

Decision Logic: if, if...else..., switch..., Boolean conditions and variables

This programming assignment uses many of the ideas presented in sections 3 through 5 of the Dale/Weems text and the lecture notes covered through June 8, so you are advised to read those carefully. Read and follow the following program specification carefully.

Your score on this project will be a weighted average of the score you receive for runtime testing and the score you receive for following the instructions in the Programming Standards section below.

The Problem:

Cat and Mouse

A cat and mouse are placed into a confined space (see Project Appendix for unnecessary details) that is divided into a rectangular grid of cells; for example:

	1	2	3	4	5	6	7	8	9
1									
2									
3							M		
4									
5									
6		C							
7									
8									
9									

Each cell is labeled by a pair of coordinates (row, column). In the picture above, the mouse M is in cell (3, 7) and the cat C is in cell (6, 2). The cat and mouse will play a game. Each may move about the grid, but may not leave the grid. If the cat and mouse reach the same cell at the same time, then the mouse is caught and the game is over.

You will write a program to track the movements of the cat and mouse, given information about the movements each makes and following the rules given in the rest of this specification.

Each player (the cat or the mouse) moves along the rows and/or columns of the grid. A move is specified by giving a pair of numbers, like (3, -1), that specify how many cells the player should move vertically (first number) and horizontally (second number). For example, if the cat made the move (3, -1) from its location shown above, it would wind up in the cell (9, 1).

Neither player is allowed to move to a location outside the grid of cells. If such a move is attempted, the player will simply “wrap around” to the opposite side of the grid. That’s because we really have a Klingon cat and a Ferengi mouse playing this game on the inside of a torus (doughnut-shaped tube). The illustration above shows the tube sliced and flattened.

For example, if the mouse attempted the move (-4, 2), from its starting position above, it would wind up at cell (8, 9). If the cat attempted the move (5, -2), from its starting position above, it would wind up at cell (2, 9).

The players move along the rows and columns, and distances are calculated accordingly. Given the initial positions shown above, the distance between the cat and mouse is 8. Distances are never negative. If the cat made the move (3, -1) from its initial position at (6, 2), it would have traveled a total distance of 4. Note that the initial position is **not** counted as a move from (0, 0).

Input file description and sample:

Your program **must** read its input from a file named `ChaseData.txt` — use of another input file name will result in a score of zero. The first line of the input file specifies the number of rows and columns in the grid used for this game. These values will be positive integers and never more than 99. Rows and columns are numbered starting at 1. Each remaining line of the input file will fit exactly one of the following formats:

- `M <integer> <integer> // initial position or movement for the mouse`
- `C <integer> <integer> // initial position or movement for the cat`
- `P // print current data, as specified below`

The values on each line will be separated by whitespace. You may assume that all the input values will be logically correct. For instance:

28	24		
M	21	13	
M	-11	9	
P			
M	-6	4	
P			
M	5	-9	
C	27	8	
M	4	-8	
P			
M	10	-7	
P			
C	12	-11	
P			
C	-3	9	
M	7	6	
P			
M	9	2	
C	-9	-10	
P			
M	-9	-2	
P			
C	-13	7	
P			

Note that you must **not make any assumptions** about the number of lines of data in the input file. Your program must be written so that it will detect when it's out of input and terminate correctly, just as in Project 2.

What to calculate:

You will write a program to read the input file and use the given values to:

- track the current location of the cat and the mouse
- track the distance currently separating them
- detect when (if) the cat catches the mouse
- compute the total distance traveled by each, where the distance is measured along the grid edges

You must also watch for moves that would take the cat or mouse across one of the boundaries of the region. If such a move is attempted then the cat or mouse should “wrap around” the boundary as described on the first page of this specification.

Output description and sample:

The first line of your output should include your name only; the second line should include the title “Cat and Mouse” only; the third line should be blank; the fourth line should display the column labels shown below; the fifth line should consist of some delimiting symbol to separate the column labels from the body of the table.

Next your output file will contain a table, with one line of output for each print command processed from the input file. Each line of the table should list the location of the cat and mouse at the time the print command was read, and the distance between them at that time (provided both are “in play”). There should be a line of delimiters immediately after the last line of the table body.

The next two lines of the output file should be blank. The next line should contain the indicated labels, and the line after that should contain the total distances traveled by the cat and mouse.

Following that, there should be one blank line. If the cat caught the mouse in this game, then print the message “Mouse caught at:” followed by the location at which the capture occurred. If the cat didn’t catch the mouse, then print the message “Mouse evaded Cat”.

