Note(s): Use of Generative AI was used for styling images in the LaTeX document. I was unsure of what was wanted as answers to the questions so if the following is not what was intended please let me know and I will redo the assignment.

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

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

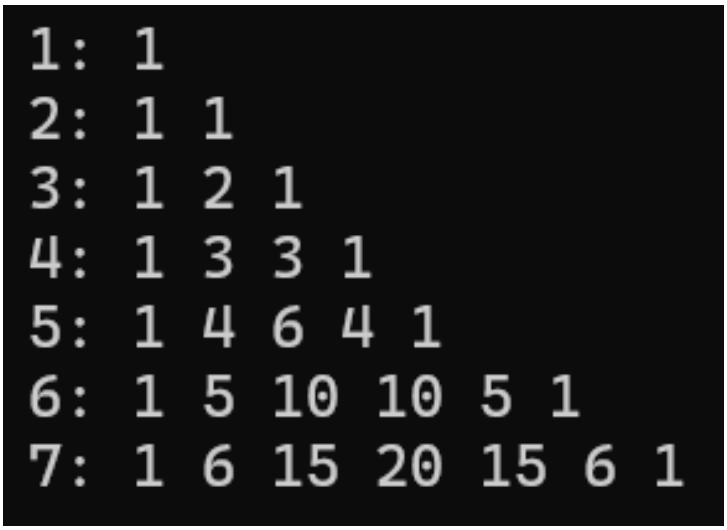
Print a table of entries in Pascal's triangle for $0 \leq n \leq 7$ and $0 \leq k \leq 7$. You should find that $c_{n,k} = 0$ for $k > n$, and those terms can be left blank in your table.

