Name: ____________________________

Directions: **Work only on this sheet** (on both sides, if needed); do not turn in any supplementary sheets of paper. There is actually plenty of room for your answers, as long as you organize yourself BEFORE starting writing.

1. In this problem you will enhance the **textfile** class on p.22.

First, you will add a member variable **tfiles**, a list of pointers to all the files for which **textfile** instances currently exist.

Second, you will add method named **cat()**, which has just a single argument, whose name is **outflname**. This function will concatenate all the files in **tfiles**, outputting the result to a new file whose name is given by **outflname**. Use the open-for-writing form of `open()`, which just involves adding 'w' as a second argument, and **writelines()**, which works as the opposite of **readlines()** except that now there is an argument, the outfile name. You should also use the **close()** method for files. You can read examples on p.52 if you wish, but it's not necessary, as all the information is above.

If for example file **a** consists of

```
abc
de
f
```

and file **b** consists of

```
8
168
```

then the concatenated file contents are

```
abc
de
f
8
168
```

PLEASE WRITE YOUR SOLUTION AS FOLLOWS: Simply write the new lines that must be added; don't copy down the entire existing **textfile** class code. So, write something like, "In between lines 5 and 6, insert the following code..."

2. Consider the unit square S in the plane, with lower-left corner at (0,0) and upper-right corner at (1,1). We are interested in distances from points in this square to (1,0). There also is a smaller rectangle R, of width 2w and height h, with lower left point (0.5-w,0) to and upper-right point (0.5+w,h) (sides parallel to the outer square).

We are interested in the minimum travel distance to (1,0) for each point in S that is not in R, under the constraint that travel is not allowed within R. Note (see the function **d()** below) that we are using "Manhattan street distance," which means paths consist only of vertical and horizontal segments.

Say for instance w = 0.25 and h = 0.50, and we are considering the point (0.20,0.10). The shortest path to (1,0) consists first of going to (0.25,0.50), then along the top of R, and then to (1,0), for a total distance of 0.05 + 0.40 + 0.50 + 0.50 + 0.25.

We set up an nxn grid of points within S [(0,0) through $(\frac{n-1}{n}, \frac{n-1}{n})$, and for each one wish to compute the length of the shortest path to (1,0). For points in R, we define this distance to be -1.0.

The function **getdists(w,h,n)** below returns the n^2 distances in a list of lists (i.e. two-dimensional "array"). Fill in the details.

```
import math

def d(x,y,x1,y1):
   return abs(x1-x) + abs(y1-y)

# returns the minimum distance
# from (x,y) to (1,0) (or returns -1.0)
def calcdistto10(x,y,w,h):
   # insert 1 or more lines here
   # ...

def getdists(w,h,n):
   # insert 1 or more lines here
   # ...
   return dists
```

IMPORTANT NOTE: Don't worry whether boundary lines of R count as part of R or not.

Solutions:

1.

```
class textfile:
   ntfiles = 0  # count of number of textfile objects
   fls = []
   def __init__(self,fname):
      textfile.ntfiles += 1
      textfile.fls.append(self)
      self.name = fname  # name
      self.fh = open(fname)  # handle for the file
      self.lines = self.fh.readlines()
      self.nlines = len(self.lines)  # number of lines
      self.nwords = 0  # number of words
      self.wordcount()

   def wordcount(self):
      "finds the number of words in the file"
      self.nwords = \
         reduce(lambda x,y: x+y,map(lambda line: len(line.split()),self.lines))
   def grep(self,target):
      "prints out all lines containing target"
      lines = filter(lambda line: line.find(target) >= 0,self.lines)
      print lines
   def cat(outflname):
      ofl = open(outflname,'w')
      lns = []
      for fl in textfile.fls:
         lns += fl.lines
      ofl.writelines(lns)
      ofl.close()
   cat = staticmethod(cat)
```

2.

```
def d(x,y,x1,y1):
   return abs(x1-x) + abs(y1-y)

def calcdistto10(x,y,w,h):
   if x > 0.5 - w and x < 0.5 + w and y < h: return -1.0
   if x < 0.5 - w and y < h:
      return d(x,y,0.5-w,h) + 2*w + h + (0.5-w)
   return d(x,y,1,0)

def getdists(w,h,n):
   dists = []
   for i in range(n):
      rowofdists = []
      for j in range(n):
         tmp = calcdistto10(float(i)/n,float(j)/n,w,h)
         rowofdists.append(tmp)
      dists.append(rowofdists)
   return dists
```