Con el objetivo de afianzar los conceptos sobre entrada/salida estudiados en este tema, se proponen una serie de prácticas de laboratorio en las que se trabaja con algunos periféricos del PC. En concreto, en estas prácticas se trabajan los siguientes concepto: entrada/salida mapeada en memoria o independiente, programación de periféricos y del controlador de interrupciones, sincronización por encuesta o interrupción, tratamiento del vector de interrupción del PC, etc.

Para el desarrollo de estas prácticas se utilizará el lenguaje de programación C, en concreto el entorno de programación TurboC v3.0. Las funciones internas de acceso al hardware de la máquina, que están implementadas utilizando las librerías de TurboC, se proporcionan implementadas en los ficheros `Maquina.c` y `Maquina.h`. Por otra parte, para probar los ejecutables que se obtienen en estas prácticas es necesaria una instalación bajo sistema operativo MS-DOS. Para ello, podemos instalar en nuestra máquina una partición bajo sistema operativo MS-DOS o W98 (bajo el sistema operativo W98 también se puede arrancar la máquina en MS-DOS). Otra posibilidad es emular una máquina virtual en W98 sobre el sistema operativo de nuestra máquina, utilizando, por ejemplo, el software VirtualPC de Microsoft (u otro software de gestión de máquinas virtuales). Esta segunda opción será la que nosotros utilizaremos en las prácticas a realizar, en la definiremos una carpeta compartida entre la máquina real y la máquina virtual para poder mover y probar los ejecutables generados bajo TurboC en la máquina real.

A continuación presentaremos los objetivos de las 4 prácticas de laboratorio a realizar y la organización de los ficheros necesarios, incluyendo la información necesaria para el desarrollo de las mismas.

Práctica 1: pantalla en modo texto [carpeta panttext]

En esta práctica programaremos algunas funciones para el control de un periférico mapeado en memoria como la pantalla del PC. Así mismo, trabajaremos con el controlador del cursor en pantalla del PC, ejemplo de periférico no mapeado en memoria (entrada/salida independiente).

En lo relativo a la organización de los ficheros de esta primera práctica, además de las funciones ya proporcionadas en el fichero `Maquina.c` (y su correspondiente fichero de cabeceras `Maquina.h`), se utilizan los módulos `Pantalla.c` (`Pantalla.h`) y `PruPanT.c`. El fichero `PruPanT.c` contiene el programa principal y una serie de funciones para probar las rutinas a implementar para la gestión de la pantalla y del cursor.

El módulo `Pantalla.c` contiene las rutinas para la gestión de la pantalla y del cursor. Algunas de ellas se encuentran ya programadas. La práctica consiste en programar el resto de las rutinas de acuerdo al funcionamiento de dichos periféricos estudiado en la parte teórica del tema. Para compilar correctamente el proyecto, en el fichero `Pantalla.h` se proporcionan una serie de definiciones (directivas define) y las cabeceras de las funciones, declaradas como `extern`, para que estas funciones puedan ser utilizadas desde otras unidades de compilación.

En concreto, las funciones ya programadas son las siguientes:

```
unsigned short DirPant()
      //Esta función devuelve la dirección base de la pantalla del PC. El programa
      //principal llama a esta función y guarda en la variable InicioPant la
      //dirección base obtenida

void EscribeTexto(int fila, int col, char * texto, unsigned char atrib)
      //Esta función escribe el texto en pantalla a partir de la fila y columnas
      //indicadas. El atributo con el que se escribe el texto es el indicado
```

A continuación se indican las funciones que hay que implementar en esta primera práctica:

```
void EscribeCar(int fila, int col, unsigned char car, unsigned char atrib)
      //Escribe el carácter car en la pantalla, utilizando el atributo atrib, en la
      //posición indicada por los parámetros fila y col
void BorrarPantalla()
      //Borra la pantalla
```

```
void LeeCarPantalla(int fila, int col, unsigned char *car, unsigned char *atrib)
      //Lee el carácter y el abributo de la posición (fila,col) de la pantalla
void ScrollPantalla(int NumFilas)
      //Realiza un scroll de NumFilas en la pantalla, es decir, desplaza hacia arriba
      //NumFilas el contenido de la pantalla
void PonCursor(int x, int y)
      //Sitúa el cursor en la posición (x,y) de la pantalla
void LeePosCursor(int *x, int *y)
      //Devuelve en los parámetros la posición del cursor en la pantalla
void FormaCursor(int Inicio, int Final)
      //Forma el cursor entre la fila de Inicio y Final dentro de la matriz de pixels
      //de un carácter
```