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

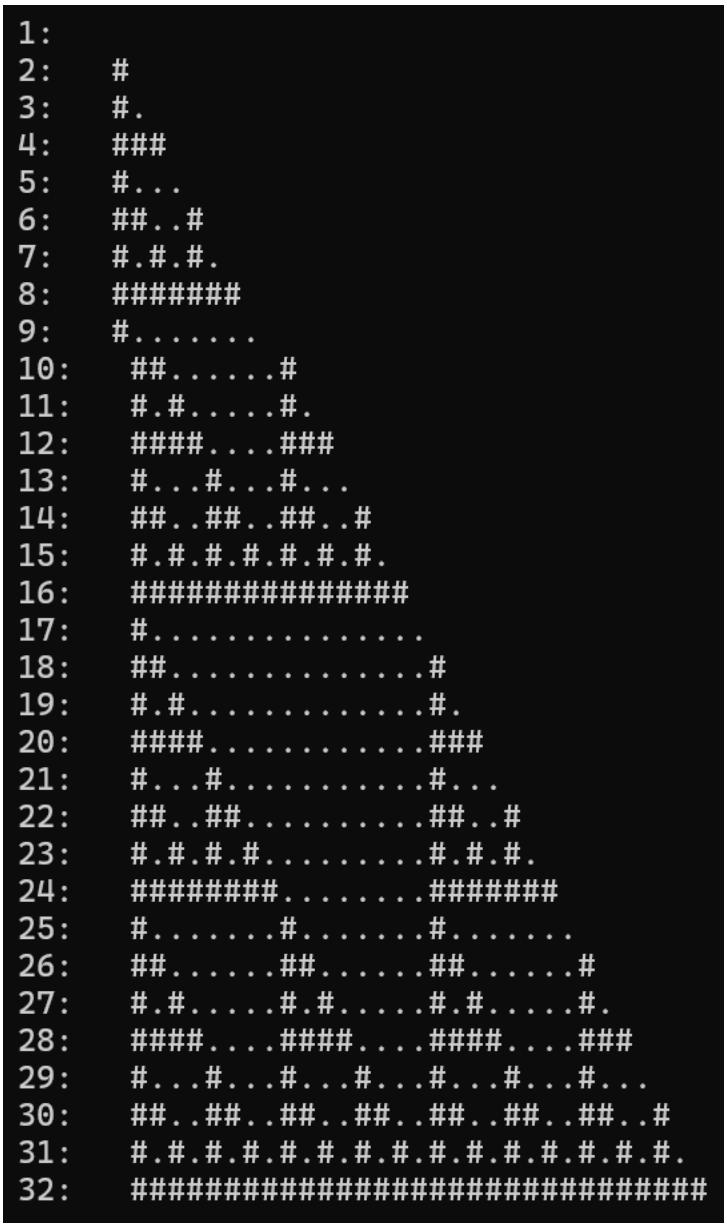
```
#
##
#.#
####
```

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.



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. The output reminded me of the Serpinski triangle. Below is the output of the program when I input 31 rows.



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

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

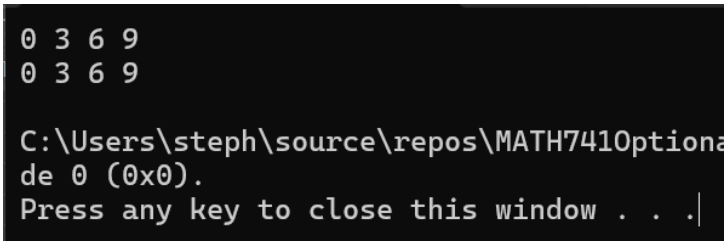
which can be generated using the provided `gradient_matrix` function.

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.

a) I created the function `mat_mul` that takes in two matrices and performs matrix multiplication. I also created a function `print_matrix` to print the resulting matrix in a nice format. Below is the output of the program when I input the matrices given in the problem statement.



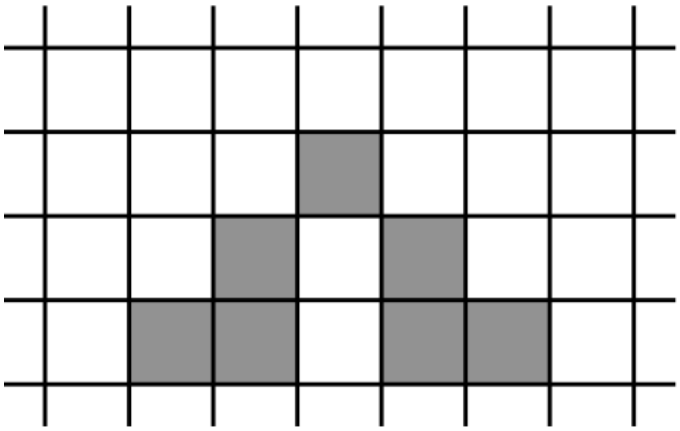
b) I did not do the optional part of this problem as I ran out of time.

Problem 3

Consider Conway's Game of Life on a regular periodic $m \times n$ grid, with cells indexed as (i, j) where $i \in \{0, 1, \ldots, m-1\}$ and $j \in \{0, 1, \ldots, n-1\}$. Each cell can either be alive or dead. Each cell has eight neighbors (the cells directly adjacent to it, including diagonals) — This is also known as the 9-cell (Moore) neighborhood. Define $N_{i,j}$ to be the number alive neighbors of (i, j). Hence $0 \leq N_{i,j} \leq 8$.
In one generation, all cells are simultaneously updated according to the following rules:

- An alive cell at (i, j) remains alive if $N_{i,j} \in \{2, 3\}$, and dies otherwise.
- A dead cell at (i, j) becomes alive if $N_{i,j} = 3$.
- All other cells remain unchanged.

