 ) go to 900
      nsteps = i4arg(1,-1)
      nwalks = i4arg(2,-1)

      if( nsteps .eq. -1  .or. nwalks .eq. -1 ) go to 900
      if( nsteps .gt. maxsteps ) then
         write(0,*) 'rw2d:  Too many steps requested'
         stop
      end if

c-----------------------------------------------------------
c     Initialize <r^2(t)> ...
c-----------------------------------------------------------
      do i = 1 , nsteps
         rsqavg(i) = 0.0d0
      end do

c-----------------------------------------------------------
c     For each walk ...
c-----------------------------------------------------------
      do iwalk = 1 , nwalks
c-----------------------------------------------------------
c        Initialize the particle position ...
c-----------------------------------------------------------
         x = 0.0d0
         y = 0.0d0

c-----------------------------------------------------------
c        For each step in the walk ...
c-----------------------------------------------------------
         do i = 1 , nsteps
c-----------------------------------------------------------
c           Generate a random direction, take a step in
c           that direction, and accumulate <r**2(t)>
c           for subsequent normalization (averaging).
c-----------------------------------------------------------
            randir = roll(4)
            if(      randir .eq. 1 ) then
               x = x + 1.0d0
            else if( randir .eq. 2 ) then
               x = x - 1.0d0
            else if( randir .eq. 3 ) then
               y = y + 1.0d0
            else if( randir .eq. 4 ) then
               y = y - 1.0d0
            else
               write(0,*) 'rw2d: Unexpected random direction ',
     &                    randir
               stop
            end if
            d = sqrt(x**2 + y**2)
            rsqavg(i) = rsqavg(i) + d**2
         end do
         if( ltrace ) then
            if( mod(iwalk,50) .eq. 0 ) then
               write(0,*) 'rw2d: Done walk ', iwalk
            end if
         end if
      end do

c-----------------------------------------------------------
c     Normalize <r**2(t)> and output ...
c-----------------------------------------------------------
      do i = 1 , nsteps
         rsqavg(i) = rsqavg(i) / nwalks
          write(*,*) i, rsqavg(i)
      end do

      stop
```

```
c-----------------------------------------------------------
c     Usage exit ...
c-----------------------------------------------------------
 900  continue
         write(0,*) 'usage: rw2d <nsteps> <nwalks>'
      stop

      end
c-----------------------------------------------------------
c     Returns uniformly distributed random integer chosen
c     from 1 to n.
c-----------------------------------------------------------
      integer function roll(n)
         implicit     none

         real*8       rand

         integer      n

         roll = min(n,1 + int(n * rand()))

