# Practica 3 Estructura de Computadores

-Javier Bautista Rosell  NIA:100315139,
Correo: 100315139@alumnos.uc3m.es

-Alejandro Blanco Samaniego  NIA:100315058,
Correo: 100315058@alumnos.uc3m.es

# Planificación

## Planificación del tiempo de trabajo y reparto de tareas:

Hemos decidido en nuestra planificación dividir los trabajos en función de tiempo estimado de trabajo y relación con él.

En el **ejercicio 1**: Alejandro Blanco Realizará las funciones Neighbor y Update diagonal. Javier Bautista realizará las funciones set, get y print. Se trabajaran 2 horas al día, desde el 20 de noviembre. Según este ajuste, terminaríamos este ejercicio el día 30 de noviembre.

| Tiempo | Alejandro Blanco | Javier Bautista |
|---|---|---|
| 0 horas | Comienza función Neighbor | Comienza función get |
| 4 horas | Función Neighbor | Fin get<br>Comienza función set |
| 6 horas | Fin Neighbor<br>Comienza Update diagonal | Función set |
| 10 horas | Update diagonal | Fin set<br>Comienza función print |
| 14 horas | Fin update diagonal<br>Comienza comprobaciones de la funcion get | Función print |
| 18 horas | Fin comprobaciones get<br>Comienza comprobaciones de las funciones set y print | Fin print<br>Comienza comprobaciones de las funciones Neighbour y update diagonal |
| 22 horas | Fin Comprobaciones | Fin comprobaciones |

En el **ejercicio 3**: Alejandro Blanco Realizará las funciones malloc y game of life. Javier Bautista realizará las funciones copy array y clean array. Se trabajará 1 hora al día desde el 2 de diciembre, 2 horas desde el 15 y 4 horas desde el 17,

reuniéndonos los últimos días durante 8 horas por día, si hiciere falta, por cualquier imprevisto. Según este ajuste, terminaríamos este ejercicio el día 17

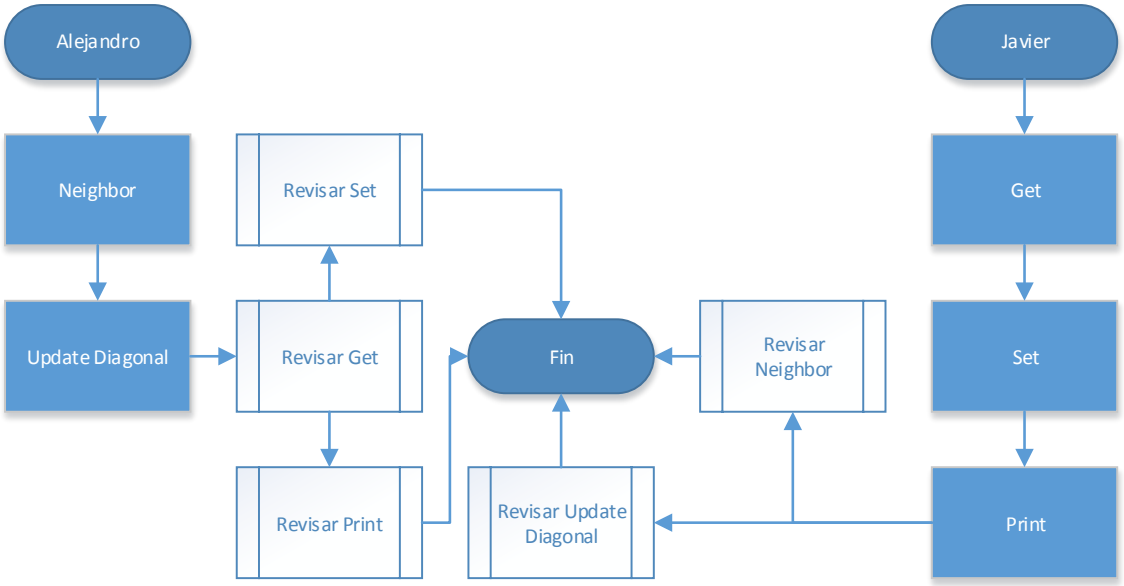
| Tiempo | Alejandro Blanco | Javier Bautista |
|---|---|---|
| 0 horas | Comienza función malloc | Comienza función copy array |
| 6 horas | Fin malloc<br>Comienza game of life | Función copy array |
| 7 horas | Game of life | Fin copy array<br>Comienza clean array |
| 14 horas | Fin game of life<br>Comienza comprobaciones de copy array | Fin clean array<br>Comienza comprobaciones de malloc |
| 17 horas | Fin comprobaciones de copy array<br>Comienza comprobaciones de la función clean array | Fin comprobaciones de malloc<br>Comienza comprobaciones de gameof life |
| 20 horas | Fin comprobaciones de clean array | Fin comprobaciones de game of life |

