UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

INTRODUCCIÓN A LOS COMPUTADORES

PROYECTO DE PRÁCTICAS | 2021-22

JUEGO 2048

1er enunciado de proyecto

Crear dos procedimientos y el programa que los invoque. Los dos procedimientos a implementar se encargará de resolver la transformación de una posición bidimensional (fila y columna) de una matriz a la correspondiente posición vectorial y viceversa. Se tendrá en cuenta que la matriz es cuadrada y que el número de filas/columnas viene dado por una constante, cuya definición se incluirá en la cabecera del programa mediante la siguiente línea:

```
COLFILJUEGO EQU 4       ;puedes variarlo entre 1 y 15 para hacer pruebas
```

1. El primer procedimiento se llamará **"MatrizAVector"** y se comportará de modo que dados como parámetros de entrada una fila (variable filMatrizJuego de tipo DB) y una columna (variable colMatrizJuego de tipo DB) de una matriz, calculará la posición lineal correspondiente de la matriz, es decir, el índice del elemento teniendo en cuenta que la matriz en memoria está almacenada como un vector, y la almacenará en la variable posMatrizJuego (tipo DB). A modo indicativo:

```
posMatrizJuego = filMatrizJuego*COLFILJUEGO + colMatrizJuego
```

2. El segundo procedimiento se llamará **"VectorAMatriz"** y se comportará de modo que dado como parámetro de entrada un índice línea de la matriz (variable posMatrizJuego de tipo DB)) devuelva en las variables filMatrizJuego y colMatrizJuego la fila y columna de la matriz del elemento correspondiente. A modo indicativo:

```
filMatrizJuego = posMatrizJuego/COLFILJUEGO ; colMatrizJuego = posMatrizJuego%COLFILJUEGO
```

2do enunciado de proyecto

Crear dos procedimientos y el programa principal que los invoque usando este código fuente como base.

1. El primer procedimiento, **ComprobarFinJuego** , debe comprobar si un vector con N posiciones/celdas de tamaño DW contiene un cierto elemento (tope) o tiene todas sus celdas con valores distintos de cero (o lo que es lo mismo, que ninguna tenga el valor cero). Es decir, debe responder a la siguiente cabecera de documentación:

- Si el tablero esta lleno y no hay un valor TOPE se ha perdido la partida
- Si hay una celda con valor TOPE se ha ganado la partida
- E: SI apunta al vector/tablero de juego
- E: CX contiene el número de celdas del vector
- E: TOPE es una variable que contiene un número a buscar que en caso de encontrarse devuelve partida ganada
- S: ah=0 si el juego debe continuar

- ah=1 si el juego debe terminar, usar al para distinguir entre partida ganada o perdida. mensajes de partida ganada o perdida impresos en pantalla, si es el caso

2. El segundo procedimiento **NumAleatorio2o4** debe generar un número aleatorio que será 2 ó 4, que devolverá en AH como parámetro de salida (2 ó 4). Trucos: para calcular este valor, como debemos proporcionar una salida entre dos posibles opciones (2 ó 4), con 1 bit aleatorio que usemos es suficiente para distinguir entre dos opciones (ver NumAleatorio proporcionado y reusar la parte correspondiente a la obtención de la hora).

Se recomienda usar instrucciones como test o and para crear máscaras y quedarnos con un solo bit; shr o shl para desplazar a nivel de bit y cuya salida (bit saliente=0 ó 1) o lo que queda en el registro se puede controlar con saltos del tipo je=jz, jne=jnz, jc, jnc, jp o jnp.

El programa principal debe (no hay que pedir nada al usuario, solo sacar por pantalla):
Generar 10 números aleatorios que sean 2 ó 4 invocando al procedimiento NumAleatorio2o4 mediante un bucle y sacar el resultado tras cada generación por pantalla.
Generar 20 números aleatorios pares entre el valor 0 y (TOTALCELDAS-1)*2, invocando al procedimiento NumAleatorio (proporcionado) mediante un bucle y sacar el resultado tras cada generación por pantalla. Truco: Se puede generar el valor entre 0 y TOTALCELDAS-1 y después multiplicarlo por 2.

