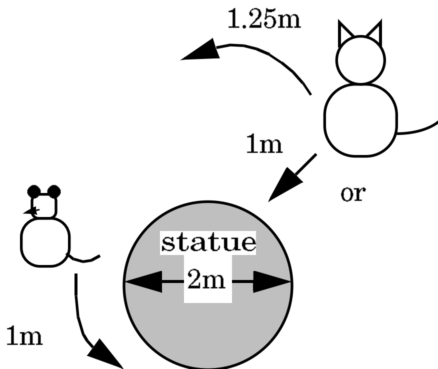
Fortran/Matlab version
M. Clancy and E. Lagache, U.C. Berkeley

Background

The scene is an urban park; a cat watches a mouse run around the base of a statue of the first computer. Over the course of a minute, the somewhat witless mouse moves one meter counterclockwise around the statues base, which is circular and two meters in diameter. Every sixty seconds, 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. The cat plans eventually to get close enough to the mouse to make a juicy lunch of it.



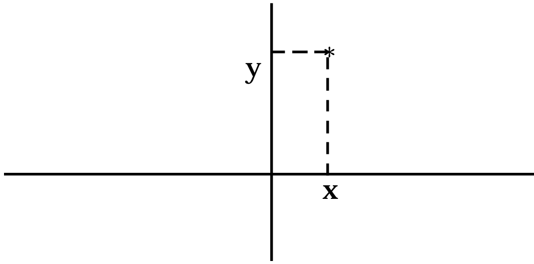
The Cat and the Mouse

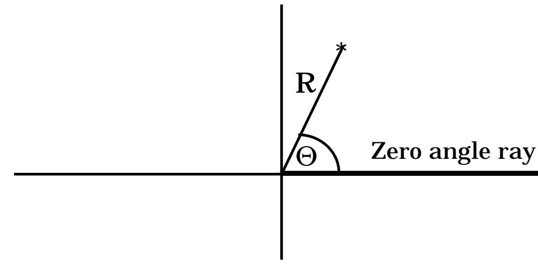
Problem

Write Fortran and Matlab programs to simulate this situation and determine if the cat catches the mouse. The Fortran program should ask the user for the initial positions of the mouse (angle) and the cat (both radius and angle). It should then simulate the chase using the rules listed above to determine how the cat and mouse move, with the cat moving first. The program should print the position of the cat and mouse after each move, followed by the results of the chase (either the mouse becomes lunch, or the cat gives up after 30 minutes). The Matlab program should be a collection of functions, one of which is called cateatsmouse; this function should take legal cat and mouse positions as arguments and run the simulation as just described, returning a true value if the cat catches the mouse and returning false if the cat gives up.

Problem representation

Because both the cat and mouse are running in circles around the statue, the best way to represent the positions of both animals is with polar coordinates. Since the radius of the mouse is always 1, you only need to keep track of the angle of the mouse. While you need to keep track of both the angle and the radius of the cat, it is still easier to work with polar coordinates because the cat only moves in only one direction (either angle or radius) per minute.





The decomposition we require

This problem is easier to think about (and work on) from the bottom up. The last thing you need to do (in at least some cases) is check if the cat has caught the mouse. In this simulation, the mouse will be considered caught if: 1) the cat is at the same radius as the mouse, and 2) the cat passes the mouse while running along the base of the status. Condition 1 is easy; the radius of the cat has to be 1. It turns out that condition 2 is true exactly when the following conditions hold:

either

Mouse Angle = Cat Angle

or

cos (Mouse Angle – Old Cat Angle) > cos (Cat Angle – Old Cat Angle), **and**
cos (Cat Angle – Mouse Angle) > cos (Cat Angle – Old Cat Angle).

You should write a logical function BTWEEN (angleA, angleB, angleC) that handles the latter case (see the appendix to this problem for more information on how this expression was derived).

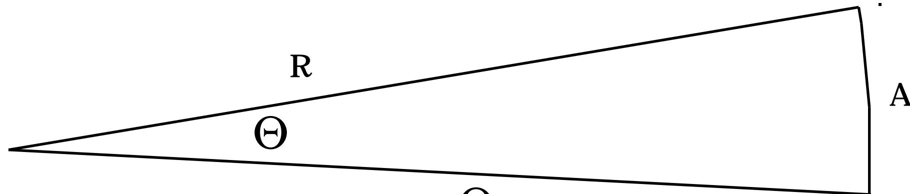
A similar problem is determining whether or not the cat can see the mouse. It turns out that the proper expression for that is

(cat radius) * cos (cat angle – mouse angle) **is at least 1.0**

Your program should include a logical function SEES that returns true if the cat sees the mouse (again, see the appendix for a derivation of this result). In terms of actually developing this program, you should probably stop and test these functions before going on to write any more parts of this assignment.

From here, you might want to write the functions (or subroutines) that return the movements of the cat and mouse. Three are required: one for the mouse, one for when the cat moves toward the statue, and one for when the cat moves around the statue. The first two are very simple mathematical expressions. However, your subprogram to move the cat toward the statue should not allow the cat to move in closer than 1 meter (the base of the statue). The last subprogram is relatively easy to implement if you remember the relationship for arc length:

The need to deal with radians in fact occurs everywhere in the program because the trigonometric functions built into Fortran expect angles in radians, not degrees. Most students write functions to convert from degrees to radians and back again so that they can enter values in a more familiar notation. Should you choose to do that, we recommend that you keep all your variables in radians and change values only



Length of arc A is: $A = R * \Theta$
Where Θ is measured in **Radians!!**

when doing input/output. That greatly decreases the risk of bugs resulting from angles in the wrong format. In any case you should write a function that reduces angles when they become greater than 2π (the AMOD function can be very useful for this). This function is needed because the Fortran trig functions become less accurate for angles that are out of the range 0 to 2π.