Práctica 2: gestión del teclado por encuesta [carpeta tecenc]

El objetivo de esta práctica es programar la gestión de un periférico por encuesta y, para ello, utilizaremos el teclado del PC. La sincronización del teclado del PC se realiza normalmente por interrupción: cada vez que se pulsa o se libera una tecla, el controlador del teclado solicita una interrupción por la línea IRQ1. En esta práctica, inhibiremos las interrupciones del teclado actuando sobre el registro IMR del controlador de interrupciones y veremos cómo gestionamos el teclado por encuesta. Para ello, realizaremos una encuesta continua sobre el bit 1 del registro IRR del controlador de interrupciones. Si el bit 1 de ese registro se activa, sabremos que se ha pulsado o liberado una tecla. Hay que recordar que estas encuestas se realizan en el programa principal.

Además de los módulos `Maquina.c` (`Maquina.h`) y `Pantalla.c` (`Pantalla.h`) ya comentados en la práctica anterior, en esta práctica utilizaremos los módulos `TecEnc.c` (y su correspondiente fichero de cabecera `TecEnc.h` para poder realizar compilación separada) y `PruTecE.c`. El módulo `PruTecE.c` contiene el programa principal y la función que realiza la encuesta del teclado. Mientras que el programa principal ya se proporciona implementado, hay que implementar la función que realiza la encuesta del teclado, `unsigned char LeerTecladoEncuesta()`. Esta función devuelve el código ASCII de la tecla pulsada. Por otra parte, en el módulo `TecEnc.c` hay que implementar las siguientes funciones:

```
unsigned char LeerIRR()
      //Función que devuelve el contenido del registro IRR
unsigned char LeerRegDatosTeclado()
      //Función que devuelve el contenido del registro de datos del teclado
void InhibirIntTeclado()
      //Función que inhibe las interrupciones del teclado. Para ello, inhibe
      //previamente todas las interrupciones del PC (IF=0) y las recupera al final
      //(IF=1)
void DesinhibirIntTeclado()
      //Función que recupera las interrupciones del teclado. Para ello, inhibe
      //previamente todas las interrupciones del PC (IF=0) y las recupera al final
      //(IF=1)
void StrobeTeclado()
      //Realiza una secuencia de strobe en el bit 7 del reg. de control del teclado
```

Así mismo, supondremos definida la variable TABLA_ASCII[], que contiene la traducción del código de rastreo de la tecla pulsada a código ASCII. Este vector se indexa sólo con el código de rastreo MAKE, nunca utilizando el código BREAK. Esto es, en esa tabla podremos saber el código ASCII de una tecla en el momento en que se pulsa, nunca cuando soltamos la tecla.

Práctica 3: gestión del teclado por interrupción [carpeta tecint]

Tal y como hemos comentado, la sincronización habitual con el teclado es por interrupción a través de la línea IRQ1. En esta práctica codificaremos una sencilla rutina de atención al teclado que se ejecutará cada vez que se pulsa o se libera una tecla. Para ello, tendremos que modificar la entrada correspondiente al teclado en el vector de interrupción del PC, de tal forma que cuando se pulse/libere una tecla se ejecute nuestra rutina y no la que tiene el sistema cargada por defecto.

El funcionamiento de esta práctica es el siguiente: el programa principal (módulo `PruTecI.c` ejecuta la función `PasaTiempo` para simular que el PC está ejecutando alguna aplicación. La función PasaTiempo finaliza cuando se comprueba que la variable `Final` vale 1. Esta variable se pone a 1 cuando se pulse la tecla '`Q`'. Cada vez que se pulse/libere una tecla, se interrumpirá al programa principal para ejecutar la rutina de servicio `RutAtencionTec`. Esta rutina escribe en la pantalla el código ASCII de la tecla pulsada (y, como se ha comentado, se pulsa la tecla '`Q`' pone un 1 en la variable `Final`).

