Problem 1

a) Write a program that can evaluate the numbers $c_{n,k}$ in Pascal's triangle using a recurrence relation. Define $c_{n,0} = 1$ for all $n \geq 0$ and $c_{0,k} = 0$ for $k \geq 0$. Then the entries for $n+1$ and be computed from the entries for n using

$$c_{n+1,k} = c_{n,k} + c_{n,k-1}.$$

Print a table of entries in Pascal's triangle for $0 \leq n \leq 7$ and $0 \leq k \leq 7$. You should find that $c_{n,k} = 0$ for $k > n$, and those terms can be left blank in your table.

b) Extend your program so that it prints ASCII art, displaying a "." character for each even number and a "#" character for each odd number. Hence the first few lines would be

```
#
##
#.#
####
```

Using your program, extend this output to $n = 31$.

a) Originally, I implemented the recurrence relation in a function (`PascalTriangle_Coefficients(int n, int k)`) that called itself. I then called this function in `main()`. While this worked, I realized that it was calculating the same values over and over again which made it very slow. Instead wrote a function (`Generate_Pascal_Triangle(int numRows)`) that output a vector of vectors that stored the values as they were calculated. This made it much faster. I also added a function to print the triangle in a nice format. Below is the output of the program when I input 7 rows.

```
1: 1
2: 1 1
3: 1 2 1
4: 1 3 3 1
5: 1 4 6 4 1
6: 1 5 10 10 5 1
7: 1 6 15 20 15 6 1
```

b) I was tasked with outputting a table of for Pascal's triangle where each even number was replaced by a "." and each odd number was replaced by a "#". I modified my previous program to do this. Below is the output of the program when I input 31 rows.

```
1:
2:   #
3:   #.
4:   ###
5:   #...
6:   ##..#
7:   #.#.#.
8:   #######
9:   #.......
10:   ##......#
11:   #.#.....#.
12:   ####....###
13:   #...#...#...
14:   ##..##..##..#
15:   #.#.#.#.#.#.#.
16:   ###############
17:   #...............
18:   ##..............#
19:   #.#.............#.
20:   ####............###
21:   #...#...........#...
22:   ##..##..........##..#
23:   #.#.#.#.........#.#.#.
24:   ########........#######
25:   #.......#.......#.......
26:   ##......##......##......#
27:   #.#.....#.#.....#.#.....#.
28:   ####....####....####....###
29:   #...#...#...#...#...#...#...
30:   ##..##..##..##..##..##..##..#
31:   #.#.#.#.#.#.#.#.#.#.#.#.#.#.#.
32:   ###############################
```

Problem 2

a) Write a function that takes in two arrays and preforms matrix multiplication.
Your function's signature should be:

```
void mat_mul(const double* A, const double* B, double* C, int m, int n, int p);
```

The function should compute $C = AB$, where $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times p}$, and $C \in \mathbb{R}^{m\times p}$. Test your program on the matrices

$$A = \begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{pmatrix},$$

which can be generated using the provided `gradient_matrix` function.

b) **Optional.** Measure the time $T(m)$ to generate two random matrices $A, B \in \mathbb{R}^{m\times m}$ and multiply them together using your routine. Calculate $T(m)$ for $m = 100, 200, 300, \ldots, 1600$. Use linear regression on fit the data to

$$T(m) = Cm^{\alpha}$$

for constants C and α, comment on whether the value α is consistent with your matrix multiplication algorithm.