```
 1: #nclude <stdio.h>
 2:
 3:
 4: void met()
 5: {
 6:
 7:     printf("Hbla ");
 8: }
 9:
10:
11: int dev()
12: {
13:     return 5;
14: }
15:
16: int sig(int n)
17: {
18:
19:     return n+1;
20: }
21:
22: char fun(char x[])
23: {
24:     return x[0];
25: }
26:
27:
28: char funfun(char x[],int p)
29: {
30:     return x[p];
31: }
32:
33:
34: void mos(int w[],int tam)
35: {
36:     int ind;
37:
38:     printf("\n|");
39:
40:     for(ind=0;ind<tam;ind++)
41:     printf("%d ",w[ind]);
42:     printf("|\n");
43:
44:
45: }
46:
47: int funA(int p)
48: {
49:
50:     return p+p;
```

```
51: }
52: int funB(int x)
53: {
54:     return x*x;
55: }
56:
57: int mifun(int k)
58: {
59:     int local;
60:
61:     printf("Dentro k=%d ",k);
62:     local = funA(k)+funB(k);
63:     printf("Dentro devuelve=%d ",local);
64:
65:     return local;
66: }
67:
68: int main()
69: {
70:     int ind,res;
71:     int v[10];
72:
73:     met();
74:
75:     for(ind=0;ind<10;ind++)
76:     {
77:         met();
78:
79:
80:     }
81:     printf("\n");
82:
83:
84:     printf("%d\n",res);
85:     res=dev();
86:     printf("%d\n",res);
87:
88:     printf("%d\n",dev());
89:     res=dev()*dev();
90:     printf("%d\n",res);
91:
92:
93:     res=sig(3);
94:     printf("%d\n",res);
95:     printf("%d\n",sig(res));
96:
97:     printf("%d\n",sig(sig(7)));
98:
99:     res=sig(dev());
100:    printf("%d\n",res);
```

```
101:
102:
103:
104:    for(ind=dev();ind<10;ind=sig(ind))
105:    printf("%d,",ind);
106:
107:
108:    printf("\n%c\n",fun("Pepe"));
109:
110:
111:
112:    for(ind=9;ind>=0;ind=sig(ind)-2)
113:    printf("%c",funfun("Multiverso",ind));
114:
115:
116:    printf("\n%d\n",v[0]);
117:
118:    for(ind=0;ind<6;ind++)
119:    v[ind]=dev();
120:
121:
122:    for(ind=5;ind<10;ind++)
123:    v[ind]=sig(dev());
124:
125:
126:    mos(v,10);
127:
128:    mos(v,5);
129:    mos(v,8);
130:
131:    printf("Main llamando a mifun con parametro valiendo =%d\n",2);
132:    ind=mifun(2);
133:    printf("\nMain ind=%d ",ind);
134:
135:    ind=3;
136:    printf("Main llamando a mifun con parametro valiendo =%d\n",ind);
137:    ind=mifun(ind);
138:    printf("\nMain ind=%d ",ind);
139:
140:
141:    //getch();
142:    return 0;
143:}
144:
```

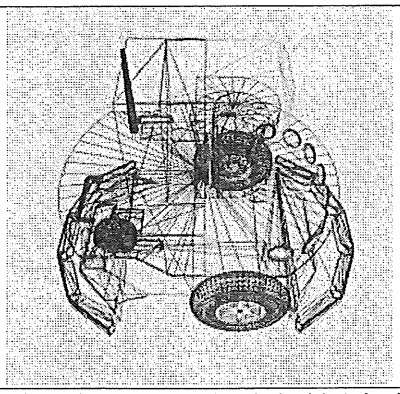
UNIVERSIDAD CARLOS III DE MADRID
PROGRAMACIÓN. GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES
CONVOCATORIA ORDINARIA MAYO 2014. PROBLEMAS

Apellidos:_____
Nombre:_____

Segunda parte. Problemas. Duración 2h 30 min.
Problema 1. (3,5 puntos del total del examen)

En la actualidad es común encontrar robots realizando tareas que era impensable que llevaran a cabo hace pocos años. Un ejemplo son los robots aspiradora que podemos encontrar en cualquier tienda de electrodomésticos. Una parte del éxito de la robótica se debe a los avances del hardware, y otra parte a los avances en el desarrollo de software, y en concreto a la capacidad de dotar a los robots de cierto grado de "inteligencia". Los simuladores son herramientas fundamentales el desarrollo de software para el control de robots, ya que nos permiten evaluar de forma rápida y segura su comportamiento ante distintas situaciones. Un simulador es un software que permite simular el comportamiento del robot y llevar a cabo pruebas que en el mundo real podrían no ser viables o seguras.
En este ejercicio se desarrollará parte de un programa que permita simular el comportamiento de varios robots, basado en los vehículos de *Braitenberg*. Este tipo de vehículos exhiben un comportamiento basado en la relación directa entre los diferentes sensores del robot (como sonar, láser, contacto, etc.) y sus diferentes actuadores (ruedas, brazos, etc.).
El ejercicio se basa en el robot Pioneer 3DX, una plataforma robótica para fines educativos y de investigación que permite la utilización de distintos sensores y actuadores. En la Figura 1 se muestra el P3DX simulado con sus dos ruedas, y la distribución de sus sensores de distancia (sonar).

