Assignment 2
CMPT 129
You may work in groups of 1 to 3 students

This assignment uses recent topics such as arrays, file input/output, functions, and function overloading. You will write a program to implement two simple automata. As examples of automata you will implement the game of life and the Sierpinski triangle. Write a C++ program to play the game of life (life.cpp) and to generate the Sierpinski triangle. For each of these examples you will break you problem into similar parts by implementing the functions specified later. You will submit a single program in a file named automata.cpp. Input and output files for one case will be posted with this assignment. Further sample cases will be provided by email shortly.

The Game of Life,

The game of life was invented by John H. Conway, is supposed to model the genetic laws for birth, survival, and death. We will play the game on a board that consists of M squares in the horizontal direction and N squares in the vertical direction (0<M<=50, 0<N<=50). Each square on the edge of the board will remain empty for the entire game. The edge of the board is defined as row 0, row N-1, column 0 and column M-1. Each square that is not on the edge of the board will be called an interior square. Each interior square can contain a space (indicating no organism is present, or can contain an X indicating that an organism is present. Each square (except for border squares) has eight neighbors. The next generation of organisms (the next turn in the game) is determined according to the following rules.

a) Birth: an organism will be born in each empty lsquare that has exactly three neighbors
b) Death: an organism with four or more organisms as neighbors will die from overcrowding. An organism with less than two neighbors will die from loneliness
c) Survival: an organism with two or three neighbors will survive to the next generation.
d) Assume that the borders (where squares have fewer than 8 neighbors) are infertile regions where organisms can neither survive nor be born. Thus, border squares will always be empty.

The inputs for your problem are as follows

1. You should use a global constant to define the maximum size of the two dimensional arrays
2. The names of the input data file and output file are supplied by the user as keyboard input.
3. The initial configuration of organisms is provided in an input data file.
   a) The first line of the input data file contains three integers. These three integers represent
      The number of rows in the game board
      The number of columns in the game board
      The number of generations to be calculated (the initial board is generation 0, the first new generation calculated is generation 1)
   b) The remainder of the game file contains an image of the game board. This image will look like a game board. A sample input file, inputFileBoard.txt is provided for you along with this problem description. Inside that file you will see that every character in the image of the game board represents one square on the board. The squares on the game board will be filled with the space and upper case X characters in the input file.
   c) The last character in the input file should be a newline (If you move to the end of your input file the cursor should be at the beginning of the line following the last piece of data)
4. Remember to consider cases of missing and corrupted data in your implemented program. Your data files are not guaranteed to be correct.

Your program must be divided into functions. You must write the following functions that are defined in detail after the description of the main program. You may add additional functions that are called by any of the

required functions. The supplied function prototypes MUST NOT be changed in any way.

```
void InitGen(char lifeBoard[][MAX_ARRAY_SIZE], int& numRowsInBoard, int& numColsInBoard,
int& generations);

void NextGen(char lifeBoard [][MAX_ARRAY_SIZE], int numRowsInBoard, int numColsInBoard
int numRowsInBoard, int numColsInBoard, int generationNum);
void PrintGen(char lifeBoard[ ][MAX_ARRAY_SIZE], ostream& outStream, int numRowsInBoard,
int numColsInBoard, int generationNum);
```

Your main program should:

1. Declare the array to hold the board (as an automatic array)
2. Use the ***InitGen*** function to initialize your newly declared array
3. Request the name of the output file,
   Open the output file and check that it was opened correctly If it did not open correctly print the following error message and terminate the program
   ERROR: output file not opened correctly
4. Use the ***PrintGen*** function to print the title and the initial configuration of the game board into the output file
5. Use the ***PrintGen*** function to print the title and the initial configuration of the game board to the screen
6. For each calculated generation requested your main program should:
   a. Use the ***NextGen*** function to calculate the locations of the organisms for the next generation.
   b. Print the results for the next generation (the one you just calculated using ***NextGen***) into the output file using the ***PrintGen*** Function.
   c. If this is the first generation calculated or the last generation calculated, print the results for the generation to the screen using the ***PrintGen*** Function.