The Fortran main program

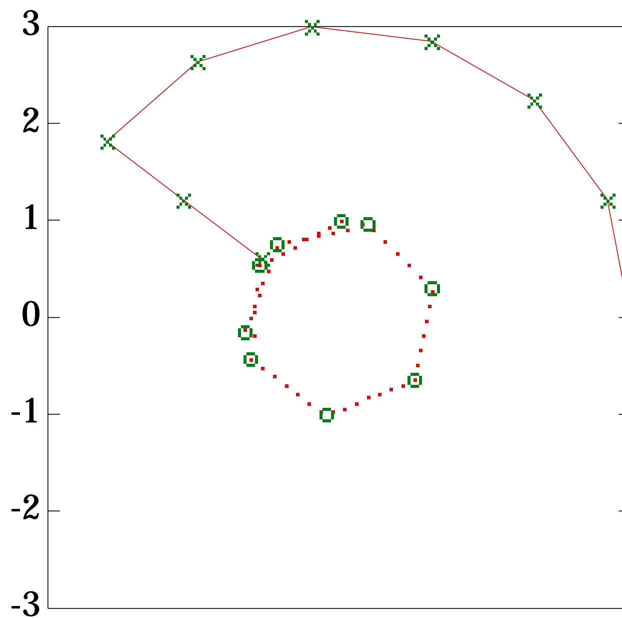
Finally one can sketch out the structure of the main program. It will need to read in the positions of the cat and mouse (this should be done by still another subprogram). There is one bit of error checking that must be done. You should check to make sure that the entered radius of the cat is greater than 1, otherwise you could place the cat inside the statue! After reading in the positions, checking them, and converting the angles to radians (if necessary), your program should enter the simulation loop. Each minute, you should do the following things: check if the cat has caught the mouse; if not, move both animals based on the rules given above (and encoded in your movement subprograms); finally, print out the status of the simulation. The information printed should include the current time, the current angle of the mouse, the current angle of the cat, and the current radius of the cat. This output should be in a columnar table format. You may wish to use FORMAT statements to achieve this.

You should use only structured loops in your program (i.e. simulated WHILE or REPEAT-UNTIL loops). In order to make this assignment fit those type of loops, you will need to embed an IF statement within the loop to deal with the two distinct possibilities (either the cat catches the mouse at this minute, or it doesnt). When your program finally exits from the loop you should print out a summary of the *events* (did the mouse get caught or not) and how much time had elapsed.

The Matlab cateatsmouse function

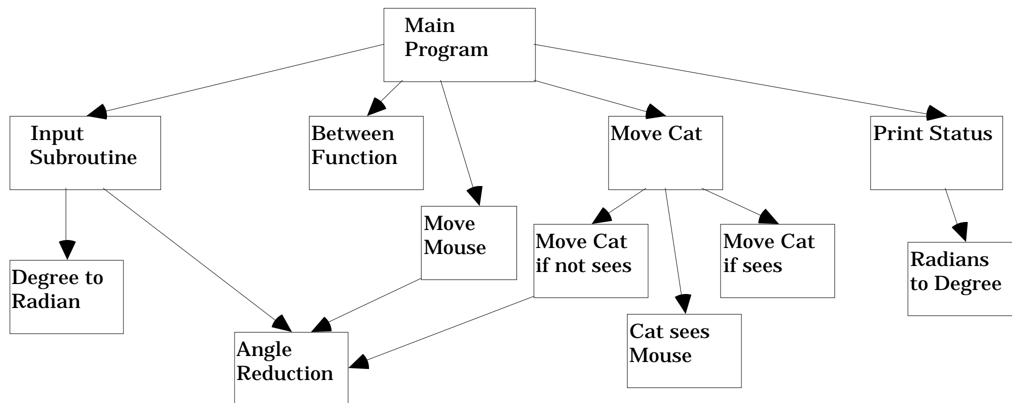
The cateatsmouse function of the Matlab version should perform somewhat differently. It need not do any error checking on the cat and mouse positions supplied as arguments. Instead of printing the successive cat and mouse positions inside the simulation loop, it should save them in vectors and plot those vectors in each loop iteration, producing something like the plot below in which the cat circles six times and moves toward the mouse twice.

The polar command described in chapter 3 of Etter will be most appropriate for this purpose; the pause command will also be useful.



Decomposition summary

One can quickly summarize the subprograms needed for this assignment by a *decomposition tree*. One is presented below. Boxes indicate program components; arrows show which components call or include others. It is certainly not the only way this problem could be solved, but is a useful way to think about the problem.



Assignment requirements

Your Fortran and Matlab programs are to include the following subprograms:

- a Fortran subroutine or Matlab function to change the radius and angle of an animal by given amounts;
- a real function to reduce an angle to the equivalent angle less than 2π (this is not necessary in the Matlab version);
- a logical function to determine if the cat sees the mouse;
- a logical function to see if an angle is between two others.

These functions should be tested separately using techniques described earlier. Print out copies of your test programs and sample output to show to the tutor when you have your program graded. Finally, your Matlab program should produce plots of the cat and mouse positions as described previously.

Your test data should include the following values:

- cat radius = 0.5,
- cat radius = 6.0,
- cat radius = 1.0, cat angle = 35.0 degrees, mouse angle = 396.0 degrees
- cat radius = 8.1, cat angle = 0.0 degrees, mouse angle = 45.0 degrees
- cat radius = 8.1, cat angle = 150.0 degrees, mouse angle = 240.0 degrees
- cat radius = 4.0, cat angle = 0.0 degrees, mouse angle = 57.0 degrees

plus whatever other values are necessary to ensure that all statements in your program have been executed at least once. Be sure to check that the output makes sense for these tests, since the output is what most tutors will check first. Students are sometimes embarrassed to have a tutor point out that their programs have the cat moving deeper and deeper inside the statue, or moving away from the mouse instead of approaching it. (Your Matlab plots should make such mistakes much more obvious.)