### Diagramas:

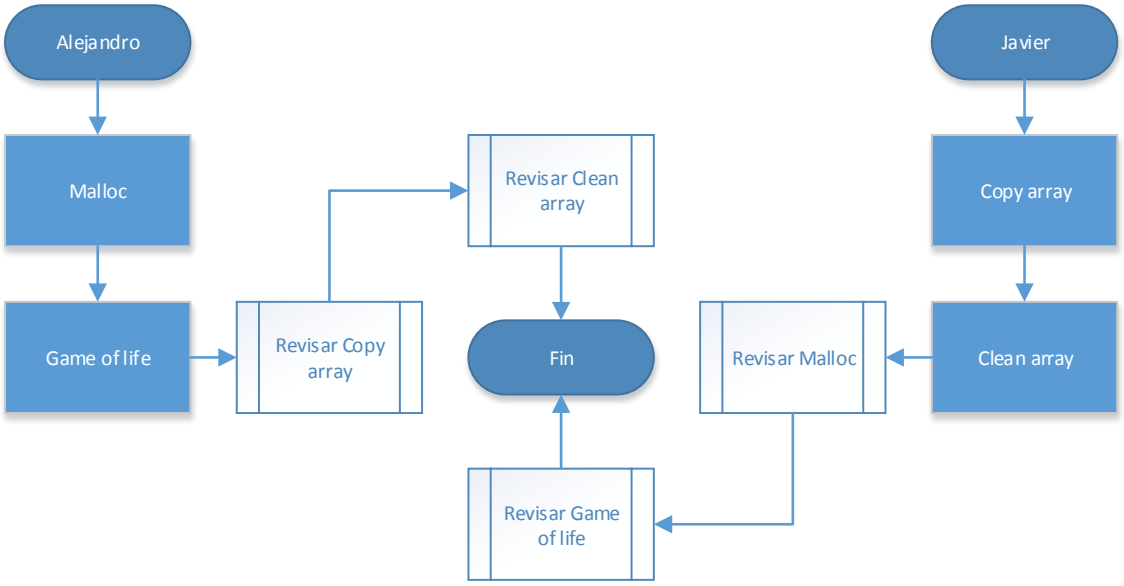
**Diagrama de estados de Software**: No se realizará, debido a que no hay interacciones en este ejercicio.

## Diagrama de flujo:

**Ejercicio 1:**




**Ejercicio 3:**




## Comunicación entre los miembros del grupo:

Nos reuniremos 1 vez a la semana, en persona, en la biblioteca de la universidad, o en la casa de Javier Bautista, hasta la finalización del primer ejercicio de la práctica. Una vez queden 2 semanas, nos veremos 2 veces por semana, una en persona, y

otra por Skipe, para mayor comodidad. La última semana nos veremos todos los días, para terminar e ejercicio. También nos vemos todos los días lectivos en clase, donde solucionaríamos, extraordinariamente, imprevistos que nos ocurran.

## Herramientas para la comunicación y el envío del trabajo:

Como herramientas usaremos Skipe y correos electrónicos, para comunicación entre nosotros y la profesora.