Combinar los procedimientos anteriores, almacenando en una posición aleatoria del vector, un número aleatorio (2 ó 4), siempre que dicha posición tenga un valor cero (vacía). Esto podría realizarse de forma opcional en otro procedimiento.

Invocar al procedimiento comprobarFinJuego para probar si funciona para el vector TableroJuego que tiene TOTALCELDAS posiciones. El procedimiento comprobarFinJuego comprobará las tres condiciones siguientes:

1. **Partida ganada**, cuando una posición del vector contenga el valor que fijes como tope; en este caso, se mostrará un mensaje de partida ganada.
2. **Partida perdida**, cuando todo el vector esté lleno de números distintos de cero y ninguno coincida con tope; en este caso, se mostrará un mensaje de partida perdida.
3. **Ninguna de las dos condiciones anteriores**, que indicará en el juego global que la partida continua (no sale nada por pantalla).

PROCEDIMIENTO NUEVO QUE CALCULA UN NÚMERO PSEUDO-ALEATORIO, ESTE PROCEDIMIENTO NO SE DEBE MODIFICAR (habrá que crear otro basado en él llamado NumAleatorio2o4):

```
;F: Calcula un valor aleatorio entre 0 y un valor maximo dado-1
;E: BL valor maximo (1...100)
;S: AH valor aleatorio calculado, entre (0...BL-1) y AL pierde su valor
NumAleatorio PROC
```

```
push cx
push dx

mov ah,2Ch ;interrupcion que recupera la hora actual del sistema  operativo
int 21h
;ch=horas cl=minutos dh=segundos dl=centesimas de segundo, 1/100 secs

xor ah,ah
mov al,dl
div bl

pop dx
pop cx
ret
NumAleatorio ENDP
```

3er enunciado de proyecto

Crear dos procedimientos a partir de este código base:

1. Un procedimiento de validación de comandos de entrada **comandoEntrada** para controlar la entrada de las distintas opciones del juego. El procedimiento sacará por pantalla una frase en la posición FILENTRADA, COLENTRADA (usar ColocarCursor) para pedir al usuario que introduzca un comando y pedirá al usuario un carácter por teclado sin eco (sin que aparezca en pantalla, usando el procedimiento proporcionado LeerTeclaSinEco) teniendo el siguiente comportamiento dependiendo del carácter introducido (solo debe funcionar con letras mayúsculas):

1. Si el carácter introducido es 'A', 'S', 'D' o 'W', terminar el procedimiento almacenando como parámetro de salida en AH el valor 0.
2. Si el carácter introducido es 'O', terminar el procedimiento almacenando como parámetro de salida en AH el valor 1.
3. Si el carácter introducido es 'E', terminar el procedimiento almacenando como parámetro de salida en AH el valor 2 y se imprimirá en pantalla la frase referente a partida finalizada en la posición FILENTRADA, COLENTRADA.
4. Para cualquier otro carácter, volver a pedir al usuario que introduzca de nuevo el carácter.

Este comportamiento se repetirá hasta que el carácter sea uno de los especificados previamente en esta lista.

Más abajo puedes ver el procedimiento **ColocarCursor**, que acepta una posición mediante dos parámetros de entrada (las variables fil y col) y posiciona el cursor de la pantalla en ese lugar.

2. Un procedimiento **ControlEntrada16_2048**. Este procedimiento pedirá al usuario introducir un número por teclado hasta que el valor introducido cumpla dos condiciones: 1) que sea potencia de dos y 2) que esté en el rango [16,2048] (ambos incluidos).

En caso de que el número introducido no cumpla ambas condiciones, se borrará la entrada realizada por el usuario y se volverá a pedir otro número, proceso que se repetirá hasta que el número sea correcto, momento en el que el procedimiento almacenará el número en la variable tope y terminará.

Para realizar el código se recomienda utilizar manipulación binaria. Te recomendamos que escribas la representación binaria de los números 16 y 2048 en binario y también otros números dentro y fuera del rango permitido y verás que puedes usar máscaras (usando test o and) para controlar el rango de números permitidos en este caso al ser potencias de 2.