System information

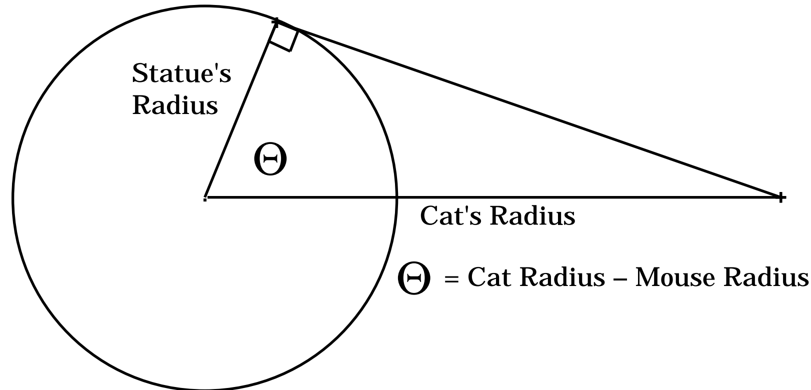
Plotting from UNIX Matlab requires that the DISPLAY shell variable be set. Normally, this is done automatically when you log in, unless

- you log in from a non-workstation, or
- you use the telnet or rlogin program to run Matlab on a computer different from the workstation youre sitting at.

Type the command echo $DISPLAY to the UNIX command prompt to find out if the variable is set. If you’re not using a computer with graphics capabilities, you wont be able to display anything anyway. If you plan to log in on a workstation remotely, ask a tutor how to set things up to enable Matlab to run easier.

Derivation of formulas used in "Cat and Mouse"

Most of the mathematical expressions used in the program in fact capture very clear visual situations. Let’s start with the SEES relation. Our model of the cat assumes that it is looking straight at the statue, so the limit of the cat’s vision is given by the two rays that are tangent to the statue’s base. Let’s look at one; the other will be symmetric about the line between the cat and the statue. The diagram below has all the essential facts:



Since the line of sight is tangent to the circle, it forms a right angle with the radius. So from the definition of cosine we get

cos(Q) = (Statue’s Radius) / (Cat’s Radius)

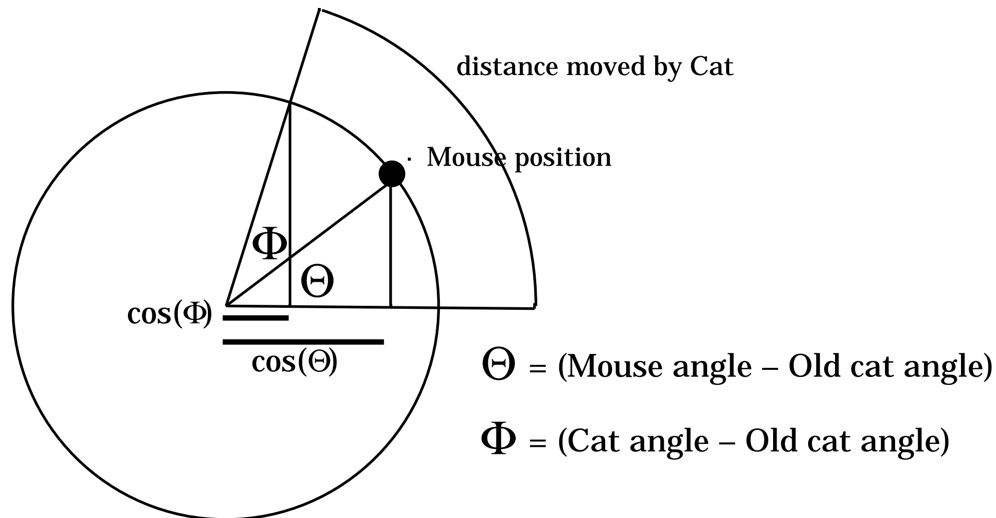
Since the statue’s radius is by definition 1.0, we can insert that into the expression and multiply both sides by the cat’s radius:

(Cat’s Radius)*cos(Q) = 1.0

That gives us the limiting condition. To confirm the inequality one need only note that the cat can see the mouse for all angles less than Q, and the cosine function increases for decreasing angles.

Finding the expression for BTWEEN is a bit more difficult because there are three angles: the previous cat angle, the current mouse angle, and the current cat angle. To solve this problem we will subtract away one angle to change to a more convenient frame of reference (as if the old cat angle was the zero position). The intuition behind the BTWEEN relation is to project the distance traveled by both the cat and mouse

onto the line between the old cat position and the center of the statue. Those distances will be cosines of some angles shown on the following diagram:



From the diagram it is clear that if the mouse position is between the cat's old position and the cat's new position, the line traveled by the mouse must be longer than the line traveled by the cat (the cat has to get closer to the origin). So we get the first part of the BTWEEN relation:

$$\cos(Q) > \cos(F)$$

The second condition is a check to make sure that the angles are in the same quadrant:

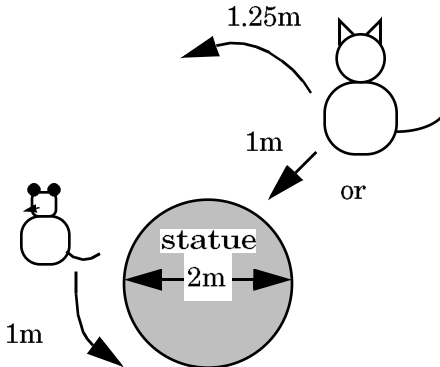
$$\cos(F - Q) > \cos(F)$$

will be true is if F is a positive angle (cosine of a smaller angle is larger), but will not be true if F is negative. Requiring both conditions makes sure that the above diagram is correct, and that the mouse has been indeed passed by the cat.