Además de los módulos `Maquina.c` (`Maquina.h`), `Pantalla.c` (`Pantalla.h`) y `PruTecI.c` (programa principal), en esta práctica intervienen los módulos: `TecInt.c` y `RutInt.c` (y sus correspondientes `Techint.h` y `RutInt.h`).

En el fichero `TecInt.c` se encuentran las rutinas relacionadas con la gestión del teclado y la rutina de servicio al mismo:

```
unsigned char LeerRegDatosTeclado()
      //Devuelve el contenido del registro de datos del teclado

void StrobeTeclado()
      // Realiza una secuencia de strobe en el bit 7 del reg. de control del teclado

void TratarCaracterLeido(unsigned char c)
      //Esta función ya está programada. Escribe en pantalla el código ASCII de la
      //tecla pulsada. Modifica la posición del cursor si se pulsan las teclas 'Y',
      //'G', 'B' o 'H'. En el caso de pulsar la tecla 'Q' pone un 1 en la variable
      //Final para finalizar el programa.

void interrupt RutAtencionTec()
      //Rutina de atención al teclado
```

El módulo `RutInt.c` contiene las funciones que gestionan el vector de interrupción del PC. Al comienzo del programa principal, se modifica la posición correspondiente en el vector a la línea de interrupción IRQ1 (entrada 9) para almacenar en esta posición la dirección de la rutina de atención programada (`RutAtencionTec`). Eso sí, antes de modificar dicha entrada, hay que almacenar su contenido para recuperar el vector de interrupción al finalizar la prueba. Las funciones a programar en este módulo son las siguientes:

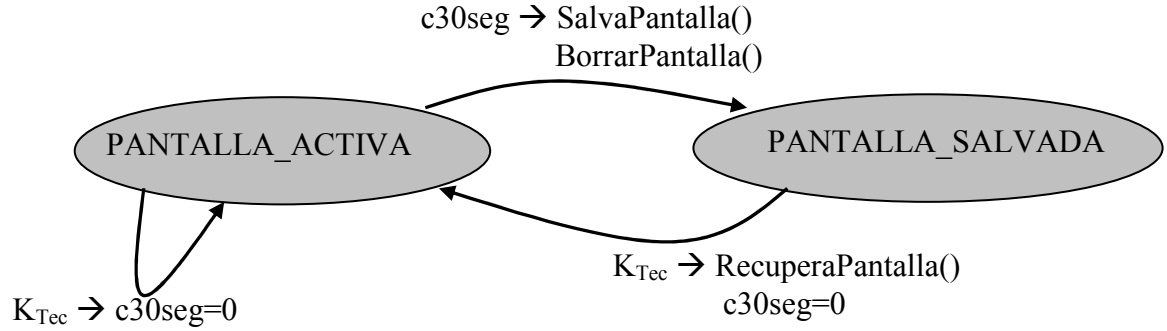
```
void CambiaVI(int nint, unsigned short IPNuevo, unsigned short IPNuevo,
              unsigned short * IPAnt, unsigned short * CSAnt)
      //Cambia la entrada "nint" del vector de interrupción con la dirección de
      //nuestra rutina de servicio. Inhibir las interrupciones al comienzo y
      //habilitarlas al final

void RecuperaVI(int nint, unsigned short IPAnt, unsigned short CSAnt)
      //Recupera el valor inicial para la entrada "nint" del vector de interrupción.
      //Inhibir las interrupciones al comienzo y habilitarlas al final

void Eoi()
      //End of Interruption: indica al controlador de interrupciones el final de una
      //interrupción
```

Finalmente, comentar que en el programa principal hay que completar los parámetros de las llamadas a las funciones que cambian y recuperan el vector de interrupción: `CambiaVI(parametros)` y `RecuperaVI(parametros)`.