(a)



(b)

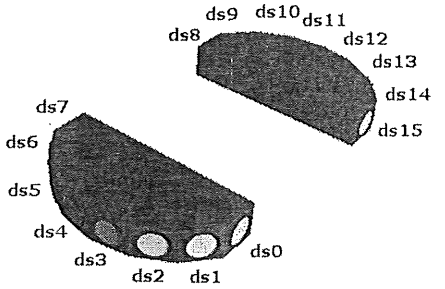


Fig. 1. Robot Pioneer 3DX simulado (a). Distribución de sensores sonar (b)

Para desarrollar el programa que simula el comportamiento de los robots es necesario completar los siguientes apartados.

Apartado 1 (2 puntos /10)
Defina un array de estructuras o registros (robots) que permita almacenar los datos de cada uno de los robots que forman parte de la simulación. Para cada robot se almacenará:

- Nombre
- Nombre de cada uno de los sensores de distancia (sónar) activos (que serán 8 o 16). Como se ve en la figura 1.b el nombre está formado por el número de sensor precedido del el prefijo "*ds*"
- Peso asignados a cada sensor para cada rueda (tipo `double`) . Estos valores servirán para calcular la velocidad de cada una de las dos ruedas motrices del robot P3DX. Cada sensor tendrá dos valores de peso diferentes, pues la medida de un sensor afectará de forma diferente al cálculo de la velocidad de cada rueda.
- Número de sensores sonar activos.

Apartado 2 (1 punto /10)
Escriba una función que inicialice la simulación. Esta función deberá obtener, pidiéndolos al usuario, el número de robots que van a participar en la simulación (máximo 10) y el número de veces que se va a repetir la simulación (número de ciclos de ejecución, máximo 10000). La función deberá controlar que los valores introducidos por el usuario son válidos.

Apartado 3 (2 puntos /10)
Escriba una función para inicializar un robot. La función deberá pedir al usuario el número de sensores sonar (de distancia) activos que tiene cada robot, que como se ha dicho para el P3DX puede ser 8 o 16, controlando que el valor introducido sea válido.
Después debe inicializar cada sensor, para lo que debe asignarle nombre de acuerdo con lo definido en el apartado uno, llamar a la función `inicializar_sensor` que activa el sensor para que luego pueda ser utilizado y pedir al usuario los pesos que desea asignar al sensor para cada una de las ruedas motrices.
Para inicializar los sensores se hará uso de la función (que se asume que existe ya, y no es necesario desarrollar) `inicializar_sensor(char nombre[])` de tipo `void`. Esta función recibe como parámetro el nombre de un sensor y se encarga de activarlo para que luego pueda ser utilizado

Nota: puede usar la función `sprintf` para concatenar el prefijo asignado al sensor (*"ds"*) y el número de sensor. La función `sprintf` funciona de manera similar a la función `printf` con la salvedad de que en vez de imprimir la cadena por pantalla, imprime en una cadena.

```
int sprintf(char *cadena, const char *formato, ...);
```

Por ejemplo, suponiendo que `nombre` contiene la cadena "`PEPITO`" y `num_s=8`, tras la llamada

```
sprintf(salida, "El Robot %s tiene %d sensores.", nombre, num_s);
```

la variable `salida` contendrá la cadena "`El Robot PEPITO tiene 8 sensores`".

Apartado 4 (3 puntos /10)
Escriba una función que mueva el robot para un paso (ciclo) de simulación. La función que leerá los valores de distancia medidos por los sensores, calculará la velocidad para cada rueda, y dará la orden de mover las ruedas.
Se asume que existe la función `sensor_get_value(char sensor[])` que recibe como parámetro el nombre del sensor y devuelve su lectura de distancia (`double`).
La velocidad de cada una de las dos ruedas motrices del robot será la suma de:

*peso_sensor * (1.0 – (valor_lectura_sensor /RANGO)*

donde el peso de cada sensor lo determina el usuario al inicio de la simulación y el RANGO es una constante definida en el código fuente.
Por otro lado, se asume que existe la función `wheels_set_speed(double S1,double S2)` de tipo `void` que lleva a cabo el movimiento del robot en un ciclo de simulación, y recibe como parámetro la velocidad de cada una de las dos ruedas.

Apartado 5 (2 puntos /10)
Implemente la función `main` que, basándose en las funciones anteriores, lleve a cabo la simulación. La función debe realizar las siguientes tareas:

- Inicializar la simulación y los robots.
- Ejecutar la simulación (para el número de ciclos determinado por el usuario) calculando en cada ciclo el movimiento de cada uno de los robots.