Otra propiedad que puedes usar es que un número NUM potencia de dos distinto de 0 cumple siempre la condición: NUM & (NUM-1)=0 (la operación & puede realizarse con la instrucción test o and).

Para resolver este procedimiento también hace falta posicionar el cursor en pantalla:

```
FILMSJPOT y COLMSJPOT   ; posición msj introduce numero potencia de 2 tope con el que jugar
FILENTPOT y COLENTPOT    ; posición para pedir el numero tope con el que jugar
```

3. Realizar el programa principal que lance la ejecución de los 2 procedimientos anteriores.

Ejemplos de ejecución:

https://youtu.be/rn10HWAeRHg

```
Procedimiento ColocarCursor:

 ; E: Dos variables de tipo db "fil" y "col" que contendrán la posicion donde se debe colocar el cursor

  ColocarCursor PROC

    push ax

    push bx

    push dx

    mov ah,2
```

```
    mov bh,0
    mov dh, fil
    mov dl, col
    int 10h
    pop dx
    pop bx
    pop ax
    ret
ColocarCursor ENDP
```

4to enunciado de proyecto

Se indican en negrita todas las variables y nombres de procedimientos; y si además son mayúsculas, las equivalencias (EQU) de uso obligado. Se recomienda integrar en este código base (recomendamos guardar los códigos que os enlazamos con: botón derecho del ratón sobre el enlace>guardar enlace/vínculo como..., pues el navegador sobre escribe algunos valores ASCII) todos los procedimientos desde la sesión 6 incluida.

La entrega de esta sesión contendrá todo lo necesario hasta la petición de un comando al usuario. Esto incluye principalmente:

1. Procedimiento **CopiarVector**, que copiará un vector con un número de elementos dado de tamaño DW en otro vector del mismo tamaño. El procedimiento tendrá 3 registros como parámetros de entrada:

1. SI, que referencia al primer elemento del vector a copiar (origen),
2. DI, que referencia al primer elemento del vector al cual se quiere copiar los elementos (destino) y
3. CX, el número de elementos que se copiarán.

2. Procedimiento **BorrarVector** (muy similar en implementación al anterior), que pondrá a cero un vector con un número de elementos dado de tamaño DW; tendrá 2 registros como parámetros de entrada (a elegir por tí). Nota: al asignar valores pequeños (byte, <256) a una dirección de memoria ensamblador no chequea si el destino es de tipo byte, word, double,... y lo almacena en el menor espacio posible (1 byte).

Eso afecta al querer almacenar 0, 3, 20 en una variable de tamaño DW, pues si esta variable inicialmente tiene un valor grande o basura almacenada de una ejecución previa, al mover a esa variable el 0 como inmediato solo se almacenaría en el 1er byte, no en los dos bytes que conforman la variable. Para solucionarlo existe el operador word ptr, que permite especificar que se tiene que tratar el valor inmediato como si fuese un word, no un byte.

Por ejemplo:

mov [si], 0 ; funcionaría bien si el destino ocupa una sola posición de memoria (1 byte, DB), pero no funcionaría bien si el destino ocupa 2 posiciones de memoria (p.ej., 2 bytes, DW)
mov [si], word ptr 0 ;soluciona el problema anterior cuando el destino ocupa 2 posiciones de memoria (p.ej., 2 bytes, DW)

3. Funcionalidad de pantalla de inicio (puedes implementarla en el programa principal directamente o como un procedimiento al que invoques desde el programa principal):

a) Mostrar la pantalla de entrada al juego (variable PantallaInicio) en la fila 0 y columna 0.

b) Mostrar en la posición (FILMSJMODO,COLMSJMODO) el mensaje de petición del modo de juego (debug o no). Se pedirá un carácter al usuario usando el procedimiento LeerTeclaSinEco proporcionado de manera indefinida hasta que la tecla pulsada sea o 'S' o 'N', no siendo válidas tampoco las minúsculas.

c) En caso de seleccionar el modo debug ('S'), se hará uso del procedimiento CopiarVector para copiar el vector TableroJuegoDebug en TableroJuego. En caso de no seleccionar el modo debug ('N') se borrará el contenido del vector TableroJuego mediante el uso del procedimiento BorrarVector.

d) Invocar el procedimiento ControlEntrada16_2048 (ya implementado en la sesión 8).