Práctica 4: reloj y teclado [carpeta reloj]

Los objetivos de esta práctica son los siguientes: por una parte, aprender a programar eventos relacionados con el tiempo en el PC, y, por otra, desarrollar una práctica que controle más de un periférico, en este caso el reloj y el teclado. Para ello, vamos a programar el funcionamiento de un sencillo salvapantallas: si pasan 30 segundos sin pulsar una tecla, guardaremos el contenido de la pantalla del PC, borraremos la pantalla y recuperaremos dicho contenido al pulsar nuevamente una tecla. Además de ello, actualizaremos la hora cada segundo (actualizando la variable `HORA`, que se inicializa a una ficticia en el programa principal) y cada dos segundos escribiremos la hora en la pantalla. El programa principal finalizará al pasar un minuto desde que comenzó su ejecución.

En al figura podemos ver el autómata que refleja el funcionamiento de nuestro salvapantallas. El autómata tiene dos estados: `PANTALLA_ACTIVA`, que indica que no han pasado 30 segundos desde que se pulsó la última tecla, y `PANTALLA_SALVADA`, que indica que han pasado 30 segundos sin pulsar una tecla por lo que la pantalla se encuentra salvada. Para controlar en qué estado se encuentra el autómata, disponemos de la variable global `EstadoAutomata`. Así mismo, utilizaremos la variable global `c30seg` para contar si han pasado 30 segundos sin pulsar una tecla. Para ello, estando el autómata en el estado `PANTALLA_ACTIVA` la rutina de atención al reloj incrementará dicho temporizador cada segundo, mientras que cada vez que se pulsa una tecla, en la rutina de atención al teclado, se inicializará el temporizador. Si el temporizador `c30seg` llega a valer 30, esto quiere decir que han pasado 30 segundos sin pulsar una tecla. En este momento, en la rutina de atención al reloj habrá que guardar el contenido de la pantalla, borrar la pantalla y cambiar el estado del autómata a `PANTALLA_SALVADA`. La rutina de atención al teclado, en el momento en que se pulse una tecla, deberá recuperar el contenido de la pantalla, cambiando el estado del autómata a `PANTALLA_ACTIVA`.



Tal y como se ha comentado, la prueba debe finalizar al minuto del comienzo de su ejecución. Para sincronizar el programa principal y el reloj con este cometido, utilizaremos la variable global Final. Esta variable se pondrá a 1 cuando en la rutina de atención al reloj se detecte que el programa principal lleva un minuto en ejecución. En este momento, el programa principal, que espera en un bucle mientras la variable Final vale 0, finalizará.

En esta prueba, al comenzar la ejecución del programa principal, hay que cambiar las entradas correspondientes al teclado (entrada `9`) y reloj (entrada `0x1c`, tal y como se ha visto en la teoría) en el vector de interrupción del PC y recuperarlas al fina, antes de finalizar su ejecución.

En cuanto a los módulos que forman esta práctica, además de `Maquina.c` (`Maquina.h`), `Pantalla.c` (`Pantalla.h`) y `RutInt.c` (`RutInt.h`), también se utilizan los siguientes: `PruReloj.c` (módulo que contiene el programa principal), `TecInt.c` y `Reloj.c` (junto con sus correspondientes `TecInt.h` y `Reloj.h`). En el módulo `TecInt.c` podemos encontrar las rutinas que gestionan el teclado y la rutina de atención al teclado:

```
unsigned char LeerRegDatosTeclado()
      //Devuelve el contenido del registro de datos del teclado

void StrobeTeclado()
      // Realiza una secuencia de strobe en el bit 7 del reg. de control del teclado

void interrupt RutAtencionTec()
      //Rutina de atención al teclado
```

Por su parte, en el módulo `Reloj.c` podemos encontrar las funciones que gestionan la hora, la rutina que escribe la hora en pantalla y las rutinas que guardan y recuperan la pantalla.

```
void SalvaPantalla()
      //Guarda el contenido de la pantalla en la variable PantallaAnterior

void RecuperaPantalla()
      //Recupera la pantalla con el contenido de la variable PantallaAnterior

void ActualizarHora()
      //Actualiza la variable global HORA en un segundo

void EscribirHora(int fila, int col)
      //Escribe la hora (contenido de la variable HORA) en la posición indicada de la
      //pantalla

void interrupt RutAtencionReloj()
      //Rutina de atención al reloj
```

Por último, en el programa principal hay que completar los parámetros de las llamadas a las funciones que cambian y recuperan el vector de interrupción del PC, tanto para el teclado como para el reloj.