CMSC131

Grids/Matrices and Nesting of Loops

Grid/Matrix Metaphor

The following (and more) all have something in common:

- Spreadsheet
- Databases
- Graph Paper
- Computer Screen
- Command-line window or sheet of paper in a typewriter (when using a fixed-width font)

They can all be thought of in terms of rows and columns in the form of a basic two-dimensional structure, sometimes called a matrix or a grid.

Interacting with Grids/Matrices

While the metaphor or rows and columns can be applied to many things, the nature of the item can restrict how we interact with it.

- In a spreadsheet, you can enter a row/col pair and jump right to that position.
- On a computer screen, programs can logically specify an x,y coordinate to place something there, but on-screen the entire image itself is refreshed using a fixed pattern.
- With a sheet of paper in a typewriter or a command-line window, you often have to access it line by line, left to right with no way to go back.

Nested Loops

It is very common to nest **for** loops and sometimes to nest **while** loops.

- You saw an example application of nesting of loops in lab this week, and will see this type of nesting in a wide variety of contexts throughout your studies and career.

The nesting of **for** loops aligns very well with things that can be thought of as grid-like.

- When you think about going through a simple list of things, you are likely to think of a single **for** loop (from the start of the list to the end of it) but when you have rows and columns of things in a grid-like pattern, nested **for** loops allow you to systematically visit every item once.

Code

The code examples on the next few slides are in your CVS repository in the project named FootballCodeExamples, so please check that out now and run them when you are asked to on the coming slides.

What would the output be?

```
public class NestedForLoopsMath {
  public static void main(String[] args) {
    int counter = 0;
    int accumulate01 = 0;
    int accumulate02 = 0;
    for (int i=0; i<5; i++) {
      for (int j=0; j<4; j++) {
        counter++;
        accumulate01+=i;
        accumulate02+=j;
      }
    }
    System.out.println(counter+" "+accumulate01+" "+accumulate02);
  }
}
```

Write the output on a sheet of paper or type it into a simple text editor, then run the above code in Eclipse to see what the output looks like.

What would this look like?

```
public class NestedForLoopsV1 {
  public static void main(String[] args) {
    for (int row=1; row<=5; row++) {
      for (int col=1; col<=5; col++) {
        System.out.print("["+row + "," + col + "]");
      }
      System.out.println();
    }
  }
}
```

Write the output on a sheet of paper or type it into a simple text editor, then run the above code in Eclipse to see what the output looks like.

What change would this cause?

```
public class NestedForLoopsV2 {
  public static void main(String[] args) {
    for (int row=1; row<=5; row++) {
      for (int col=1; col<=row; col++) {
        System.out.print("["+row + "," + col + "]");
      }
      System.out.println();
    }
  }
}
```

Again, write the output on a sheet of paper or type it into a simple text editor, then run the above code in Eclipse to see what the output looks like.

Does the output of this "look right" to you?

```
public class NestedForLoopsV3 {
  public static void main(String[] args) {
    for (int row=1; row<=5; row++) {
      for (int col=row; col<=5; col++) {
        System.out.print("["+row + "," + col + "]");
      }
      System.out.println();
    }
  }
}
```

One more time, write the output on a sheet of paper or type it into a simple text editor, then run the above code in Eclipse to see what the output looks like.

This is visually accurate for rows and columns. How would you modify the code to accomplish it?

```
[1,1][1,2][1,3][1,4][1,5]
     [2,2][2,3][2,4][2,5]
          [3,3][3,4][3,5]
               [4,4][4,5]
                    [5,5]
```

Computer Screen Example

The display on your computing device can be thought of as an x,y-coordinate plane with (0,0) being the upper-left corner.

- If you wanted to instruct a program to draw a line from the upper-left corner to the upper-right corner, how would you approach that if the only thing you could use was some method of the form **paint(x, y, color)**? What information would you need about the screen itself?
- What if you wanted to draw a rectangle instead? A triangle?

This is a type of thing that we will explore in a coding lab and a project after the first exam, painting various patterns into a grid representing a screen.

Comparison: **while** rather than **for**

```
int row = 1;
while (row <= 5) {
   int col = 1;
   while (col <= row) {
     System.out.print(row + "," + col + "  ");
     col = col + 1;
   }
   System.out.println();
   row = row + 1;
 }
}
```

Notice that here each loop control variable needs to be dealt with in several places rather than all on one line as with the for loop. This is commonly seen as a weaker/riskier style than **for** loops in this type of context.

More to come…

The nesting of coding elements such as conditionals, iteration, etc. and the related computational thinking approaches are topics that we will explore more and more as the semester progresses, in a variety of contexts that motivate their use.