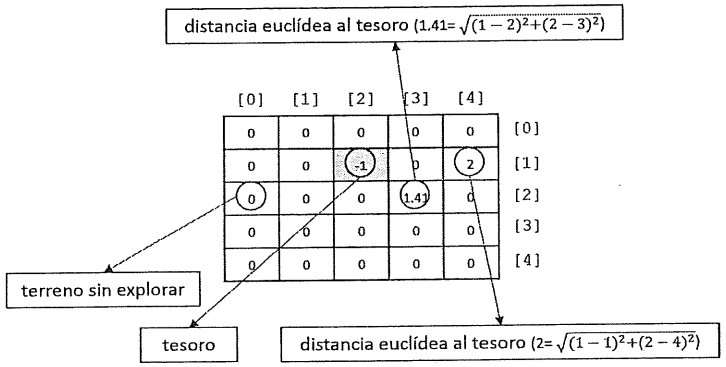
Problema 2. (3,5 puntos del total del examen)

Introducción

El juego *el tesoro escondido* consiste en lo siguiente: en un tablero de $N \times M$ casillas hay escondido un tesoro; el jugador tendrá que adivinar, tras una serie de intentos, cuáles son las coordenadas bajo las que se encuentra escondido. En cada turno el jugador indica las coordenadas donde cree que está enterrado el tesoro; se aplican las siguientes reglas:

- Si el jugador no acierta
  - Se le indica la distancia en línea recta a la que se encuentra el tesoro.
  - Si aún tiene la posibilidad, se le ofrece utilizar un radar de orientación, con cual podrá realizar dos tipos de exploraciones
    - Horizontal: comunica al usuario si el tesoro se encuentra al Este, al Oeste o en la misma longitud respecto de las últimas coordenadas de búsqueda introducidas.
    - Vertical: comunica al usuario si el tesoro se encuentra al Norte, al Sur o en la misma latitud respecto de las últimas coordenadas de búsqueda introducidas.
- Si el jugador acierta o se supera un número máximo de aciertos, se acaba la partida.

Para implementar este juego existirá una matriz *mapa* que contendrá un -1 en la casilla donde se encuentra el tesoro, un 0 las casillas que no hayan sido aún exploradas por el usuario y el valor de la distancia euclídea (en línea recta) al tesoro en las casillas que sí hayan sido exploradas.
Ejemplo:



Preguntas

1. Búsqueda (3/10)

Implemente una función llamada `buscar_tesoro` que calcule las coordenadas en las que se encuentra el tesoro (`fila_t, columna_t`). La cabecera de la función ha de ser la siguiente:

```
void buscar_tesoro(float mapa[N][M], int *fila_t, int *columna_t);
```

2. Cálculo de distancia (1/10)

Implemente una función llamada `distancia_tesoro` que actualice en la matriz *mapa* la distancia en línea recta desde las coordenadas de exploración (`fila_e, columna_e`) hasta el tesoro. La cabecera de la función ha de ser la siguiente:

```
void distancia_tesoro(float mapa[N][M], int fila_e, int columna_e);
```

Nota: Para calcular la distancia en línea recta hágase uso de la fórmula de la distancia euclídea $d = \sqrt{(f_t - f_e)^2 + (c_t - c_e)^2}$, en la que f_t y c_t se corresponden con la fila y la columna bajo la cual se encuentra escondido el tesoro y f_e y c_e con la fila y la columna de exploración. Puede hacer uso de las funciones potencia y raíz cuadrada, cuyas cabeceras son:

- `float pow(float base, float exp);`
- `float sqrt(float valor);`

3. Impresión del mapa (1/10)

Implemente una función llamada `imprimir_mapa` que muestre por pantalla la matriz *mapa* teniendo en cuenta que la casilla que contiene el tesoro tiene que ser mostrada como si no hubiera sido explorada. Las casillas no exploradas deben mostrarse con el valor 0 y las exploradas con el valor de la distancia calculada al tesoro. La cabecera de la función ha de ser la siguiente:

```
void imprimir_mapa(float mapa[N][M]);
```

4. Utilización del radar (2/10)

Implemente una función llamada `usar_radar` tal que, a partir de las coordenadas de exploración introducidas previamente por el usuario (`fila_e`, `columna_e`) y del tipo de radar a utilizar, devuelva el carácter 'N' si el tesoro se encuentra arriba de esas coordenadas de búsqueda, 'S' si se encuentra abajo (radar vertical), 'O' si se encuentra a la izquierda y 'E' si se encuentra a la derecha (radar horizontal). Si el tesoro se encuentra en la misma latitud o longitud, la función devolverá el carácter 'L'. La cabecera de la función ha de ser la siguiente:

```
char usar_radar(int tipo, float mapa[N][M], int fila_e, int columna_e);
```

5. Juego completo (3/10)

Implemente una función *main* que, utilizando las funciones anteriores, permita a un usuario jugar una partida a *el tesoro escondido*. Este programa principal deberá de:

1. Declarar las variables necesarias.
2. Inicializar la matriz *mapa*. Para esta inicialización se hará uso de una función que se asume ya implementada llamada `esconder_tesoro` que recibe como parámetros dicha matriz y no devuelve ningún valor. La matriz del mapa quedará con un -1 en una casilla aleatoria y con un 0 en el resto (terreno sin explorar).
3. Ir pidiendo coordenadas de búsqueda al jugador hasta que este gane o supere un número máximo de intentos dado por la expresión `(N + M) / 3`. Una vez que el usuario haya introducido las coordenadas, si no ha encontrado el tesoro, se imprimirá el mapa de búsqueda y se le ofrecerá la posibilidad de usar el radar si no lo ha utilizado previamente (se ha de preguntar al jugador el tipo de radar a usar). Si se utiliza el radar, se comunicará al usuario si el tesoro se encuentra al Norte, al Sur, al Este, al Oeste o si está en la misma latitud o longitud respecto de las coordenadas de exploración introducidas.
4. Terminada la partida, informar al usuario de si ha ganado o si por el contrario ha superado el número de intentos.

