Programación Avanzada
Proyecto 1.5: El juego de la vida

En esta práctica se desarrollará una clase llamada `Life` que implemente el Juego de la Vida de John Conway[1]. La práctica requiere el uso de un arreglo bidimensional de tamaño arbitrario, el cual debe implementarse como un arreglo unidimensional cuyos elementos están en orden lexicográfico, de manera que los métodos de la clase sean capaces de manejar arreglos de cualquier tamaño.

La clase `Vida` debe contener los siguientes miembros de datos **privados**:

- `char *cell, *temp_cell, *saved_cell` - Arreglos bidimensionales de dimensiones `nrow` $\times$ `ncol` que serán reservados de manera dinámica en el constructor de la clase (y liberados en el destructor).
- `int nrow, ncol` - Las dimensiones de los arreglos.

Además, la clase debe implementar los siguientes métodos públicos:

1. Un constructor que tome `int m` e `int n` como argumentos, los cuales se usarán para inicializar el tamaño de las matrices. El constructor debe reservar memoria dinámica para las tres matrices `cell` y `temp_cell` y `saved_cell` e inicializar ambas con ceros.
2. Un destructor que libere la memoria reservada por el constructor. Tomar las medidas de seguridad necesarias para que solo se libere memoria que fué correctamente asignada durante el constructor.
3. Un método `void gen_random(double p)` que asigne a todos los elementos de `cell` valores binarios (0 ó 1) aleatorios con probabilidad `p`. Para cada celda de la matriz, obtenga un número aleatorio real entre 0 y 1; si este número es menor que `p`, entonces asignarle 1 a esa celda; de lo contrario, asignarle 0.
4. Un método `void print()` que borre la pantalla (en Windows esto puede hacer usando `system(cls)`, mientras que en Linux/OSX se utiliza `system("clear")`) y luego imprima la matriz binaria `cell`, donde los ceros se imprimirán como espacios en blanco y los unos como asteriscos.
5. Un método `int add_neighbors(int i, int j)` que calcule y devuelva la suma de los valores de los ocho vecinos (celdas mas cercanas) de la celda `i,j` de la matriz `cell`. Para las celdas que se ubican en las orillas, se pueden considerar únicamente los vecinos existentes, o bien, considerar los vecinos de manera toroidal (e.g., el vecino izquierdo de la celda `0,0` es la celda `0,n-1`).
6. Un método `int evolve()` que calcule la matriz `temp_cell` a partir de la matriz `cell` de la forma siguiente: Digamos que si un elemento vale 1, la celda correspondiente está viva. La función debe calcular $g_{i,j}$ (`temp_cell`) a partir de los ocho vecinos de $f_{i,j}$ (`cell`), considerando las siguientes reglas:

[1] Para mas información consultar: `http://es.wikipedia.org/wiki/Juego_de_la_vida`

- Si una celda viva en `f` tiene menos de dos vecinos vivos, entonces morirá en `g`.
- Si una celda viva en `f` tiene mas de tres vecinos vivos, entonces morirá en `g`.
- Una celda viva en `f` continuará en ese estado en `g` solamente si tiene dos o tres vecinos vivos.
- Una celda muerta en `f` cobrará vida en `g` si tiene exactamente tres vecinos vivos, de lo contrario continuará en ese estado.

Una vez calculados todos los elementos de `temp_cell`, el método debe copiar toda la matriz `temp_cell` en la matriz `cell`. En otras palabras, la matriz `temp_cell` solamente se utiliza de manera temporal para el cómputo de la evolución. Para realizar la copia de manera eficiente, puede utilizarse la función `memcpy()` de la librería `<cstring>`, o bien simplemente intercambiar los apuntadores `cell` y `temp_cell`.

La función `evolve()` debe devolver el número de celdas vivas que quedan en la matriz después de la evolución. Este puede obtenerse fácilmente incrementando un contador cada vez que se le asigne 1 a una celda dentro del ciclo donde se calcula `temp_cell`.

7. Usando los métodos anteriores, escriba un programa (la función `main()`) que implemente el juego de la vida: el programa iniciará mostrando una matriz binaria aleatoria, y luego, de manera iterativa, encontrará una nueva matriz realizando una evolución, la mostrará en pantalla, y esperará a que el usuario presione la tecla Enter (por ejemplo, usando `cin.get()`). La iteración continuará hasta que el usuario presione la tecla x (seguida de Enter). Utilice una malla que sea lo suficientemente grande para casi llenar la ventana de consola.

8. Además, agregue los siguientes métodos públicos a la clase (se utilizarán posteriormente en el curso):

   (a) `char get_cell(int i, int j)` - devuelve el valor de la celda (i, j) de la matriz `cell`.

   (b) `void save_state()` - copia la matriz `cell` en la matriz `saved_cell`. Ojo: hay que copiar los datos, no intercambiar los apuntadores.

   (c) `void recover_state()` - copia la matriz `saved_cell` en la matriz `cell`. Ojo: hay que copiar los datos, no intercambiar los apuntadores.