## Métodos de unión del trabajo:

Uniremos el trabajo del ejercicio 1 el 1 de diciembre. Ese mismo día probaremos la función con distintos valores iniciales, probando con casos extremos y asegurándonos la menor cantidad de fallos. Si hiciere falta, quedaríamos el 2 para terminar.
Uniremos el trabajo del ejercicio 3 el 17 de diciembre. Ese mismo día probaremos la función con distintos valores iniciales, probando con casos extremos y asegurándonos la menor cantidad de fallos. Si hiciere falta, quedaríamos el 18 y el 19 para terminar.

## Plan de pruebas:

Para las pruebas del ejercicio 1, los días 1 y 2 de diciembre comprobaremos que las funciones son válidas para diferentes tipos de valores en el .data, variando tamaños de x, de y, y los valores de la matriz. No probaremos con valores de x e y 0 o inferiores, al ser inútil usarlos.

Para las pruebas del ejercicio 3, os días 17, 18 y 19 de diciembre comprobaremos que las funciones son válidas para diferentes tipos de valores en el .data, variando tamaños de x, de y, y los valores de la matriz. No probaremos con valores de x e y 0 o inferiores, al ser inútil usarlos. Tampoco la matriz puede tomar valores que no sean 1 o 0.

## Evaluación del resultado de la planificación:

Nos Parece una Planificación aceptable, algo optimista conforme a la dificultad y a la duración del tercer ejercicio. No se pueden dedicar más horas por día, ya que

también tenemos que trabajar para otras asignaturas, o trabajar en empleos extrauniversitarios. Consideramos que, aunque tal vez sin pleno éxito, conseguiremos sacar adelante todas o al menos la mayoría de las funciones satisfactoriamente.

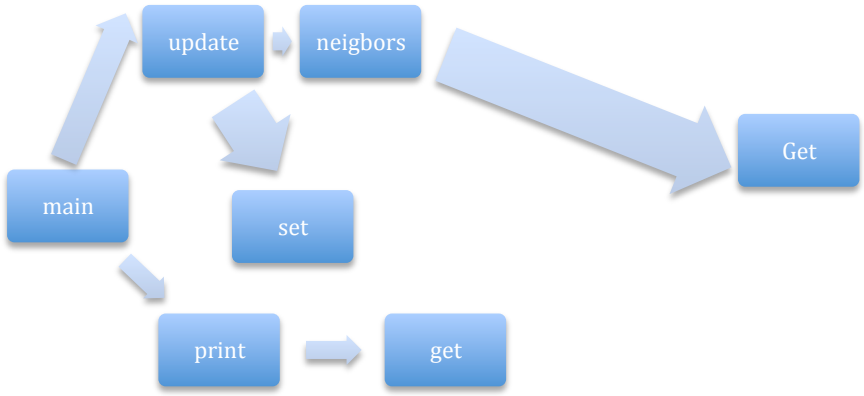
## Ejercicio 1:

Descripción del programa:

En este ejercicio se nos solicita implementar un programa que, dada una matriz origen, nos imprima otra con su diagonal principal modificada, siendo cada uno de los valores de las celdas de la diagonal principal la suma de los valores de las posiciones adyacentes a cada una.

El esquema que hemos seguido para relacionar las funciones (entrada/salida) es el siguiente:

El main necesita llamar solo al update_diagonal y al print. El update_diagonal, a su vez, debe llamar al neighbors, para la suma de las celdas, y al set, para actualizar la diagonal. El neighbors solo necesitará llamar al get, para saber qué valores sumar. El print, al igual que este último, también solo necesitará llamar al get, esta vez para saber qué valores imprimir por pantalla.




Para el desarrollo de estas funciones, nos hemos basado en los fragmentos de código en .c del enunciado.

## Main:

En el main cargamos las variables que necesitamos en el resto de funciones, y las reinicializamos para empezar cada una. Desde el main llamamos primero a la función print, que imprima la matriz original, y luego al update_diagonal, que imprima la matriz con la diagonal modificada.