UNIVERSIDAD CARLOS III DE MADRID
PROGRAMACIÓN. GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES
CONVOCATORIA EXTRA ORDINARIA JUNIO 2014. PROBLEMAS

Apellidos:_____
Nombre:_____

Segunda parte. Problemas. Duración 2h 30 min.
Problema 1. (3,5 puntos del total del examen)

Se desea diseñar un programa en C para realizar una simulación de la Copa Mundial de Fútbol que se está celebrando actualmente en Brasil. Vamos a suponer que en el torneo participan un total de 16 selecciones. Cada una de ellas se modela mediante una estructura de datos que contiene la siguiente información: identificador de selección (número entero entre 1 y 16), nombre del país, cabeza de serie (número entero que indica si esta propiedad es verdadero o falso), coeficiente FIFA (número real que indica lo buena que es una selección), y un vector con los datos de los 20 jugadores convocados.
Cada jugador se modela mediante una estructura de datos que contiene la siguiente información: identificador del jugador (número entero entre 1 y 20 que coincide con su dorsal), nombre completo y número de goles marcados durante el campeonato.
Se pide:

a) Definir las estructuras necesarias para representar las composiciones de datos descritas. [1 punto]

b) Definir una función que solicite por pantalla los datos de una selección, incluido el coeficiente FIFA. El número de goles marcados por cada jugador debe inicializarse a 0 y al identificador de selección debe asignársele el valor dado por el parámetro formal *numsel*. Además, las selecciones con identificador entre 0 y 3 serán cabeza de serie. La función debe tener la siguiente cabecera [1.5 puntos / 10]:

```
void anadir_seleccion(struct seleccion selecciones [MAXSELE], int numsel);
```

c) Para disputar el torneo, se ha decidido dividir inicialmente las selecciones en NUMGRUPOS grupos de NUMSELECCIONES selecciones, que vamos a representar en nuestro programa mediante una matriz con el identificador de las selecciones y de tamaño NUMGRUPOS x NUMSELECCIONES. Cada celda contendrá el identificador de una selección, y cada columna corresponderá con un grupo. Se pide generar el código de una función que genere automáticamente esta matriz teniendo en cuenta los siguientes requisitos [1.5 puntos/ 10]:
   1) Cada grupo sólo debe contener una selección cabeza de serie (existen únicamente 4 en el campeonato).
   2) Cada selección sólo puede estar en un grupo.

   La función debe tener la siguiente cabecera:

```
void generar_grupos(struct seleccion selecciones [MAXSELE], int grupos [NUMGRUPOS][NUMSELECCIONES]);
```

d) Diseñar una función que determine el vencedor de cada grupo en la fase inicial. Se considera que el vencedor de un grupo es directamente la selección del grupo que tenga un mayor coeficiente FIFA en el campo correspondiente. [2 puntos/ 10].

```
void ganadores_fase_inicial (struct seleccion selecciones [MAXSELE], int grupos [NUMGRUPOS][NUMSELECCIONES],
                             int ganadores_fase1 [NUMGRUPOS]);
```

e) Suponiendo que al final de la primera fase del campeonato se hubiese actualizado el número de goles marcados por cada jugador, se pide diseñar una función que ordene los jugadores de una selección en función del número de goles marcados. [2 puntos].

```
void ordenar_jugadores(struct seleccion sel, struct jugador jugadores_ordenados[MAXJUG]);
```

f) Por último, diseñe la función principal del programa de acuerdo con el siguiente conjunto de pasos [2 puntos/ 10]:
   1. Requerir al usuario que complete los datos de las selecciones que participan en el campeonato.
   2. Realizar una simulación de la fase inicial del campeonato (creación de la matriz de grupos y determinación del ganador de cada uno de ellos).
   3. Mostrar el nombre de la selección ganadora de cada grupo en la fase inicial.
   4. Mostrar el nombre del máximo goleador de las selecciones ganadoras de cada grupo en la fase inicial.

Problema 2. (3,5 puntos del total del examen)

Se quiere programar una aplicación *e-learning* para ayudar en el aprendizaje de un conjunto de temas. La aplicación realiza al usuario una serie de preguntas (que nunca se repiten), organizadas en tandas de 2 preguntas por tema. Cada pregunta sólo puede tener por respuesta *verdadero* o *falso*. El programa comienza con una tanda de 2 preguntas del tema 1. Si el usuario responde correctamente las dos preguntas se pasa a una nueva tanda del tema siguiente, pero si falla alguna, se le vuelve a presentar una tanda del primer tema, hasta que responda bien una tanda completa. Lo mismo sucede en el tema 2 y siguientes, con la diferencia de que en estos temas, si se fallan las dos respuestas en una tanda, se sitúa al usuario en el tema anterior hasta que lo supere contestando bien las dos preguntas de una tanda. Si contesta una bien y otra mal, la siguiente tanda será del mismo tema.
El programa termina cuando se hayan respondido bien las dos preguntas de una tanda del último tema. Al terminar, el programa comunicará al usuario la nota obtenida en cada tema y le recomendará repasar los temas en los que la nota obtenida sea menor que una nota de corte. Tras cada tanda de preguntas se da al usuario la opción de terminar el programa, en cuyo caso, por ser una salida forzada, no se le comunica la nota ni la recomendación.