Pascal version
M. Clancy, U.C. Berkeley

Background

The scene is an urban park, where a cat watches a mouse run around the base of a statue of Klaus Wirth. Over the course of a minute, the somewhat witless mouse moves one meter counterclockwise around the statues base, which is circular and two meters in diameter. Every sixty seconds, the cat pursues the mouse as follows. If the cat can see the mouse, the cat moves one meter in toward the statue. If the cat can't see the mouse, the cat circles 1.25 meters counterclockwise around the statue. The cat plans eventually to get close enough to the mouse to make it a juicy lunch.



The Cat and the Mouse

Write a program to keep track of the cat's pursuit of the mouse. Your main program should contain statements to input and check the initial positions of the cat and mouse; within the program, all positions should be represented in polar coordinates, and the mouses radius should be assumed to be one meter. It should also contain a loop, each repetition of which represents the passage of one minute, and which contains calls to procedures to move the mouse and the cat.

Your program should contain, within its loop, statements to print the new positions of the cat and mouse after each minute, and to keep track of how much time the cat is taking to catch the mouse. The output should be in the form of a table for maximum readability. (Informative output at other places in the program may also prove useful.)

Here are an assortment of useful facts and things to watch out for.

1. The cat sees the mouse if

    (cat radius) * cos (cat angle – mouse angle)

    is at least 1.0.

2. When the cat circles distance d around the statue, its radius does not change, and the change in its angle can be calculated from the relationship

    d = angle * (radius of arc)

3. The cat catches the mouse when it (the cat) moves past the mouse while at the base of the statue, i.e. when the cat radius is 1.0 and the mouse angle lies

between the old cat angle and the new cat angle. Your program should include a boolean function

```
IsBetween (angleA, angleB, angleC)
```

that returns true if angleB is between angleA and angleC, and false otherwise. If angleA and angleC are less than two radians apart, the between condition holds exactly when

cos (angleB – angleA) > cos (angleC – angleA), **and**
cos (angleC – angleB) > cos (angleC – angleA).

4. Note that the cat cannot move inside the statue's base; hence if the cat is, say, at radius 1.7 and sees the mouse, it can move in only 0.7 meters, up to the base of the statue.
5. The mouse by accident 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. If the cat has not caught the mouse after thirty minutes of moving, you should print an appropriate message and stop the program.
6. Remember that angles a, a + 2π, a + 4π, ... are all equal.

Miscellaneous requirements

Your program is to include procedures to move the cat if it sees the mouse, to move the cat if it doesn't see the mouse, and to move the mouse; a real function to reduce an angle to the equivalent angle less than 2π; and boolean functions to determine if the cat sees the mouse and if an angle is between two others (see 3 above). Pass the information these subprograms need through parameters. You should then test these subprograms singly, using appropriate test data, to see if they work independently before putting them all together. Tutors may ask to see the results of these tests. In no case should any of your routines contain more than twenty-four lines of Pascal.

The test data for your complete program should include the following values:

cat radius	cat angle	mouse angle
0.5		
6.0		
1.0	35.0	396.0
8.1	0.0	45.0
8.1	150.0	240.0
4.0	0.0	57.0

plus whatever other values are necessary to ensure that all statements in your program have been executed at least once.

Checklist

program listing;

program appropriately split into subprograms,
including an IsBetween function, a Sees function, an angle reduction function, and movement procedures, using parameters as specified;

evidence of independent tests of the functions and subroutines;

no routine more than twenty-four lines of code;

variable and subprogram names which reflect their use;

comments at the head of the main program and subprograms which explain the purpose of each;

informative tabular output;

cat radius less than 1 detected and dealt with appropriately;

clean case analysis and simple loop structuring;

appropriate use of Pascal's various looping and testing constructs;

correct execution: loop termination conditions, loop entry conditions, case statement adequately guarded;

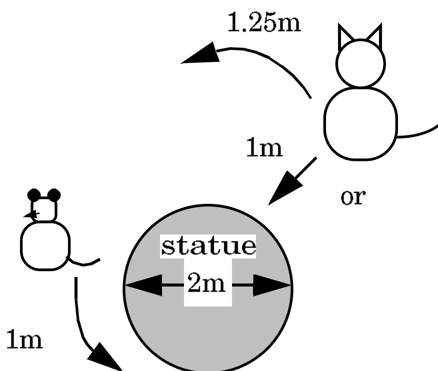
test data as specified.

C++ version, no inheritance
M. Clancy, U.C. Berkeley

Background

The scene is an urban park; a cat watches a mouse run around the base of a statue of Bjarne Stroustrup. Over the course of a minute, the somewhat witless mouse moves one meter counterclockwise around the statues base, which is circular and two meters in diameter. Every sixty seconds, 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.

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.



The Cat and the Mouse

Problem

Complete the program found in ~cs9f/lib/cat+mouse.cc to produce a simulation of this situation and determine if the cat catches the mouse. Don't change any of the code already provided in that file. The program uses a Position class declared in ~cs9f/lib/positions.h; you are also to provide a file positions.cc that correctly implements the operations of this class.

The program first calls the GetPositions function to ask the user for the initial positions of the mouse (angle in degrees) and the cat (both radius and angle in degrees). You may assume that the user provides legal position values.

The program then calls the RunChase function to simulate the chase using the rules listed above. (The cat moves first.) Your code should use member functions of the Position class where appropriate. It should print the position of the cat and mouse after each move, followed by the results of the chase (either the mouse becomes lunch, or the cat gives up after 30 minutes).