## get:

La función get recoge y guarda el valor de una posición de una matriz. Para que el get pueda situarse correctamente en la matriz, se multiplica "x" por la posición de x en la que nos encontramos, y luego por 4 (.word). Luego multiplicamos la posición de "y" donde nos encontramos también, y sumamos estos dos valores obtenidos. Sumamos este nuevo valor a la dirección del array donde nos encontramos, y ya estamos en la dirección de memoria deseada. Luego cargamos el valor que guarda esta dirección de memoria en un registro, y la función devolverá ese registro.

## set:

La función set guarda un valor solicitado en una posición de una matriz. El método seguido para el posicionamiento de esta función en la matriz, es el mismo que el utilizado en el get. Luego guardamos el valor solicitado en la dirección de memoria en que nos encontremos.

## Print:

La función print se encarga de imprimir una matriz. El método que seguimos fue el siguiente: con los contadores de posición a 0, comenzamos los bucles para recorrer la matriz. En estos bucles se encuentra la función get, que se encarga de recoger el valor que se encuentra en una posición de la matriz. Una vez recogido ese valor, se imprime por pantalla (además de un espacio). Una vez terminado el bucle interior, en el exterior se realiza un salto de línea, antes de continuar con la siguiente fila. De esta manera hemos impreso por pantalla la matriz entera.

**Batería de pruebas:**

1. En la siguiente imagen vemos como la función print imprime la matriz dada en el código inicial

http://gyazo.com/70a7c4ac5273f8ac49148294019b84e9

2. En la siguiente imagen vemos como la función print imprime además una matriz 5x5

http://gyazo.com/2ed7f6ce2772c7b05bd8ec1887a73096

## Neighbors:

La función neighbors se encarga de sumar los valores de las posiciones adyacentes de una celda de una matriz dada. Siguiendo la siguiente tabla, el neighbors debería retornar un 8, que es la suma de los valores de las posiciones adyacentes a la celda de origen.

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | Celda origen | 1 |
| 1 | 1 | 1 |

El método que seguimos es largo, pero sencillo. Con cada if, averiguamos si existe o no cada celda adyacente; es decir, si nos encontramos en una esquina, solo habría que sumar tres valores. Si la celda no existe, es decir, por ejemplo, nos encontramos en la banda izquierda y toca sumar el valor de la celda que se encuentra a la izquierda, omitimos ese paso al no entrar en el if. Una vez hemos entrado en uno, porque la celda exista y debe sumarse el valor que contiene, usamos el get para averiguar el valor a sumar, y lo sumamos a un registro que guarde el sumatorio ($t4 en nuestro caso es el registro que guarda el sumatorio). Una vez terminada la suma de cada celda existente adyacente, la función devuelve éste sumatorio ($t4).

### Update_diagonal:

La función update_diagonal  se encarga de modificar los valores de la diagonal principal de una matriz, siendo cada uno de los valores de las celdas de la diagonal principal la suma de los valores de las posiciones adyacentes a cada una. Esto se consigue mediante las funciones neighbors, que sumará los valores adyacentes, y set, que actualizará estos valores en la diagonal principal. Para esto usamos un bucle, y un contador de diagonal, que será el valor de contador de "x" y de "y" a la vez. En cada posición de la diagonal, es decir, para cada valor de este contador, usamos neighbors para saber la suma de las posiciones adyacentes, y set, para sustituir esa posición de la diagonal por la suma. Luego actualizamos el contador de diagonal.

**Batería de pruebas:**

1. En la siguiente imagen vemos como se imprime por pantalla la matriz original y luego la actualizada mediante el update diagonal, tal como viene en el enunciado de la práctica.

http://gyazo.com/2d503751fd9a90cac0d14330e52c06b4

2. En la siguiente imagen vemos como esto también funciona para una matriz 5x5.

http://gyazo.com/5c012f63c7248d2cbe236bda62629359

## Ejercicio 3:

Descripción del programa:

En este ejercicio, se nos solicita implementar un programa para desarrollar "El juego de la vida", con las funciones malloc, clean_array, copy_array y game_of_life; usando además las funciones generadas en el apartado anterior.

Para el desarrollo de estas funciones, nos hemos basado en los fragmentos de código en .c del enunciado.

