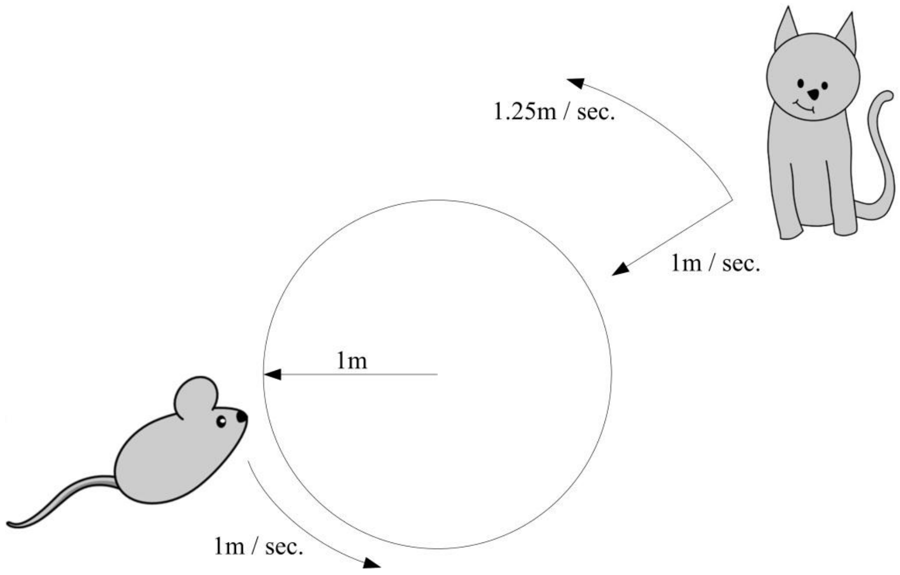
Project One: Cat’s Lunch

Background

A hungry cat is hunting for its lunch. Luckily, the cat finds a mouse run around the base of a statue, which is circular and one meter in radius. Suppose that the witless mouse always runs around the base of the statue counterclockwise with a uniform speed of one meter per second. Now the cat decides to chase this mouse for its lunch. Every second, the cat pursues the mouse as follows: If the cat can see the mouse, the cat moves one meter toward the statue. If the cat can't see the mouse, the cat circles 1.25 meters counterclockwise around the statue with its radius unchanged.

The cat plans eventually to get close enough to the mouse to make it a juicy lunch. The mouse by accident, however, may manage to keep completely out of sight of the cat, since both the cat and the mouse are moving counterclockwise. The cat will eventually tire of the chase if this happens.



Problem

You are supposed to write a program to simulate this process and to determine if the cat catches the mouse.

Assumptions

To make things simple, we make the following assumptions:

1. We are working in a polar coordinate system. Assume that the origin of this system is the center of the statue.

2. We ignore the geometric size of both the cat and the mouse. We just treat them as two points on the two-dimensional plane.

3. We are doing the discrete-time simulation instead of the continuous-time simulation. We analyze the process at each integral second, such as 1st second, 2nd second, etc. For each second, we assume that the cat moves according to the location of the mouse **at the beginning of that second**.

4. The cat catches the mouse when it (the cat) moves past the mouse during a certain second while at the base of the statue, i.e. when the radius of the cat's movement is one meter.

5. The cat cannot move inside the statue's base; hence if the cat is, say, at radius 1.7 meters and sees the mouse, it can move in only 0.7 meters, up to the base of the statue.

6. If the cat reaches the base of the statue in a certain second, we assume that it reaches the base at the end of that second. Since the cat reaches the base at the end of that second, the cat cannot catch the mouse during that second even if the final location of the cat and the mouse are the same.

7. Since both the cat and the mouse are moving counterclockwise, the mouse by accident may manage to keep completely out of sight of the cat. The cat will eventually tire of the chase if this happens. We assume that if the cat has not caught the mouse after 30 seconds of moving, it gives up.

8. Define $\pi = 3.1415927$.

Inputs and Outputs

Rather than ask you to write the input and output routines for this assignment, we have provided a library that does this I/O for you. The definitions are given in the header file io.h seen below:

```
************************************************************************
#ifndef IO_H
#define IO_H

#include <string>
```

```
using namespace std;

double getParam(const string& prompt);
// EFFECTS: prints the prompt, and reads a double from cin.
// Returns the value it reads.

void printHeader();
// EFFECTS: prints out a nice header for the table recording the
// simulation process.
// MODIFIES: cout

void printData(int second, double mouseAngle, double catAngle,
     double catRadius);
// EFFECTS: prints out a row in the simulation table.
// MODIFIES: cout

void printResult(bool catches);
// EFFECTS: prints out the final result of whether the cat
// catches the mouse.
// MODIFIES: cout

#endif
```
****

The inputs to your program contain three parameters: 1) the initial angle of the mouse, 2) the initial angle of the cat, and 3) the initial radius of the cat. The angles should be provided in terms of degree, e.g., 45°, and the radius should be provided in terms of meter. Use the getParam() function to obtain the three parameters. To obtain each parameter, your program must ask the user to type it in. Use the prompt argument for this purpose. The three prompts are:

"Please enter the initial mouse angle: "

"Please enter the initial cat angle: "

"Please enter the initial cat radius: "

You must use exactly these prompts. Don't forget the trailing single spaces!