Useful information for writing positions.cc

A number of the member functions you are to write for the Position class involve trigonometry. Given below are some formulas.

1. The cat sees the mouse if

   (cat radius) * cos (cat angle – mouse angle)

   is at least 1.0.

2. When the cat circles distance d around the statue, its radius does not change, and the change in its angle can be calculated from the relationship

   d = angle * (radius of arc)

3. The cat catches the mouse when it (the cat) moves past the mouse while at the base of the statue, i.e. when the cat radius is 1.0 and the mouse angle lies between the old cat angle and the new cat angle. An angle B is between angles A and C in the following circumstances:

   cos (B – A) > cos (C – A), and
   cos (C – B) > cos (C – A).

   The difference C – A is assumed to be less than 180°, or π radians.

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

5. The mouse by accident 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. If the cat has not caught the mouse after thirty minutes of moving, you should print an appropriate message and stop the program.

6. Remember that angles a, $a + 2\pi$, $a + 4\pi$, ... are all equal.

The cos function appears in the C++ math function library. Note that the argument to this function must be specified in *radians*, not degrees. A source file that uses any of the math functions must include the line

```
#include <math.h>
```

The command that creates an executable version of the program must include the term lm (minus-ell-em), for example as follows:

```
g++ cat+mouse.cc positions.cc -lm
```

(You can abbreviate this command using the make program described later in this document; a make file for this assignment is available in ~cs9f/lib/p2a.makefile.) *Computing Concepts* second edition says you can use

```
#include <cmath>
```

instead, and omit the lm. This seems to work, but we haven't tested it thoroughly.

It's legal and appropriate to assign one object to another, for instance by saying

```
oldCatPosition = newCatPosition;
```

Note also that an object's member function (e.g. IsBetween) can access the private data of any other object of the same class.

Miscellaneous requirements

You should test the member functions of the Position class in isolation, using appropriate test data, to see if they work independently before calling them from the RunChase function. Tutors may ask to see the results of these tests. Your test data should include the following values:

cat radius	cat angle	mouse angle
1.0	35.0°	396.0°
8.1	0.0°	45.0°
8.1	150.0°	240.0°
4.0	0.0°	57.0°

plus whatever other values are necessary to ensure that all statements in your program have been executed at least once. Be sure to check that the output makes sense for these tests, since the output is what most tutors will check first. Students are sometimes embarrassed to have a tutor point out that their programs have the cat moving deeper and deeper inside the statue, or moving away from the mouse instead of approaching it.

Your code should call Position member functions where appropriate; code in cat+mouse.cc should not duplicate their functionality. You should not change the public interface of the Position class; you may, however, provide additional private member functions. (Don't forget to provide the function prototypes for any functions you provide.) Your program should also adhere to standards described in the section Style guidelines of this document.

Checklist

Correctly working code.

No changes made to functions already provided in cat+mouse.cc, or to the public interface in positions.h.

Sufficient testing, with output sufficient to verify test correctness:

tests on specified values for cat and mouse positions;
evidence of independent tests of all functions in positions.cc;
tests sufficient to exercise all statements in the program.

Adherence to CS 9F style standards:

appropriate use of indenting and white space;
avoidance of forbidden C++;
variable and function names that reflect their use;
informative comments at the head of each function;
no routine more than twenty-four lines of code.

Clean case analysis and simple loop structuring.

Suitable input prompts and informative output, including at least the positions of the cat and mouse after each move, and the final result of the chase.

cat+mouse.cc framework

```
#include <iostream.h>
#include <math.h>
#include "positions.h"

// You define the GetPositions function.
// It should read legal cat and mouse positions from the user
// and return the position values in its two arguments.
void GetPositions ( ... [you fill these in] ) {
   ... [you fill this in]
}

// You define the RunChase function.
// Given initialized cat and mouse positions,
// it should simulate the cat chasing the mouse, printing the
// result of each movement of cat and mouse.  Either the cat will
// catch the mouse, or 30 time units will go by and the cat will
// give up.

void RunChase ( ... [you fill this in] ) {
   ... [you fill this in]
}

int main () {
   Position cat, mouse;
   GetPositions (cat, mouse);
   RunChase (cat, mouse);
   return 0;
}
```

positions.h

```
#ifndef POSITIONS_H
#define POSITIONS_H

// typedef int bool; // necessary for some C++ programming environments

class Position {
public:
   // Initialize a position.
   Position (); // radius = 1, angle = 0.
   Position (float r); // angle = 0.
   Position (float r, float thetaInRadians);

   // Reinitialize a position.
   void SetAbsolutePosition (float r, float thetaInRadians);

   // Change the current position, incrementing the radius by
   // rChange and incrementing the angle by distChange .
   // Negative radius values represent movement toward the statue.
   // Positive distance changes represent counterclockwise
   // motion; negative values are clockwise.
   void IncrementPosition (float rChange, float distChange);

   // Print a position.
   void Print ( );

   // Return true if someone at the position can see someone or
   // something at the argument position (i.e. the statue does
   // not block one's view), and return false otherwise.
   bool Sees (Position pos);

   // Return true if the position is at the base of the statue,
   // i.e. its radius = 1, and return false otherwise.
   bool IsAtStatue ( );

   // Return true if the position is between the first argument
   // position and the second.  Precondition: the counterclockwise
   // difference between the first and second argument positions
   // is less than π radians, and the radii of all the positions
   // are the same.
   bool IsBetween (Position pos1, Position pos2);

private:
   float radius;
   float angleInRadians;
};

#endif
```