## Malloc:

La función malloc reserva en memoria espacio para una matriz. Siguiendo el método en c del enunciado, generamos en ella primero un array, y luego, para cada hueco del array, generamos ahí otro array…es decir, creamos la matriz como un array de arrays, en lugar de crearla como un único array (x*y), que sería lo más sencillo. Esta función servirá para poder crear una matriz auxiliar con la que trabajar, y no echar a perder la original.

http://gyazo.com/7a49eeec41a47f642e1300c354c18a94

Como se puede ver, en el bucle "buclecreacion" creamos los array a partir de diferentes posiciones de un array previamente generado.

## Copy_array:

La función copy_array se encarga de copiar cada valor de una matriz en otra, de las mismas dimensiones. El método que se sigue es sencillo, primero usamos el get para tomar el valor de una posición de la matriz original a copiar en su correspondiente en la matriz de destino. Luego usamos un set para guardar ese valor obtenido con el get en la posición de la matriz de destino correspondiente. Esta función servirá para actualizar los valores (1 ó 0) de la matriz original tras cada iteración.

http://gyazo.com/b13f96a9f253b98b015d5270feaed099

Como se puede ver, en el bucle "buclecony", usamos el get para conseguir el número a guardar, y tras el correspondiente paso de parámetros usamos el set para guardar éste número en la matriz de destino.

## Clean_array:

La función clean_array se encarga de colocar en cada posición de una matriz un 0. Para esto simplemente cargamos un 0 en el set, y con el set colocamos un 0 en cada posición de la matriz. Sirve para limpiar el array auxiliar, y poder colocar ahí los nuevos valores de la siguiente iteración.

http://gyazo.com/4d96eaec4827517e16cc811602d4109e

Como se puede ver, metemos un 0 en el registro $t5, que lo pasamos al set por la pila en el bucle "bucley", y a continuación el set se encarga de poner un 0 en cada posición.

## Game_of_life:

La función game_of _life se encarga de englobar toda la práctica para simular "El juego de la vida". Primero pasamos parámetros y creamos registros que contengan valores de contadores para las iteraciones, para x y para y, además de dos registros para los valores obtenidos en el neigbors y el get (cell). Aquí además generamos con la función malloc espacio para una matriz auxiliar que nos ayudará en el proceso.

Luego entramos en los bucles. Cada vez que pasamos por el primero, el de las iteraciones, ponemos a 0 los valores de todas las posiciones de la matriz auxiliar con la función clean_array. Lo que hacemos a partir de ahora, en estos tres bucles es ir mirando posición a posición, si hay un 1 o un 0, y en función de esto, aplicando neighbors, ver en que casillas tenemos que colocar unos o ceros. Lo primero que hacemos es averiguar si en la posición en la que nos encontremos hay un 0 o un 1, y esto lo averiguamos con un get, guardando su resultado en el registro $t5 (cell). Luego calculamos con la función neighbors el valor de la suma de las casillas adyacentes.

Tras esto, llega el momento de seleccionar si poner un 0 o un 1 en la celda en la que nos encontremos. Si nos encontramos en una celda de valor 0, saltamos a poner un 0 en la misma posición de la matriz auxiliar si el valor del neighbors obtenido no es 3. De serlo, saltamos a "vive" conde colocamos un uno en esa posición de la matriz auxiliar. Si nos encontramos en una celda de valor 1, y el valor del neighbors es 2 o 3, saltamos a poner un 1 en la esta posición de la matriz auxiliar. Si no, pondremos un 0 en dicha posición.