Your function InitGen() should

1. Request the name of the input file using the prompt
   Enter the name of the input file:
2. Open the input file and check that it was opened correctly
   If it did not open correctly print the following error message and terminate the program
   ERROR: input file not opened correctly
3. Read the number of rows in the game board from the input file. Check that the number of rows was actually read. If the number of rows was read check that it was within the specified range 0<N<=50 Use the following error messages if number of rows is not read or is out of range
   ERROR: Cannot read number of rows
   ERROR: Read an illegal number of rows for the board
4. Read the number of columns in the game board from the input file. Check that the number of columns was actually read. If the number of columns was read check that it was within the specified range 0<M<=50. Use the following error messages if number of columns is not read or is out of range
   ERROR: Cannot read number of columns
   ERROR: Read an illegal number of columns for the board
5. Read the number of generations to be calculated from the input file. Check if the generations was read and if the value read was within the acceptable range. Use the following error messages if generations is not read or is out of range (<1)
   ERROR: Cannot read the number of generations
   ERROR: Read an illegal number of generations
6. Read the initial configuration of the game board from the input file into the character array **myLifeBoard[MAX_ARRAY_SIZE]][MAX_ARRAY_SIZE]**. Place the blanks or Xs, one character into each element of the 2-D array representing the life game board.

a) You should check that the input data contains only blanks and Xs. If it contains any other characters you should print an error message and terminate the program, the error message should say which element of the array was being filled when the error occurred. That is the message should be
ERROR: Input data for initial board is incorrect
Location (8, 9) is not valid
The numbers in the brackets will give the element for which the error occurred

b) You should assume that each line in the input file corresponds to one row of the game board. If there are too many characters in a given line of the input file you should print the error message
ERROR: ignoring extra characters in line I of the input array
where i is the index of the row being read. Be sure not to read the extra characters in the line into the array holding the board.

c) You should be sure you have actually read enough data (that you have not reached the end of the file) If one of your reads fails print the following error message
ERROR: Not enough characters in row i of the input array

d) You should check to make sure that all data on the edges of the game board are blanks and if any are not you should print the following error message and terminate the program.
ERROR: organisms are present in the border of the board, please correct your input file

Your ***PrintGen*** function

We have not yet discussed type ostream, it is a more general type that includes variable of types ofstream and cout and cerr.

When you call the PrintGen function you will use cout as the actual argument that corresponds to the formal argument outStream when you wish to print to the screen and you will use your output stream as the actual argument when you wish to print to your file. For example suppose you created your output stream with the statement
ofstream yourOutStream;

then, to print an array myLifeboard with 20 rows and 30 columns that shows the board for the 10th generation to the screen (cout) the call would be
PrintGen(myLifeBoard, cout, 20, 30, 10);
To print the same array myLifeboard to your output file would the call would be
PrintGen(myLifeBoard, yourOutStream, 20, 30, 10);

1. Print the game board for this iteration to ***outStream***

a) Print a header to label the game board. The header should look like
   i. ***LIFE game board generation 123***
   ii. The number will indicate which generation's game board is illustrated below the title. The example title would be for generation 123

b) Print the game board to ***outStream.*** When you print out the array print an X for each array location containing an organism, and a blank for any location not containing an organism. Print one row of the game board per line.

c) As you go also print the line and column numbers and the frame as illustrated in the sample ouput and console output files posted with this assignment.

d) Print three blank lines to ***outStream***

Your ***NextGen*** function should

1. Within the function, ***NextGen,*** a second array, ***nextGenBoard***, will be declared. It will have size ***nexGenBoard[MAX_ARRAY_SIZE][MAX_ARRAY_SIZE]***.
2. Based on the locations of the organisms in the array ***lifeBoard***, determine the locations of organisms for the next generation and place those organisms into the array ***nextGenBoard.*** To determine the locations of organisms for the next generation

a. First for each row in the ***lifeBoard*** array (except the first row and the last row)
   a. For each element in a row in the lifeBoard (except the first element and the last element)
      i. Count the number of organisms (Xs) in adjacent elements of the 2-D array. If we are considering lifeBoard[Z][Q], then the adjacent elements are lifeBoard[Z+1][Q+1], lifeBoard[Z+1][Q], lifeBoard[Z+1][Q-1], lifeBoard[Z][Q-1], lifeBoard[Z][Q+1], lifeBoard[Z-1][Q-1], lifeBoard[Z-1][Q] and lifeBoard[Z-1][Q+1]
      ii. If the element has exactly three neighbors (three adjacent elements containing organisms) then the element nextGenBoard[Z][Q] will be an X
      iii. If the element lifeBoard[Z][Q] contains an X and has two neighbors then nexGenBoard[Z][Q] will be an X
      iv. Otherwise nextGenBoard[Z][Q] will be a blank
3. Once the entire next generation is determined copy the contents of nextGenBoard into the array lifeBoard and then return to the calling program.