C++ version with inheritance
M. Clancy, U.C. Berkeley

The directory ~cs9f/lib/inherit/cat+mouse contains a version of the cat-and-mouse programs. You are to complete it by adding a module that uses inheritance. The following files are provided in ~cs9f/lib/inherit/cat+mouse.

cat+mouse.cc

This contains the main program, which constructs a scene of animals running around the statue and runs the simulation. The simulation stops when one of the animals in the scene captures its target. The code doesn't do much error checking. Dont worry about that; just use it unmodified.

park.h and park.cc

These comprise the module for the scene and the basic animal object. The Scene constructor prompts the user for the name, animal type, starting position, and target object for each animal in the simulation. The Animal class is the base class for simulation animals. It defines a virtual Chase method that Animal objects use to move.

positions.h, hp.positions.o, dec.positions.o, and solaris.positions.o

These comprise a module for positions similar to what you wrote for the flow of control project. Use the appropriate .o file depending on whether you work on a Hewlett-Packard workstation (in Soda or Cory), a DEC workstation (in Cory), or a workstation running Solaris (in Soda). If youre working on another platform, you will have to implement the changes yourself to the positions module from the flow-of-control project, but they wont be difficult.

You are to provide files animals.h and animals.cc that define classes Mouse, Cat, and Person. A Mouse object doesnt chase anything; it merely moves counterclockwise around the statue, one meter per call to Chase. A Cat object has a Mouse object as its target. If the cat sees its target, it moves one meter toward the statue; otherwise it circles 1.25 meters counterclockwise around the statue. A Person object is trying to photograph the situation. It doesn't try to capture anything. If it sees its target, it doesn't move; otherwise it circles 2 meters *clockwise* around the statue.

Mouse, Cat, and Person are to be derived from class Animal. Provide enough diagnostic output in each member function so that a tutor can see whether its performing as specified. Test your program in the same way you did for the flow-of-control project. Your program should also adhere to standards described in the section “Style guidelines” in this study guide.

cat+mouse.cc

```
#include <iostream.h>
#include "inherit/cat+mouse/park.h"
void RunChase (Scene allAnimals) {
   for (int time=1; time<=30; time++) {
      for (int k=0; k<allAnimals.Length (); k++) {
         if (allAnimals[k]->Chase ()) {
            return;
         }
      }
      cout << endl;
   }
   cout << "Chase took too long; all animals drifted away."
        << endl;
}

int main () {
   Scene allAnimals;
   RunChase (allAnimals);
   return 0;
}
```

park.h

```
#ifndef PARK_H
#define PARK_H
#include "vectors.h"
#include "Astrings.h"
#include "inherit/cat+mouse/positions.h"

class Animal {
friend class Scene;
public:
   Animal (string s, Position p);
   string Name () const;
   Position Pos () const;
   // Move the animal, and return true if it catches
   // its target.
   virtual bool Chase () = 0;
protected:
   string name; // animal's name
   Position pos; // animal's position
   Animal *target; // ptr to animal being chased; 0 if none
};

class Scene {
public:
   Scene ();
   int Length ();
   Animal* &operator[] (int k);
private:
   Vector<Animal*> allAnimals;
};
#endif
```

park.cc

```
#include "vectors.h"
#include "Astrings.h"
#include "inherit/cat+mouse/positions.h"
#include "inherit/cat+mouse/park.h"
#include "animals.h"

const int NUMANIMALS = 3;

// Set up the scene of animals.
Scene::Scene (): allAnimals (NUMANIMALS) {
   string response1, response2;
   float r, thetaInRadians;
   int k;
   for (k=0; k<NUMANIMALS; k++) {
      cout << "What kind of animal should be next? ";
      cin >> response1;
      cout << "What is its name? ";
      cin >> response2;
      cout << "What is its starting position?" << endl
           << "  radius = ";
      cin >> r;
      cout << "  angle in radians = ";
      cin >> thetaInRadians;
      Position coords (r, thetaInRadians);
      if (response1 == "mouse") {
         allAnimals[k] = new Mouse (response2, coords);
      }else if (response1 == "cat") {
         allAnimals[k] = new Cat (response2, coords);
      }else if (response1 == "person") {
         allAnimals[k] = new Person (response2, coords);
      }else {
         cout << "I don't know what that is." << endl;
         exit (1);
      }
   }
   // Define all the targets for the animals.
   for (k=0; k<NUMANIMALS; k++) {
      cout << "Who is " << allAnimals[k]->name << " chasing? ";
      cin >> response1;
      for (int k2=0; k2<NUMANIMALS; k2++) {
         if (response1 == allAnimals[k2]->name) {
            if (k2 == k) {
               cout << allAnimals[k]->name << " can't chase itself!" << endl;
               exit (1);
            }else {
               allAnimals[k]->target = allAnimals[k2];
               break;
            }
         }
      }
      if (allAnimals[k]->target == 0) {
         cout << "*** not chasing anyone." << endl;
      }
   }
}
```

```
// Return the number of animals in the scene.
int Scene::Length () {
   return allAnimals.Length ();
}

// Index the scene.
Animal* &Scene::operator[] (int k) {
   return allAnimals[k];
}

// Constructor (s is the animal's name,
// coords is its starting position).
Animal::Animal (string s, Position coords) {
   name = s;
   pos = coords;
   target = 0;
}

// Access functions.

string Animal::Name () const {
   return name;
}

Position Animal::Pos () const {
   return pos;
}
```