La nota de un tema se calcula como 10 x aciertos / preguntas hechas

El programa utiliza una llamada a la función *cargarPreguntas,* que se encarga de rellenar las fichas de preguntas, cuyo prototipo es el siguiente:

void cargarPreguntas (struct fichaPregunta cuestiones[]);

Esta función se considera que ya está disponible, por lo que bastará con insertar en las directivas del preprocesador la línea <include preguntas.h>

Se pide:

1. Definir las estructuras de datos: **(1 punto /10)**
    a. Ficha de pregunta, con tres campos: número de tema, texto de la pregunta y respuesta (1: verdadero o 0: falso).
    b. Vector de fichas de preguntas, que almacene todas las preguntas posibles
    c. Ficha de resultados del usuario en un tema, con tres campos: número de cuestiones preguntadas, número de contestadas bien, y nota.
    d. Vector de fichas de resultados, que almacene los resultados del usuario en cada tema (el programa está diseñado para que haya un único usuario)
2. Escribir la función *elegirPregunta,* que tiene como parámetros formales el número de tema, un vector de fichas de preguntas y un vector de preguntas usadas. La función buscará la primera pregunta no usada del tema y devolverá su posición en el vector que contiene las fichas de preguntas. **(2 puntos /10)**
3. Escribir la función *preguntarTema,* que tiene como parámetros formales el número de tema, un vector de fichas de preguntas, un vector de preguntas usadas y un vector de fichas de resultados. La función *preguntarTema* debe llamar a la función *elegirPregunta* para elegir qué preguntas hacer, formularlas al usuario y actualizar el número de respuestas correctas en un determinado tema. **(3 puntos /10)**
4. Escribir la función *main* que contendrá el bucle principal para subir o bajar de tema, y llamará a *cargaPreguntas, preguntarTema, y calificar* y *recomendar.* **(3 puntos /10)**
5. Escribir las funciones *calificar y recomendar.* Calificar calcula la nota de cada tema y la muestra por pantalla, y recomendar muestra la lista de temas que se deben repasar. **(1 puntos /10).**

Nota: el número máximo de preguntas y de temas y la nota de corte se definirán como constantes. Se les puede dar el valor que se desee.

UNIVERSIDAD CARLOS III DE MADRID
PROGRAMACIÓN. GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES
CONVOCATORIA ORDINARIA MAYO 2013. PROBLEMAS

Apellidos:_____
Nombre:_____
NIA:_____

Problema 1 (3,5 puntos)

El Bingo es un juego de azar muy antiguo, y bastante popular alrededor de todo el mundo. Se juega con un bombo que contiene 90 bolas numeradas. Cada jugador tiene un cartón con números entre 1 y 90. Un locutor va sacando las bolas del bombo y recitando los números, y los jugadores que tienen el número en su cartón lo tachan. El ganador es el primer jugador en tachar todos los números de su cartón (a esto se le llama "hacer bingo").
En este ejercicio se va a escribir un programa en C que permita simular una versión simplificada del juego del Bingo.
Supondremos que pueden jugar como máximo 100 jugadores. Se debe definir un vector de estructuras que almacene la información de cada jugador, que será:

- Nombre
- Edad
- Cartón actual que está jugando

El cartón se representará como una matriz de 3 filas y 9 columnas que contendrá números entre el 1 y el 90. Este cartón se irá modificando a lo largo de la partida (según se vayan "tachando" números). La siguiente tabla muestra un ejemplo de cartón:

21	51	89	82	4	1	68	29	1
19	10	35	43	68	8	2	38	90
43	6	22	30	59	57	26	1	85

Antes de comenzar la partida, se pedirá por teclado el número de jugadores (*n_jugadores*) que deberá estar entre 1 y 100. Se irán leyendo de teclado el nombre y edad cada jugador. Además, se rellenará la matriz que representa el cartón, para lo que se utilizará la siguiente función (que se supone ya programada, por lo que solo es necesario llamarla):

```
void inicializaCarton (int miCarton[FILAS][COLUMNAS])
```

Actualizados estos datos, se comenzará la partida. Para ello, se irán leyendo por teclado números (que simularán los que salen del bombo) y deberán estar entre 1 y 90 (para simplificar, no es necesario controlar si un número ya ha salido anteriormente, podrán repetirse).
A partir de este número, se deberá comprobar si éste está en los cartones de los jugadores. Si fuera así, se deberá sustituir en el cartón de ese jugador por un 0, indicando que el número está tachado.
Este proceso se repetirá hasta que un jugador obtenga bingo, es decir, todos los números de su cartón ya hayan sido tachados. Podría darse el caso de que varios jugadores obtengan bingo en la misma jugada.
Acabada la partida, deberá mostrarse por pantalla una clasificación de los jugadores, en función de cuántos números tachados tenga cada uno.