4. Funcionalidades de la pantalla de juego (puedes implementarla en el programa principal directamente o como un procedimiento al que invoques desde el programa principal):

a) BorrarPantalla e imprimir pantTablero en la `posición (FILPANTALLAJ, COLPANTALLAJ).

b) Crear un procedimiento que pinte/Imprima el contenido del tablero de juego, es decir, tableroJuego.

Para ello se requerirá realizar un doble bucle for (p.ej., loop), transformando cada número contenido en cada posición de tableroJuego a cadena e imprimiéndola en la posición correspondiente... La posición del valor de la primera celda de tableroJuego debe pintarse en la posición (FILINICIOTAB, COLINICIOTAB) y las siguiente sumando a estos valores los desplazamientos correspondientes. Los ceros de tableroJuego no se imprimen en la pantalla de juego.

c) Invocar al procedimiento comandoEntrada (sesión 8) al que le iremos dando funcionalidad en futuras sesiones. https://youtu.be/5CaiebesC44

5to enunciado de proyecto

En esta práctica se realizará código relativo a la ejecución de los comandos, la comprobación de las condiciones de fin de juego y a la generación del número aleatorio 2 o 4:

Comando E: Salir del juego, tal y como ya veíamos en el vídeo de la sesión 9. Nota: no se permite realizar saltos entre procedimientos ni desde procedimientos al programa principal o viceversa como buena práctica de modularidad. Es decir, se corregirá el proyecto como si los saltos tuviesen localidad a nivel de procedimientos o programa principal.

Comando O: Volver a jugar un nuevo juego, es decir, volver a la pantalla de inicio a preguntar si se quiere jugar en esta ocasión en modo debug o no,... Nota: cuidado con las variables que haga falta resetear a un estado inicial

Comando A: (se deja para futuras entregas el comando D, W y S muy similares a este comando A): El comando A para mover a la izquierda realiza internamente dos acciones secuenciales:

> 1) calcular los valores que es necesario unir (aquellas parejas de valores iguales y distintos de cero que están seguidas o que entre ellas no tienen valores distintos de cero de izquierda a derecha). Nota: se pueden sumar los valores, x2 uno de ellos (incluso usando una instrucción lógica para ello)
>
> 2) desplazar los valores hacia la izquierda, "compactando" todos los valores distintos de cero de izquierda a derecha. Nota: es interesante ver el comportamiento requerido en todas las situaciones que se pueden dar, algunas de ellas recogidas en la vídeo demo.

Para simplificar el problema recomendamos utilizar "divide y venderás", transformando el problema de mover toda la matriz a la izquierda en un problema que mueva un vector a la izquierda, de modo que este procedimiento se invoque 1 vez por cada fila de la matriz. Es decir, moverIzq ([0,4,0,4]) debería darnos de resultado del primer paso [0,8,0,0] o bien [0,0,0,8] y el segundo paso [8,0,0,0] y podríamos llamar a este procedimiento moverIzq una vez por cada fila de la matriz (COLFILJUEGO). Revisa que tu solución se comporte correctamente en todas las situaciones que se pueden dar.

Tras la ejecución de cualquier comando de movimiento será necesario invocar al procedimiento que pinta el tablero de nuevo en pantalla (sesión 9).

Comprobar fin de juego: Después de cada movimiento se comprueban las condiciones de fin de juego, y si es necesario se mandan los mensajes correspondientes de partida ganada o partida perdida (procedimiento comprobarFinJuego de la sesión 7).

Generar un número aleatorio 2 o 4 y colocarlo en una posición del vector vacía generada también aleatoriamente:

Si el juego continua (aún hay elementos a valor cero en el vector tableroJuego), se genera un número aleatorio 2 o 4 y una posición aleatoria (0-30 par, que esté vacía) y se coloca el número en la posición del vector. El número generado debe aparecer también en la pantalla de juego en la posición correspondiente (Sesión 7).

https://youtu.be/cgNqDuEwYuU