AERE 361 Lab 5, Spiraling Matrix

Jordan Reese

February,16 2018

1 Problem Statement

We were trying to create a square matrix that started at 1 and counted up a specified number squared. We had to display these numbers in a matrix that spiraled out from the center. For example
Input 5:
Output:

```
21 22 23 24 25
20  7  8  9 10
19  6  1  2 11
18  5  4  3 12
17 16 15 14 13
```

So you can see, we spiral out until we reach a square matrix of 5.

2 Design

This was quite a challenging algorithm problem, and it was even harder for someone who really hasn't had any algorithm classes or development. What I did was just break down the problem into the smallest most understandable form. The first thing I did was decided I will need to ask the user for inputs. This is simple enough just prompt the user and then register the input. My user input section looks as follows.

```
printf("Please input a integer betwixt 1 and 100: ");
scanf("%s",s);
if (!is_int(s)){
  return PrintUsage();
}
```

After my user input. I knew I needed an empty matrix so I didn't have to allocate any memory. That was just declared at the top of the code.

I know needed to fill my empty matrix in some fashion, so I needed a loop. I decide to do a while loop, so as long as a haven't reached my final number then keep looping. So now I have a counter that fills an empty matrix at random. So here lies the hardest part of the lab. I needed to spiral. I started to look at the patterns associated with the sprialing matrix. In my head I used a Cartesian like plane to justify directions. The direction are always right, down, left, up on repeat. Left and down were negative counts and after the up and down directions I had to increment the amount of terms in the corresponding row or column. This is where I decided to set up a system of if statements for each of my directions need. All I had to do was check which direction I was going and just add numbers until I had to turn again. Turning took me a bit to figure out, but I finally realized that I could increment the streak counter at the ends of my up and down if statements. And each loop I could have a variable named remain that told me how much longer I needed to stay in the current direction. Below is how the loop was set up.

```
while (n < input*input){
matrix[r][c] = ++n;
/************************************************/
if (direct_right == direction){/*start right*/
  remain--;
  if(0 == remain){
   remain = streak;
   direction = direct_down;
   r++;
  }
  else{
   c++;
  }
}/*******************************end right*/
else if (direct_down == direction){/*start down*/
  remain--;
  if(0 == remain){
   streak++;
   remain = streak;
   direction = direct_left;
   c--;
  }
  else{
   r++;
  }
}/***********************************ending down*/
else if (direct_left == direction){/*start left*/
  remain--;
  if(0 == remain){
   remain = streak;
   direction = direct_up;
   r--;
  }
  else{
   c--;
  }
```

```
   }/**********************************ending left*/
   else if (direct_up == direction){/*start up*/
     remain--;
     if(0 == remain){
      streak++;
      remain = streak;
      direction = direct_right;
      c++;
     }
     else{
      r--;
     }
   }/*********************************ending up*/
 }
```

After the matrix forming all I had to do was print the matrix. Well printing the matrix is fairly easy, but C doesn't print matrices as one entity so I had to loop through each line. So it printed but the spacing was terrible. I knew there were space modifiers for numbers in C so I created a set of if statements that checked to see how long the largest number was and just set the matrix spaces to that many spaces. That code can be seen below.

```
 sprintf(squarestr,"%d",input*input);
 maxwidth = strlen(squarestr);
 /************adjusting spacing*****************/
 if(1 == maxwidth){
   format = "%1d ";
 }
 if(2 == maxwidth){
   format = "%2d ";
 }
 if(3 == maxwidth){
   format = "%3d ";
 }
 if(4 == maxwidth){
   format = "%4d ";
 }
 if(5 == maxwidth){
   format = "%5d ";
 }
 /************ending adjusting*****************/
 for(r = 0; r<input; r++){
   for (c = 0; c<input; c++){
     printf(format,matrix[r][c]);
   }
   printf("\n");
 }
```

That is the logical reasoning behind the way I set up my code.

3 Loop Invariants

I only had one major loop and two for loops for the printing of the matrix. The loop invariants of my while loop were each of my if statements, they didn't change although they edited some values as the loop ran. The if statements never changed. The matrix also always put in the value of the loop count, so that never changed. As for my for loops at the bottom of the code only one thing stayed constant and that was through each loop I added a new line to compensate for C not being able to print matrices as a single unit.

4 Complexity

The complexity of my matrix calculation was N squared where N was the input of the user. This is because the while loop had to create a square matrix. The for loops at the bottom had a complexity of of N squared as well because they had to print N rows by N columns.

5 Conclusion

Overall I am very happy with the way this lab turned out. It was problem in algorithms and not really code typing and syntax. Once you thought through the entire process the rest was pretty straightforward. I hope to do more codes like this in the future as I think they help us learn more than just doing single exercises.