El programa se deberá hacer **OBLIGATORIAMENTE** utilizando **FUNCIONES** y de acuerdo a los pasos que se detallan a continuación (tener en cuenta que la función principal deberá realizarse al final):

1. Defina los tipos de datos estructurados necesarios para almacenar toda la información del juego (0,5 puntos).
2. Implemente la función "*inicializaJugadores*". Esta función recibe como parámetro el vector con todos los jugadores y el número total de jugadores. Así, inicializará el vector leyendo nombre y edad por teclado, e inicializando cada cartón LLAMANDO para ello a la función "*inicializaCarton*" que se supone ya programada (0,5 puntos)..
3. Implemente la función "*tachaNumero*". Esta función recibe como parámetro un cartón y un número, comprueba si dicho número está en el cartón, y lo tacha (lo marca con un 0). (1 punto).
4. Implemente la función "*compruebaFila*". Esta función recibe como parámetro un cartón y el número de una fila, y comprueba si dicha fila está completa (todos los números están a 0). Devolverá un 1 si la fila está completa y un 0 en caso contrario. (1 punto).
5. Implemente la función "*compruebaBingo*". Esta función recibe como parámetro un cartón y comprueba si todos sus números están a 0. Para ello, deberá llamar OBLIGATORIAMENTE a la función *compruebaFila.* Si el cartón tiene todas sus casillas a 0, la función deberá devolver un 1, en caso contrario, un 0. (1 punto).
6. Implemente la función "calculaNumerosParaBingo". Esta función recibe como parámetro un cartón y devuelve el número de números que faltan en dicho cartón para conseguir un Bingo (números distintos de 0 que tiene dicho cartón) (1 punto).
7. Implemente la función "mostrarClasificacion". Esta función recibe como parámetro el vector de jugadores y muestra por pantalla una clasificación de dichos jugadores en función de los números tachados. Se mostrará primero los que menos números sin tachar tengan. (Si dos jugadores están empatados podrá mostrarse primero cualquiera de ellos). Se mostrará un listado con el nombre del jugador y el cuántos números sin tachar tiene. (2 puntos).
8. Escriba el programa principal teniendo en cuenta las indicaciones del juego y empleando las funciones desarrolladas anteriormente. No es necesario volver a declarar las variables ya declaradas en el apartado 1. (3 puntos)

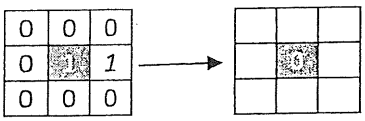
PROBLEMA 2.

Introducción

El *juego de la vida* es un juego de simulación creado en 1970 por el matemático inglés John Horton Conway. El juego se desarrolla sobre un tablero de n x m celdas que alojan "organismos". Al principio del juego, el jugador especifica qué celdas contienen organismos vivos. Los organismos vivos interactúan con las 8 celdas adyacentes, de forma que la configuración de la siguiente generación se calcula según las siguientes reglas:

- Una celda viva con menos de dos vecinos vivos muere en la siguiente generación (por *aislamiento*)
- Una celda viva con dos o tres vecinos vivos vive en la siguiente generación (por *colaboración*)
- Una celda viva con más de tres vecinos vivos muere en la siguiente generación (por *superpoblación*)
- Una celda muerta con 3 vecinos vivos vive en la siguiente generación (por *reproducción)*

Ejemplo: (un 1 representa una celda viva y un 0 una muerta)



En este ejemplo se aplica la primera regla, la célda marcada está viva y tiene un vecino vivo. En la siguiente generación, la celda marcada pasa a estar muerta.

Estas reglas se aplican a todas las celdas del tablero en cada generación para generar la siguiente configuración del tablero.

Preguntas

En este problema se implementará paso a paso una versión simplificada del juego de la vida. Siga las instrucciones que se ofrecen en cada una de las siguientes secciones.

1. **Definición de estructuras de datos**

   Declare una matriz de dos dimensiones para representar el tablero y guardar la información sobre las celdas vivas y muertas. Debe representar las celdas con números enteros: 1 celda viva, 0 celda muerta. El tamaño del tablero debe definirse con dos constantes FIL, COL.

2. **Lectura e Inicialización de tablero**

   Implemente una función llamada *iniciarTablero* que reciba como parámetro una matriz *tablero* y lea de teclado los valores (0 ó 1) para cada una de sus casillas. La función no debe devolver ningún valor.

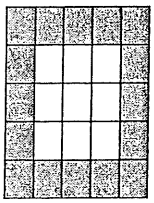
3. Recuento de población

Implemente una función llamada *contarPoblación* que reciba como parámetro una matriz *tablero* y calcule el número de individuos vivos del tablero. La función debe devolver un valor entero resultado de dicho cálculo.

4. Recuento de vecinos

Implemente una función llamada *contarVecinos* que reciba como parámetro una matriz *tablero* y dos valores enteros *i*, *j*. La función debe contar el número de organismos vivos vecinos de la celda (*i*, *j*) del tablero. La función debe devolver un valor entero resultado de dicho cálculo.