positions.h

```
#ifndef POSITIONS_H
#define POSITIONS_H

class Position {
public:
   // Initialize a position.
   Position ();                 // r = 1, angle = 0.
   Position (float r);          // angle = 0.
   Position (float r, float thetaInRadians);

   // Reinitialize a position.
   void SetAbsolutePosition (float r, float thetaInRadians);

   // Change the current position, incrementing the radius by
   // rChange and incrementing the angle by distChange.
   // Negative radius values represent movement toward the
   // statue. Positive distance changes represent
   // counterclockwise motion; negative values are clockwise.
   void IncrementPosition (float rChange, float distChange);

   // Compare two positions.
   bool operator== (Position coords);

   // Print a position.
   void Print ();
   friend ostream& operator<< (ostream &out, Position &pos);

   // Return true if someone at the position can see someone
```

```
   // or something at the argument position (i.e. the statue
   // does not block one's view), and return false otherwise.
   bool Sees (Position pos);

   // Return true if the position is at the base of the statue,
   // i.e. its radius = 1, and return false otherwise.
   bool IsAtStatue ();

   // Return true if the position is between the first argument
   // position and the second.  Precondition: the difference
   // between the first and second argument positions is less
   // than pi radians, and the radii of all the positions
   // are the same.
   bool IsBetween (Position old, Position current);

private:
   float radius;
   float angleInRadians;
   float Normalize (float radians);
};

#endif
```

Java applet version
M. Clancy, U.C. Berkeley

Background reading

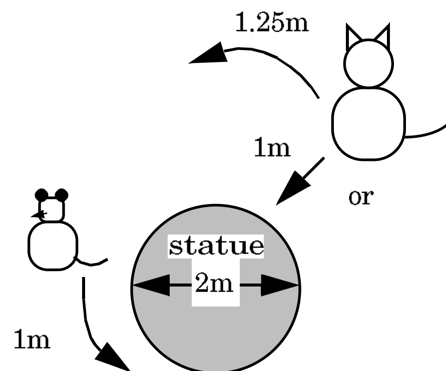
Java for Students, chapters 8, 9, and 12.

Background

The scene is an urban park; a cat watches a mouse run around the base of a statue of the first computer. Over the course of a minute, the somewhat witless mouse moves one meter counterclockwise around the statue's base, which is circular and two meters in diameter. Every sixty seconds, 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.

The cat plans eventually to get close enough to the mouse to make a juicy lunch of it. Should the chase go on more than 30 minutes, however, the cat will get tired and wander off.



The Cat and the Mouse

Problem

Complete an applet to simulate this situation and determine if the cat catches the mouse. The applet instantiates a Cat object and a Mouse object, each having a move method, and sets up a button that when clicked executes the next step of the simulation (either a mouse move or a cat move). Clicking the button once the cat has caught the mouse or abandoned the chase (after 30 simulated minutes) merely causes an error message to be displayed.

Program files

Applet framework files are in the directory ~cs9g/cat+mouse. They include the files CatMouseChase.fw.java, Position.fw.java, Cat.fw.java, and Mouse.java described below. ("fw" stands for "framework".)

- CatMouseChase.fw.java supplies the paint and init methods for you, along with a couple of auxiliary functions. The init method retrieves initial position values for

the cat and the mouse from the html file used to start the applet and initializes a cat and a mouse appropriately. It also sets up a status string that represents the progress of the simulation. The paint method draws a simplified representation of the cat, the mouse, and the statue, and also displays the cat and mouse positions and the status string.

- Position.java defines the Position class. Positions are represented in polar coordinates. Each Position object consists of a double radius, and a double angle; all these are private, and the Position class should not provide direct access to them. Methods provided for you in the Position class are several constructors, toString (to allow positions to be printed), and xCoord and yCoord methods that convert "real-world" coordinates to window pixel coordinates. You are to supply an update method to increment the position's radius or angle. You will also need several comparison methods to allow a user to determine if a position is at the base of the statue, if it is at the same coordinate as another position, if its angle is between two other angles, and if a position on the base of the statue is in its line of sight.
- Cat.fw.java and Mouse.java define the Cat and Mouse methods, respectively. Both have a move method and a constructor. Neither class is allowed direct access to the components of a Position. The complete Mouse class is provided for you; you must provide the move method of the Cat class. The Mouse move method doesn't return anything. The Cat move method returns true if the cat eats the mouse in the move and returns false otherwise.

Don't change the code provided for you; merely add methods as specified above.

Applet input

You provide initial positions for the cat and mouse in the html file that starts the simulation, using the param construct; its format is

```
<param name="parameter_name " value=parameter_value >
```

To supply cat and mouse positions, use parameter names "mouseangle", "catradius", and "catangle" whose values are expressed in radians. For example, an html file that starts the cat on the x axis 3 meters from the statue and the mouse on approximately the opposite side of the statue contains the specification

```
<applet code="CatMouseChase.class"
 width=600 height=500>
 <param name="mouseangle" value=3.14>
 <param name="catradius" value=4.0>
 <param name="catangle" value=0.0>
 </applet>
```

Supply sufficient html files to test your applet thoroughly, making sure that every statement in your code has been exercised at least once. Bring listings of these files to the Self-Paced Center when you have your applet graded.

Trigonometric information

- The cat sees the mouse if

  (cat radius) * cos (cat angle – mouse angle)

  is at least 1.0.
- When the cat circles distance d around the statue, its radius does not change, and the change in its angle can be calculated from the relationship

  d = angle * (radius of arc)
- The cat catches the mouse when it (the cat) moves past the mouse while at the base of the statue, i.e. when the cat radius is 1.0 and the mouse angle lies between the old cat angle and the new cat angle. An angle B is between angles A and C in the following circumstances:

  cos (B – A) > cos (C – A), and

  cos (C – B) > cos (C – A).

  The difference C – A is assumed to be less than 90°, or $\pi/2$ radians.
- Note that the cat cannot move inside the statue's base; hence if the cat is, say, at radius 1.7 and sees the mouse, it can move in only 0.7 meters, up to the base of the statue.
- Remember that angles a, a + 2π, a + 4π, ... are all equal.