         return

      end
```

Sample output



`gnuplot` *input which produced the above plot*

```
set terminal postscript portrait
set title "<r^2(t)> for 10, 100 and 1000 walks"
set xlabel "t (step number)"
set ylabel "r^2"
set size 0.760,1.0
set output "distance.ps"
plot [0:5000] [0:5000] 'out10' t ' ' with dots 3, \
     'out100' t ' ' with dots 3, 'out1000' t ' ' with dots 3
quit
```

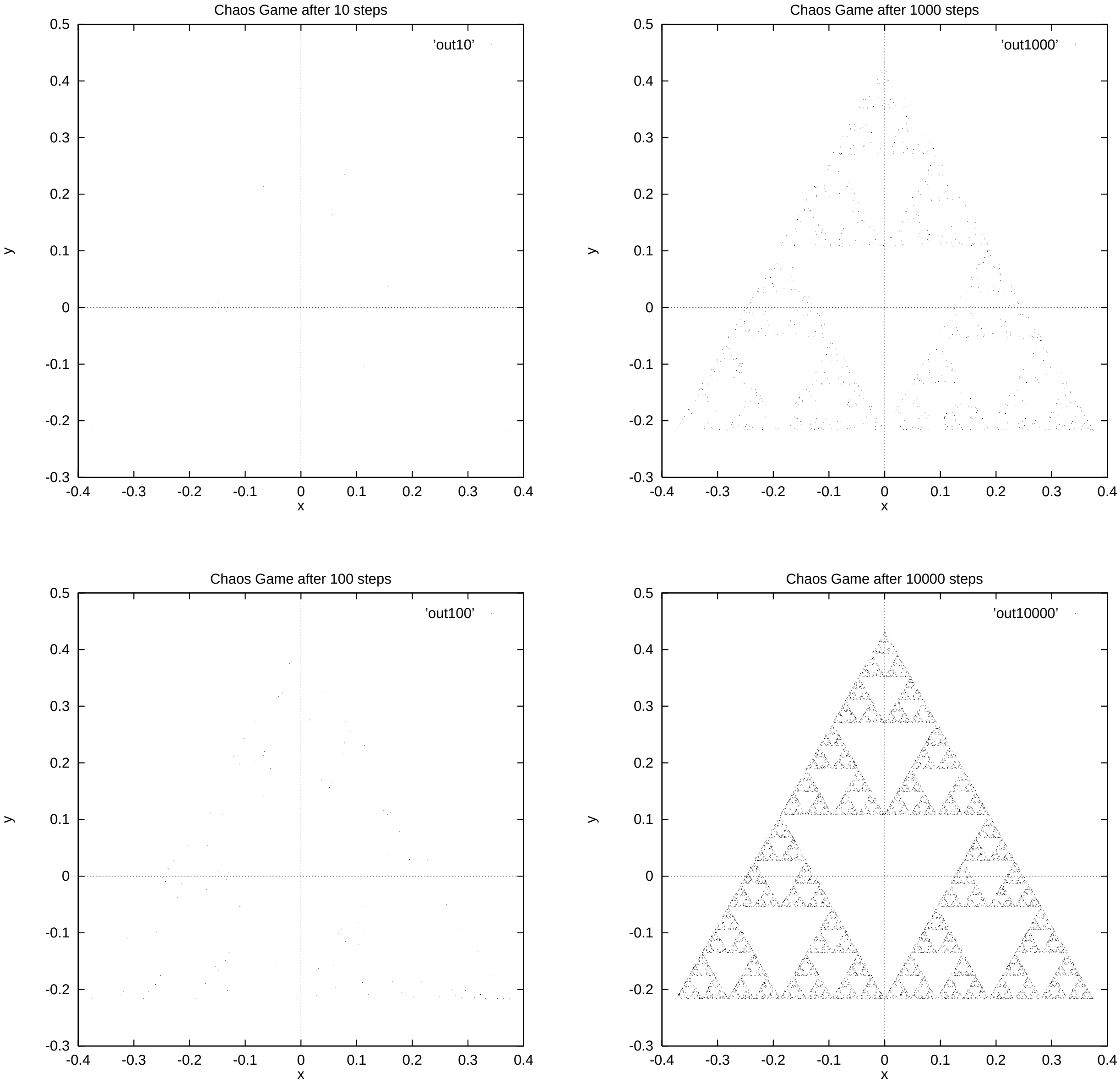
As the number of walks gets large, the plot of $< r^2(t) >$ (where we can conveniently interpret the step-number as the "time", t) appoaches the function t, as expected from the standard analysis of random walks (see e.g. *The Feynman Lectures in Physics, Volume I, Section 6.3 and following*).

Problem 7:

The self-similar pattern which emerges as the game unfolds is known as the Sierpinski triangle or Sierpinski gasket.

Sample source code:

```
c-----------------------------------------------------------
c     PHY/CAM 381C, Homework 1, Problem 7
c
c     chaos2d: Plays game of chaos in 2D.
c
c     Uses 'roll' function defined in previous problem.
c-----------------------------------------------------------
      program        chaos2d

      implicit       none

      real*8         pi
      parameter    ( pi   = 3.141 5926 5358 9793 d0 )

      integer        iargc,       i4arg,      roll

      real*8         vx(3),       vy(3)
      real*8         xactive,     yactive,    pi23,
     &               r,           theta
      integer        i,           nsteps,     randv

      if( iargc() .ne. 1 ) go to 900
      nsteps = i4arg(1,-1)
      if( nsteps .eq. -1 ) go to 900

c-----------------------------------------------------------
c     Initialize vertices of equilateral triangle with unit
c     side, centred at origin with one vertex on positive
c     y-axis.
c-----------------------------------------------------------
      r = 0.25d0 * sqrt(3.0d0)
      pi23 = 2.0d0 * pi / 3.0d0
      theta = 0.5d0 * pi
      do i = 1 , 3
         vx(i) = r * cos(theta)
         vy(i) = r * sin(theta)
         theta = theta + pi23
         write(*,*) vx(i), vy(i)
      end do

      xactive = 1.0d0
      yactive = -0.5d0
      write(*,*) xactive, yactive
      do i = 1 , nsteps
         randv = roll(3)
         xactive = 0.5d0 * (xactive + vx(randv))
         yactive = 0.5d0 * (yactive + vy(randv))
         write(*,*) xactive, yactive
      end do

      stop

 900  continue
         write(0,*) 'usage: chaos2d <nsteps>'
      stop

      end
```


Chaos Game after 10 steps
'out10'
Chaos Game after 1000 steps
'out1000'
Chaos Game after 100 steps
'out100'
Chaos Game after 10000 steps
'out10000'
x
y
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
0.4