NOTA: Asuma que la función *contarVecinos* nunca será llamada para una celda situada en el borde del tablero; por ejemplo, para un tablero de tamaño 5x5 como el de la siguiente figura, no se llamará a *contarVecinos* con (0, 0), (0, 1), ..., (0, 4), (1, 0), (1, 4), (2, 0), (2, 4), ..., (4, 0), (4, 1), ..., (4, 4).



5. Cálculo de la segunda generación

Implemente una función *main* que, basándose en las funciones anteriores, obtenga la segunda generación de individuos a partir de una configuración inicial del tablero aplicando las reglas anteriores. La función debe llevar a cabo las siguientes tareas:

- Declarar variables tablero y tablero_siguiente
- Leer la configuración inicial del tablero (tablero)
- Imprimir en pantalla el tablero inicial y el número de celdas vivas
- Para cada una de las celdas del tablero que no esté en el borde:
  - Calcular el número de vecinos vivos
  - Calcular el valor correspondiente de tablero_siguiente de acuerdo a las reglas dadas en la Introducción
  - Imprimir en pantalla el tablero_siguiente y el número de celdas vivas

UNIVERSIDAD CARLOS III DE MADRID
PROGRAMACIÓN. GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES
PROBLEMAS ENERO 2011.

Apellidos:_____
Nombre:_____
NIA:_____

PARTE 2. PROBLEMAS. Duración 2 horas y 30 minutos. 7 PUNTOS

Responda a cada problema en folios diferentes.

Problema 1 (3,5 puntos)

En la literatura científica, cada documento está caracterizado por su título, autor, año de publicación, una palabra clave, el área científica a la que pertenece, y el índice de impacto en el área (número real). Así, por ejemplo, una posible descripción de un documento puede ser:

Título: Estructura Molecular de los Ácidos Nucleicos
Autor: James D. Wantson
Año: 1953
Palabra: Nucleótidos
Área: 5
Impacto: 31,43

Teniendo en cuenta estos datos, se desea gestionar la Bibliografía de una Tesis Doctoral en la que se satisfacen los siguientes requisitos:

- El número de citas bibliográficas es inferior a 150
- Cada cita bibliográfica puede pertenecer a **una** de las siguientes áreas científicas:
  1 = Citoesqueleto
  2 = Canales
  3 = Proteína
  4 = Membrana
  5 = RNA

Desarrollar un programa en C que:

1. Lea y almacene en un array de estructuras los datos (título, autor, año de publicación, palabra clave, área científica, e índice de impacto) de todos los documentos que forman de la Bibliografía.
2. Lea el identificador de un área científica (número entero entre 1 y 5) y muestre por pantalla:
   a. El nombre de dicha área (Citoesqueleto, Canales, Proteína, Membrana o RNA) y
   b. Los datos de todos los documentos que pertenecen a esa área ordenados de mayor a menor índice de impacto.

Nota: En la resolución de este problema no es necesario usar funciones.

Problema 2. (3,5 puntos):

El juego de las tres en raya es un famoso juego donde dos jugadores tienen que marcar de forma alterna los espacios de un tablero de 3×3 casillas. Un jugador gana el juego si consigue tener en una misma línea tres de sus fichas: la línea puede ser horizontal, vertical o diagonal.

Se pide implementar en el lenguaje de programación C una versión **simplificada** de este juego. En el programa a implementar, cada casilla del tablero estará representada por un número entero empleando la siguiente codificación:

El número 0 representa una casilla vacía.
El número 1 representa una casilla con una ficha del Jugador1.
El número 2 representa una casilla con una ficha del Jugador2.

El primer turno de la partida es siempre para el Jugador1 y se admitirá que no es posible mover una ficha una vez que ésta ha sido colocada en el tablero. La partida termina cuando uno de los jugadores gana o cuando se hayan colocado 9 fichas en el tablero.

Para realizar el programa descrito se solicita:

a. Defina los tipos de datos estructurados necesarios para almacenar toda la información del juego e implemente la definición de la función "initTablero". La función "initTablero" debe emplearse para inicializar un tablero indicando que todas sus casillas están vacías.

b. Implemente la definición de la función "anadeFicha", necesaria para colocar una nueva ficha de un jugador en el tablero. Esta función debe recibir, entre otros parámetros, las coordenadas de la casilla en la que se desea colocar la ficha y debe devolver un valor que indique si ha sido posible colocar la ficha en la casilla especificada o no.

c. Implemente la definición de la función "compruebaGanador", necesaria para determinar si la **última jugada** provoca que el jugador que la ha realizado gana la partida. Se recuerda que un jugador gana el juego si consigue tener en una misma línea tres de sus fichas. Para el desarrollo de este punto considere que se cuenta con las siguientes dos funciones **ya implementadas**:

   1. La function "perteneceDiag" (`int perteneceDiag(int fila, int columna)`) recibe las coordenadas de una posición del tablero y devuelve un valor lógico (1 ó 0)que controla si dicha casilla pertenece a la diagonal del tablero (/).

   2. La function "perteneceDiagInv" (`int perteneceDiagInv(int fila, int columna)`) recibe las coordenadas de una posición del tablero y devuelve un valor lógico (1 ó 0) que controla si dicha casilla pertenece a la diagonal inversa del tablero (\).