You may assume that the user provides numeric values. The mouse angle and the cat angle can be any real values. However, the initial radius of the cat must be a value greater than or equal to one, which is the radius of the statue base. You need to check the value of this input. **If the user gives a value less than one, you should first issue an error notice exactly like**:

“Error: the cat radius is less than the base radius 1!”

Then, you should prompt again to request input for the cat radius until you get a legal value.

Your program should print the position of the cat and the mouse **at the end of each second**, **starting from the 0th second**, which corresponds to the initial positions. Your program should display output in a nice table format. Your program should first call the printHeader() function to print out the table header. It should then call printData() once for each second of simulation. The table contains four columns, which include the second, the mouse angle, the cat angle, and the cat radius. If the cat catches the mouse eventually, the table should list the simulation data up to the second the cat catches the mouse. Otherwise, the table should list 31 rows, corresponding to the simulation data **at the end of** the 0th, 1st, up to 30th second. Finally, you program should call the printResult() function to print out the chase result of whether the cat catches the mouse.

An example of the entire output is demonstrated below (assuming that the initial mouse angle is 0, the initial cat angle is 90, and the initial cat radius is 2.):

```
Second    Mouse Angle    Cat Angle Cat Radius
------    -----------    --------- ----------
0                0.00        90.00       2.00
1               57.30       125.81       2.00
2              114.59       161.62       2.00
3              171.89       161.62       1.00
4              229.18       233.24       1.00
The cat catches the mouse!
```

Notice that in this example, the cat catches the mouse in the middle of the fourth second. However, we are doing discrete-time simulation. To make things simple, for the last second, you only need to print the final location of the cat and the mouse **assuming that the cat does not catch the mouse, i.e., the cat and the mouse move according to the position updating rules we define before**.

Since degree *d* ° and (*d*+360) ° are the same, we require you to always convert the degree into the interval [0 °, 360 °). For example, the degree -1 ° should be converted to 359 °.

Hints

1. The cat sees the mouse if the value

(catRadius) * cos (catAngle – mouseAngle)

is at least 1.0. (Think about why it is the case.) Given this rule, when the cat radius equals the base radius, it sees the mouse if and only if the cat angle and the mouse angle are the same. If the cat and the mouse are at the same location at the beginning of a certain second, the cat catches the mouse during that second. However, in this special situation, be careful about the final location of the cat. Think about what it should be under our assumptions.

2. The `cos` function appears in the C++ math function library. Note that the argument to this function must be specified in radians, not degrees. In order to use this function, you need to include <cmath>:

```
#include <cmath>
```

Files

There are several files located in the Project-One folder of our Sakai Resources:

1. io.h: The header file for the I/O functions described above. This should be included (via #include "io.h") in your solution.

2. io.C: The implementation of the I/O functions. Should **not** be #included by your program.

3. test1: A sample input file for your program.

4. test1-result: The output that should be generated by input test1.

You should copy the above files into your working directory. **DO NOT modify io.h or io.C!** You may use only the C++ standard libraries, and no others.

Compiling and Testing

You are required to compile your program under the Linux environment. In order to do that, write a Makefile. **Name your program as “p1”.**

Your first test of your program is based on the `test1` file we provide for you. In order to use a file to provide your program with the inputs, you use the Linux I/O redirection function:

```
./p1 < test1 > output1
```

Then, the input to your program is given by `test1` and the output of your program is written into a file called `output1`.

You can check whether your program generates the correct output by doing a `diff` between `output1` and `test1-result`:

```
diff test1-result output1
```

If `diff` outputs nothing, your program passes the test. Otherwise, if `diff` reports any differences at all, you have a bug somewhere.

We will test your code using this test case, as well as several others. You should therefore definitely pass this test case. However, you should also create a family of test cases that exercises your program with different inputs, since the test case we have given you is not sufficient to catch all bugs.

Submitting and Due Date

You should submit your source code via the Assignment tool on Sakai. The due date is 11:59 pm on June 3rd, 2012. Your submission should include your source files together with the Makefile. Since we require you not to modify the io.h and io.C, we will ignore these two files you upload.

Grading

Your program will be graded along two criteria:

1. Functional Correctness

2. General Style

An example of Functional Correctness is whether or not you produce the correct output. General Style speaks to the cleanliness and readability of your code. We don't need you to follow any particular style, as long as your style is consistent and clear. Some typical style requirements include: 1) appropriate use of indenting and white space, 2) program appropriately split into subroutines, 3) variable and function names that reflect their use, and 4) informative comments at the head of each function.

Bonus Points

If you think some of your test cases are good, you can upload **no more than three** of your test cases together with a short document about what special situations your test cases cover. We will include some of the good test cases you provide into the entire test set. If yours is selected, you will be given some bonus points.