Suggested approach

Your first task should be a design task: outline the cat's move algorithm, then figure out what methods the Position class should provide. Then start on the coding and the testing.

You should especially test the Position methods in isolation. To simplify this, set up an applet with a init method that tests the Position methods and outputs results with drawString.

Be sure to check that the output makes sense for all your tests; a student is sometimes embarrassed to have someone point out that his or her program has the cat moving into and through the statue, or moving away from the mouse instead of approaching it.

Checklist

Correctly working code:
- completion of the Cat move method;
- completion of the Position class;
- completion of the CatMouseChase actionPerformed method.

Sufficient testing:
- applet to test the Position methods;
- an init method in that applet that supplies sufficiently comprehensive tests;
- html files that exercise all statements in the remainder of the applet.

Good style:
- appropriate use of indenting and white space;
- variable and method names that reflect their use;
- informative comments at the head of each method;
- clean case analysis.

CatMouseChase.fw.java

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class CatMouseChase extends Applet implements ActionListener {

   public void init ( ) {
      super.init ( );
      nextStep = new Button ("Move once");
      add (nextStep);
      nextStep.addActionListener (this);

      ourWindowWidth = getIntegerParam ("width");
      ourWindowHeight = getIntegerParam ("height");
      myXCenter = ourWindowWidth/2;
      myYCenter = ourWindowHeight/2;
      int smallestDimension
         = ourWindowWidth < ourWindowHeight? ourWindowWidth: ourWindowHeight;

      double mouseAngle = getDoubleParam ("mouseangle");
      double catRadius = getDoubleParam ("catradius");
      double catAngle = getDoubleParam ("catangle");
      myUnit = (smallestDimension/2 - BORDER)/((int) (catRadius+1.0));

      myCat = new Cat (new Position (catRadius, catAngle));
      myMouse = new Mouse (new Position (1.0, mouseAngle));
      statusString = "Chase is in progress.";
   }

   private int getIntegerParam (String name) {
      String value = this.getParameter (name);
      if (value == "") {
         return 0;
      }
      try {
         return Integer.parseInt (value);
      } catch (Exception e) {
         return 0;
      }
   }

   private double getDoubleParam (String name) {
      String value = this.getParameter (name);
      if (value == "") {
         return 0.0;
      }
      try {
         return Double.valueOf (value).doubleValue ( );
      } catch (Exception e) {
         return 0.0;
      }
   }

   public void actionPerformed (ActionEvent event) {
      // You fill this in.
```

```
   }

   public void paint (Graphics g) {
      // The magic numbers used here should really depend on the height of the
      // type face used to display the window annotations.
      // We'll fix that for next semester.
      g.drawOval (myXCenter+5-myUnit, myYCenter+5-myUnit, 2*myUnit-10,
2*myUnit-10);
      g.drawString ("C",
         myCat.getPosition ( ).xCoord (myXCenter, myUnit)-5,
         myCat.getPosition ( ).yCoord (myYCenter, myUnit)+5 );
      g.drawString ("M",
         myMouse.getPosition ( ).xCoord (myXCenter, myUnit)-5,
         myMouse.getPosition ( ).yCoord (myYCenter, myUnit)+5 );
      g.drawString ("Mouse at " + myMouse.getPosition ( )
         + "; cat at " + myCat.getPosition ( ), 50, ourWindowHeight-30);
      g.drawString (statusString, 50, ourWindowHeight-15);
   }

   private Cat myCat;
   private Mouse myMouse;
   private boolean chaseIsOver = false;
   private int timeElapsed;
   private final int TIME_LIMIT = 30;

   private Button nextStep;
   private String statusString = "Not yet started.";
   private final int BORDER = 50; // margin for buttons and status strings

   private static int ourWindowWidth, ourWindowHeight;
   private int myXCenter, myYCenter, myUnit;
}
```

Position.fw.java

```
public class Position {

   // Represent a position (radius, angle) in polar coordinates.
   // All angles are in radians.
   // The internal representation of an angle is always at least 0
   // and less than 2 * PI.  Also, the radius is always at least 1.

   public Position ( ) {
      myRadius = 0;
      myAngle = 0;
   }

   public Position (Position p) {
      myRadius = p.myRadius;
      myAngle = p.myAngle;
   }

   public Position (double r, double theta) {
      myRadius = r;
      myAngle = theta;
   }

   public int xCoord (int xCenter, int unit) {
```

```
      return xCenter + (int) (myRadius * Math.cos (myAngle) * unit);
   }

   public int yCoord (int yCenter, int unit) {
      return yCenter - (int) (myRadius * Math.sin (myAngle) * unit);
   }

   public String toString ( ) {
      return "(" + myRadius + "," + myAngle + ")";
   }

   // Update the current position according to the given increments.
   // Preconditions: thetaChange is less than 2 * PI and greater than -2 * PI;
   //   one of rChange and thetaChange is 0.
   public void update (double rChange, double thetaChange) {
      // You fill this in.
   }

   // You will also need other methods.

   private double myRadius;
   private double myAngle;
}
```

Cat.fw.java

```
public class Cat {

   public Cat ( ) {
      myPosition = new Position ( );
   }

   public Cat (Position p) {
      myPosition = p;
   }

   public Position getPosition ( ) {
      return myPosition;
   }

   public boolean move (Position mousePosition) {
      // You fill this in.
   }

   private Position myPosition;
}
```

Mouse.java

```
public class Mouse {

   public Mouse ( ) {
      myPosition = new Position ( );
   }

   public Mouse (Position p) {
      myPosition = p;
   }
```

```
   public Position getPosition ( ) {
      return myPosition;
   }

   public void move ( ) {
      myPosition.update (0.0, 1.0);
   }

   private Position myPosition;
}
```