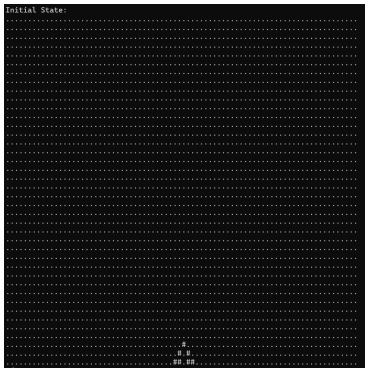
a) Implement this automaton on a grid of size $(m, n) = (80, 40)$. Initialize the grid as empty except for the following configuration, where gray indicates alive cells:



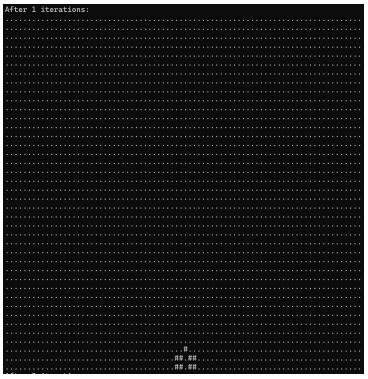
Since periodic boundary conditions are used, the exact placement of the pattern in the grid is not important. Print a snapshow of the intitial condition, using ASCII art or another method of your choice. Run the system for 200 generations, and rpint snapshots after 1, 2, 4, 50, 100, and 200 generations.

b) **Optional.** Experiment with other known patterns in the Game of Life. Or create your own patterns!

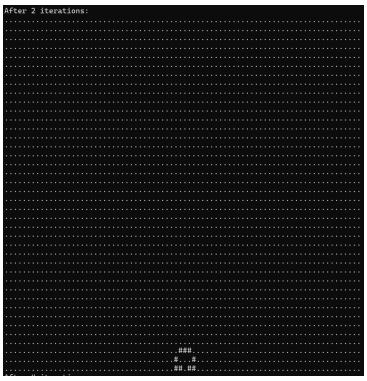
a) I implemented the Game of Life automaton on a grid of size $(m, n) = (80, 40)$. The initial configuration was set up as specified in the problem statement. Below is a snapshot of the initial condition:



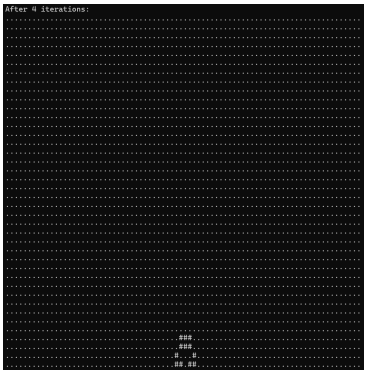
I ran the system for 200 generations and printed snapshots after 1, 2, 4, 50, 100, and 200 generations. Below are the snapshots at the specified generations:



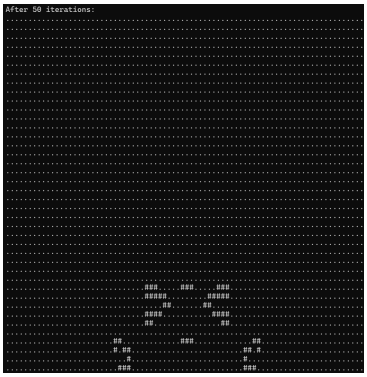
Generation 1



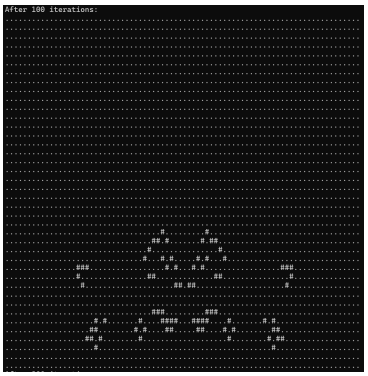
Generation 2



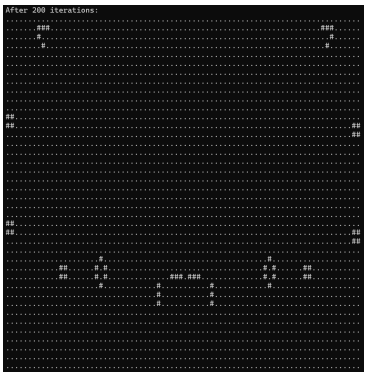
Generation 4



Generation 50



Generation 100



Generation 200