Tras colocar el valor, sea el que sea, en la posición correspondiente de la matriz auxiliar, se concluye el bucle. Una vez concluido esto, copiamos el array auxiliar obtenido en el original con la función copy_array, para actualizarlo, produciéndose una iteración con cada actualización. Este proceso se repetirá tantas veces como iteraciones se nos solicite. Como el enunciado no nos da un valor para iteraciones, y debe haberlo, escogimos un valor nosotros.

http://gyazo.com/c45cbd9a52dae3a3e6489f9f8f039af7 imagen parte 1

Aquí vemos el inicio de la función, el paso de parámetros, la inicialización de los registros, $t4 y $t5, neighbors y cell respectivamente, los contadores de iteración it, x, y; y lo importante, la llamada a la función malloc, que nos devolverá una dirección de memoria, donde se encuentra la primera posición de la matriz. Luego entramos en los bucles y obtenemos valores para cell y neighbors.

http://gyazo.com/88ddb1e2c21928cadd9fe394bf84e45e imagen parte 2

Aquí se pueden ver los primeros if, donde se selecciona con que criterio colocar un 0 o un 1 en la posición correspondiente de la matriz. Si la celda es 0, saltamos a "celdacero" donde con el set colocamos un 0 en la posición correspondiente de la matriz auxiliar si el valor del registro $t4 (neighbors) no es tres. Si lo es, saltamos a

“vive”. Si la celda resulta ser 1, saltamos a “vive” si $t4 es 2 o 3, mientras que si no, colocamos nuevamente un 0.

http://gyazo.com/23cfd8d62ea6db581af320e5fd3c9f96 imagen parte 3

Aquí vemos la etiqueta “vive”, donde colocamos un 1 en la posición correspondiente de la matriz auxiliar si se llega hasta aquí. Tras una colocación de un valor en la matriz auxiliar, actualizamos el valor del contador de y y volvemos al bucle. Cuando este contador llega al valor de y, actualizamos el valor de x (con “mini2”) y volvemos al bucle de la x. una vez que el contador de x es igual a x, habremos terminado de colocar valores en la matriz auxiliar. Entonces, en “mini1” copiamos todos los valores de la matriz auxiliar en la matriz original, actualizándose las celdas en las que debe haber 1. Una vez hecho esto, se actualiza el valor de las iteraciones, y se vuelve al bucle de iteración, del cual saldremos cuando el valor de la iteración en que nos encontremos sea igual al valor de iteración que se haya usado en el main (4 en nuestro ejemplo).

**Batería de pruebas:**

http://gyazo.com/5dedbf610d02e99f058871d3375ec168

Desafortunadamente, la única prueba que podemos poner en este apartado es esta, en la que vemos los errores que ocurren al ejecutar este programa.

## Conclusión:

Tras un montón de fallos y aciertos, conseguimos implementar correctamente el programa solicitado en el ejercicio 1, el cual es válido para matrices cuadradas, pero desconocemos el motivo por el cual no sirve para otras matrices. El ejercicio en si no resulto complicado, lo difícil fue solucionar el problema del cambio de los valores de los registros usados entre funciones, es decir, por ejemplo, en el get se nos modificaba el $t4, el cual era crucial para el neighbors que mantuviera su valor correcto. Afortunadamente, sorteamos este problema satisfactoriamente.

En cuanto al ejercicio 3, si bien nosotros consideramos que nuestras funciones están correctamente programadas, y el programa compila, no hemos podido juntarlas con éxito, ya sea por la sobre escritura de valores en los registros, o por algún fallo que desconocemos a la hora de generar la matriz auxiliar en malloc (obtenemos excepción 7 en consola).

Este programa no estaba destinado a imprimir por pantalla nada, (tampoco lo ponía en el enunciado), pero aún modificándolo usando la función print en un archivo .s aparte, no salía el resultado esperado. La dificultad del ejercicio 3 se hizo mayor al tener que trabajar con una matriz no original, y tener que trabajar con más funciones dentro de otras. Consideramos buena la realización de las funciones por separado, pero mala la unión.

En general, la práctica nos resulto larga y complicada, sobre todo teniendo en cuenta que no nos dedicamos exclusivamente a esta práctica, sino que también tenemos otros trabajos.

La planificación resultó un éxito en el primer ejercicio, pero nos dejo con poco tiempo para el segundo. La batería de pruebas planificada tampoco fue la deseada, ya que no conseguimos que el ejercicio 1 funcione con todas las matrices, solo con las cuadradas.