Remember when passing an array into a function we can change the contents of the array within the function and expect those changes to be reflected in the values of the array in the calling program. Do remember that you cannot change the contents of a variable that is not an array within your function and expect the changes to be reflected in the main program (unless it is passed by reference).

You will print the results of the life game in the formats illustrated in the supplied sample output file and console output file. The console output file contains what should be printed to the screen.

The Sierpinski Triangle

You will generate a pattern called a Sierpinski gasket on a board of N by M squares (M and N both <=50). The board will be represented by a 2-D array. You should use the exact prompts and error messages specified in the examples in your code.
Your program must be divided into functions. You must write the following functions that are defined in detail after the description of the main program. You may add additional functions that are called by any of the required functions. The supplied function prototypes MUST NOT be changed in any way. Notice that you are overloading each of the functions written for the game of life

void InitGen(int sierpBoard[][MAX_ARRAY_SIZE], int& numRowsInBoard, int& numColsInBoard, int& generations);

void NextGen(int sierp[][MAX_ARRAY_SIZE], int numRowsInBoard, int numColsInBoard);

void PrintGen(int sierpBoard[][MAX_ARRAY_SIZE], ostream& outStream, int numRowsInBoard, int numColsInBoard, int generationNum);

The following should be added to your main program for the game of life:

4. Declare the array to hold the board (as an automatic array at the start of the main program)
5. After finishing your last output for the game of life use the ***InitGen*** function for the Sierpinski triangle to initialize your newly declared array
6. Request the name of the output file for the SIerpinski triangle output
7. Open the output file and check that it was opened correctly If it did not open correctly print the following error message and terminate the program ***ERROR: output file not opened correctly***
8. Use the ***PrintGen*** function for the SIerpinski Triangle to print the title and the initial configuration of the game board into the output file
9. Use the ***PrintGen*** function to print the title and the initial configuration of the game board to the screen

10. For each calculated generation requested your main program should:
d. Use the ***NextGen*** function to calculate the locations of the organisms for the next generation.
e. Print the results for the next generation (the one you just calculated using ***NextGen***) into the output file using the ***PrintGen*** Function.
f. If this is the first generation calculated or the last generation calculated, print the results for the generation to the screen using the ***PrintGen*** Function.

Your InitGen program for the Sierpinski triangle should:

1) Prompt the user for the name of an input file.
2) Open the input file. Check that it has been opened properly
3) If the input file has not opened properly print the error used in life then terminate
4) Read the number of rows in the game board***, 0<N<50*** from the file
5) If N cannot be read or is out of range print the same error messages used for this case for the game of life
6) Read the number of columns in the game board, ***0<M<50*** from the file.
7) If M cannot be read or is out of range print the same error messages used for this case for the game of life
8) Read the number of generations from the file
9) If the number of generations cannot be read or is out of range print the same error messages used for this case for the game of life
10) Initialize ***board[2][4]=1***

Your NextGen function for the Sierpinski triangle should

1. Within the function, ***NextGen,*** a second array, ***nextGenBoard***, will be declared. It will have size ***nextGenBoard[MAX_ARRAY_SIZE][MAX_ARRAY_SIZE]***. Initialize all elements of **nextGenBoard** to 0
2. Based on the locations of the organisms in the array ***sierpBoard***, determine the locations of organisms for the next generation and place those organisms into the array ***nextGenBoard.*** To determine the locations of organisms for the next generation
    a. Every square on the top edge and the left hand edge of the board must be 0 and remain 0 for every generation.
    b. Every other square in the board (not on the top or the left hand edge) contains a 1 or a 0.
    c. If k=0 or j=0 or k=j=0 then ***nextGenBoard[k][j] = 0;***
    d. Otherwise ***nextGenBoard[k][j] = (sierpboard[k][j] + sierpboard[k-1][j] + sierpboard[k][j-1]) % 2***
3. Once the entire next generation is determined copy the contents of nextGenBoard into the array sierpBoard and then return to the calling program.

The **PrintGen** function for the Sierpinski triangle is the almost the same as the function for the game of life. The significant differences are

The type of the array differs.
For each 1 in the array print an upper case X and for each 0 print a space.