Empleando las funciones definidas en los anteriores puntos del ejercicio implemente la función principal del programa. El programa solicitará por teclado la ubicación de cada nueva ficha de forma alterna para cada uno de los dos jugadores y comprobará si al añadir una nueva ficha el jugador que la ha añadido resulta ganador.

En la resolución de este problema se debe tener en cuenta que **no es posible mover una ficha una vez ha sido colocada en el tablero**, por lo que el juego se resolverá como máximo en 9 jugadas.

UNIVERSIDAD CARLOS III DE MADRID
PROGRAMACIÓN. GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES.
JUNIO 2012. PARTE 2. PROBLEMAS.

Apellidos:_____
Nombre:_____
NIA:_____

PARTE 2. PROBLEMAS. Duración 2 horas y 45 minutos. 7 PUNTOS

Responda a cada problema en folios diferentes.

Problema 1 (3,5 puntos)

El objetivo de este ejercicio es desarrollar un PROGRAMA MODULAR en C que implementará una versión simplificada del juego *Hundir la Flota*, que llamaremos Mini-Flota.

Mini-Flota es un juego de dos jugadores: uno es el jugador humano (que llamaremos simplemente 'humano') y otro es el ordenador (que llamaremos 'CPU'). Al principio del juego, la CPU coloca sus barcos en una cuadrícula de 10x10 casillas, que representa el tablero del juego. A continuación, el humano ataca a la CPU indicando las coordenadas de su disparo. La CPU informará del resultado del disparo (si corresponde a uno de los barcos o no). Si un barco ha sido alcanzado, la CPU comprueba si se trata de la última celda sin daños del barco, en cuyo caso anuncia que el barco ha sido hundido. En este caso, cuando un barco ha sido hundido, la CPU comprueba si era el último de su flota; si es así, la CPU proclama que el humano ha ganado el juego. Este ciclo disparo/comprobación se repite hasta que el humano gana el juego.

IMPORTANTE: En esta versión simple del juego, el humano y la CPU no disparan en turnos alternativamente. Solo la CPU tiene barcos y es el humano quien dispara repetidamente sobre la flota de la CPU.

Para resolver este problema, deben realizarse las siguientes tareas:

- (0.25) Crear una estructura de datos llamada Barco para almacenar la información de un barco:
  - posicion: Posición del tablero donde el barco está situado. Es una estructura con dos atributos:
    - fila: fila del tablero (valor entero 0, 1, ..., 9)
    - columna: columna del tablero (valor entero 0, 1, ..., 9)
  - orientacion: Valor 0, 1 que representa la orientación del barco (0 horizontal, 1 vertical)
  - tamano: Número de celdas que ocupa el barco (valor entero 1, ..., 5)

- (0.25) Crear una estructura de datos para almacenar el tablero 10x10. Utilizaremos diferentes caracteres para representar los posibles valores de las celdas:
  - *AGUA*: Casilla vacía --> 'o' (letra *o*)
  - *AGUA_TOCADO*: Casilla de agua que ha recibido un disparo --> '.' (carácter punto)
  - *BARCO*: Casilla de barco no tocada --> '#' (carácter almohadilla)
  - *BARCO_TOCADO*: Casilla de barco tocada --> '@' (carácter arroba)

UNIVERSIDAD CARLOS III DE MADRID
PROGRAMACIÓN. GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES
CONVOCATORIA ORDINARIA: 24 DE MAYO DE 2012.

Apellidos:_____
Nombre:_____
NIA:_____

Problema 1 (3.5 puntos)

La encriptación de información en conflictos bélicos consiste en modificar los mensajes generados a partir de un alfabeto en base a una "clave" que solo conocen el emisor y el receptor. De esta forma se puede ocultar dicha información al enemigo.

Julio Cesar ideó un sistema que consiste en modificar el alfabeto original alterando el comienzo del mismo en un número de posiciones, en el rango 1 a 26, y haciendo que **las letras desplazadas queden dispuestas en orden inverso al final del alfabeto**.

Por ejemplo, si la clave utilizada es +3, la A pasa a ser "D", la B pasa a ser "E" y así sucesivamente, y el **final del alfabeto será C, B, A.**

La correspondencia entre el alfabeto original y el codificado será en este ejemplo la siguiente.

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z "_"
D E F G H I J K L M N O P Q R S T U V W X Y Z - C B A
```

Con esta clave (+3) el mensaje "HOLA-MUNDO" una vez ENCRIPTADO sería "KRODAPXQGR"

Obsérvese que los espacios en blanco se codifican como una letra más, representada por "_" en este ejemplo.

Se pide implementar un programa en *C* que permita:

1. Almacenar en memoria y mostrar por pantalla el alfabeto original indicado en el enunciado.
2. A partir de una clave que se solicitará por pantalla, modificar el alfabeto original según los criterios establecidos y mostrar dicho alfabeto por pantalla.
3. Solicitar un mensaje de 20 caracteres y mostrarlo codificado según el nuevo alfabeto.

El alfabeto original, el modificado y los mensajes se codificarán como vectores de caracteres, se desaconseja hacerlo con strings. Además, por simplicidad, se considerará que la clave utilizada siempre será un número positivo.

No es necesario utilizar funciones.