Your program must write output data to a file named `PursuitLog.txt` — use of any other output file name will result in a score of zero. The sample output file shown below corresponds to the input data given above:

Cat and Mouse		
Cat	Mouse	Distance
(?, ?)	(10, 22)	
(?, ?)	(4, 2)	
(27, 8)	(13, 9)	15
(27, 8)	(23, 2)	10
(11, 21)	(23, 2)	31
(8, 6)	(2, 8)	8
(27, 20)	(11, 10)	26
(27, 20)	(2, 8)	37
(14, 3)	(2, 8)	17

Distance	Mouse	Cat
	108	74
Mouse evaded Cat		

You are not required to use this exact horizontal spacing, but your output must satisfy the following requirements:

- You must use variables of type `int` for all the locations, distances, etc. If you use decimal values, your answers may be incorrect due to numerical roundoff.
- You must use the specified header and column labels, and include your name in the first line as shown.
- You must arrange your output in neatly aligned columns, with a label identifying the contents of each column. Note that while the Curator doesn’t deduct points for horizontal alignment, the TA who grades your source code for programming standards may do so.
- You must use the same ordering of the columns as shown here.
- You must print a newline at the end of each line, including the last line.
- You must print each coordinate in two columns as shown; no coordinate will require more than two.

Programming Standards:

You'll be expected to observe good programming/documentation standards. All the discussions in class about formatting, structure, and commenting your code will be enforced. See the *Programming Standards* page on the course website. Some specific requirements follow.

Documentation:

- You must include header comments specifying the compiler and operating system used and the date completed.
- Your header comment must describe what your program does; don't just plagiarize language from this spec.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Precede every major block of your code with a comment explaining its purpose. You don't have to describe how it works unless you do something so sneaky it deserves special recognition.
- If you write user-defined functions, each must have a valid C++ prototype and the definition of each must be preceded by a block comment, explaining in one sentence what the function does, and listing each parameter to the function and explaining its purpose.
- You must use indentation and blank lines to make control structures like loops and if-else statements more readable. The code from Project 1 is a good guide for how to handle some of the file I/O issues.

Implementation:

- Use named constants instead of variables where appropriate.
- Use at least one `bool` variable.
- Use at least one `if` statement, at least one `if...else` statement, and at least one `switch` statement.
- Choose your control structures appropriately.

Your submission that receives the highest score will be graded for adherence to these requirements, whether it is your last submission or not. If two or more of your submissions are tied for highest, the earliest of those will be graded. Therefore: implement and comment your C++ source code with these requirements in mind from the beginning rather than planning to clean up and add comments later.

Incremental Development:

You'll find that it's easier and faster to produce a working program by practicing incremental development. In other words, don't try to solve the entire problem at once. First, develop your design. When the time comes to implement your design, do it piece by piece. Here's a suggested implementation strategy for this project. As usual, verify and correct as necessary after each addition to your implementation.

- First, write the code necessary to read the entire input file. This should be similar to the input code used in the example payroll program. To test your work, include code to write what you're reading (and nothing else) to the output file.
- Second, add the code to process the moves and print the locations after each move.
- Third, add the code to determine if the cat has caught the mouse and react accordingly.
- Fourth, write the code to compute the total distance traveled by each player and print that along with the locations.
- Finally, implement processing of print commands and print locations only when specified.

Now you have a substantially complete program. At this point, you should clean up your code, eliminating any unnecessary instructions and fine-tuning the documentation you already wrote. Check your implementation and output again to be sure that you've followed all the specifications given for this project, especially those in the Programming Standards section above. At this point, you're ready to submit your solution to the Curator.

Testing:

At minimum, you should be certain that your program produces the output given above when you use the given input file. However, verifying that your program produces correct results on a single test case does not constitute a satisfactory testing regimen.

Additional input/output file examples will be posted on the website. You may use those for additional testing.

You should make up and try additional input files as well; of course, you'll have to determine by hand what the correct output would be.

Submitting your program:

You will submit this assignment to the Curator System (read the *Student Guide*), and it will be graded automatically. Instructions for submitting, and a description of how the grading is done, are contained in the *Student Guide*.

You will be allowed up to ten submissions for this assignment. Use them wisely. Test your program thoroughly before submitting it. Make sure that your program produces correct results for every sample input file posted on the course website. If you do not get a perfect score, analyze the problem carefully and test your fix with the input file returned as part of the Curator e-mail message, before submitting again. The highest score you achieve will be counted.

The *Student Guide* and other pertinent information, such as the link to the proper submit page, can be found at:

<http://www.cs.vt.edu/curator/>

Honor Code:

In addition to the limits set in the posted course contract, students are expressly forbidden from providing the following information to the CS 1044 Forum, or by discussion or private e-mail to other students in this course:

- any explanation of **how** to calculate the position of a player
- any explanation of **how** to calculate the total distance traveled by a player

It **is** permissible to post an input file and the corresponding output file to the listserv. Be advised that such postings may be incorrect.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the header comment for your program:

```
// On my honor:
//
// - I have not discussed the C++ language code in my program with
//   anyone other than my instructor or the teaching assistants
//   assigned to this course.
//
// - I have not used C++ language code obtained from another student,
//   or any other unauthorized source, either modified or unmodified.
//
// - If any C++ language code or documentation used in my program
//   was obtained from another source, such as a text book or course
//   notes, that has been clearly noted with a proper citation in
//   the comments of my program.
//
// - I have not designed this program in such a way as to defeat or
//   interfere with the normal operation of the Curator System.
//
// <Student Name>
```

Failure to include this pledge in a submission is a violation of the Honor Code.