```c
/**
 * @file mi_io.c
 * @author Jose Ruben Cespedes Heredia
 * @date 9 Mar 2018
 * @brief Implementacion de las funciones que contiene la biblioteca propia.
 */

#include <dos.h>

#include "mi_io.h"

// Variables globales usadas en la implementacion
BYTE FG_COLOR = 7;  // Color del texto. Por defecto sera el 7, correspondiente al gris claro.
BYTE BG_COLOR = 0;  // Color de fondo. Por defecto sera el 0, correspondiente al negro.
int ALTO = 24;  // Alto de la consola
int ANCHO = 79; // Ancho de la consola

/**
 * @brief Devuelve el color actual
 *
 * La funcion usa las variables globales BG_COLOR y FG_COLOR para obtener el decimal
 * que usaremos para escribir un caracter con el color de fondo y el color de texto
 * actual.
 *
 * @return Un entero que representa el codigo decimal del color actual.
 */
int getcolor(){
    int color = BG_COLOR << 4 | FG_COLOR;

    return color;
}

/**
 * @brief Avanza el cursor
 *
 * La funcion tiene en cuenta donde se encuentra el cursor actualmente y se mueve
 * a la siguiente columna. En caso de haber llegado al final de la linea, avanzara
 * a la primera columna de la siguiente.
 */
void avanzar_cursor(){
    int linea = wherey();   // Linea actual
    int columna = wherex(); // Columna actual

    if(linea == ALTO && columna == ANCHO){  // Si estamos al final de la ultima linea, hacemos scrollup para poder seguir escribiendo
        scrollup(1, getcolor(), 0, 0, ALTO, ANCHO);
        linea = ALTO;
        columna = 0;
    } else if(columna < ANCHO){ // Si aun no hemos llegado al final del ancho, avanzamos una columna
        columna = columna + 1;
    } else {    // Si estamos al final del ancho, saltamos al principio de la siguiente linea
        linea = linea + 1;
        columna = 0;
    }

    gotoxy(columna, linea);
}

/**
 * @brief Indica si se ha pulsado alguna tecla
 *
 * La funcion llama a la subfuncion 1 de la interrupcion numero 16. Una
 * vez llamada, devuelve el registro cflag (zero flag), en el cual se
 * almacena un 0 si se ha pulsado una tecla y un 1 en caso contrario.
 *
 * @return devuelve el valor del registro zero flag
```

```
 */
int kbhit(){
    // Tenemos dos registros, uno donde almacenamos los parametros de la interrupcion
    // seleccionada y otro donde se guardan los datos devueltos por dicha interrupcion
    union REGS inregs, outregs;

    // En el registro ah introducimos la subfuncion de la rutina que queremos, en este caso, es "deteccion de tecla pulsada en bufer de teclado"
    inregs.h.ah = 0x01;

    // Mediante la funcion int86 llamamos a la interrupcion 0x16 que se asocia con la rutina de "gestion del teclado"
    int86(0x16, &inregs, &outregs);

    return outregs.x.cflag;
}

/**
 * @brief Indica la posición x actual del cursor
 *
 * La funcion llama a la subfuncion 3 de la interrupcion numero 10. Una
 * vez llamada, devuelve el registro dl, en el cual se
 * almacena la columna actual donde se encuentra el cursor.
 *
 * @return devuelve el valor del registro dl
 */
int wherex(){
    // Tenemos dos registros, uno donde almacenamos los parametros de la interrupcion
    // seleccionada y otro donde se guardan los datos devueltos por dicha interrupcion
    union REGS inregs, outregs;

    // En el registro ah introducimos la subfuncion de la rutina que queremos, en este caso, es "obtener tamaño y posicion del cursor"
    inregs.h.ah = 0x03;
    inregs.h.bh = 0x00;

    // Mediante la funcion int86 llamamos a la interrupcion 0x16 que se asocia con la rutina de "comunicacion con la tarjeta de video"
    int86(0x10, &inregs, &outregs);

    return outregs.h.dl;
}

/**
 * @brief Indica la posición y actual del cursor
 *
 * La funcion llama a la subfuncion 3 de la interrupcion numero 10. Una
 * vez llamada, devuelve el registro dh, en el cual se
 * almacena la fila actual donde se encuentra el cursor.
 *
 * @return devuelve el valor del registro dh
 */
int wherey(){
    // Tenemos dos registros, uno donde almacenamos los parametros de la interrupcion
    // seleccionada y otro donde se guardan los datos devueltos por dicha interrupcion
    union REGS inregs, outregs;

    // En el registro ah introducimos la subfuncion de la rutina que queremos, en este caso, es "obtener tamaño y posicion del cursor"
    inregs.h.ah = 0x03;
    inregs.h.bh = 0x00;

    // Mediante la funcion int86 llamamos a la interrupcion 0x16 que se asocia con la rutina de "comunicacion con la tarjeta de video"
    int86(0x10, &inregs, &outregs);

```

```
126     return outregs.h.dh;
127 }
128 
129 /**
130  * @brief Mueve el cursor a una posicion determinada
131  *
132  * La funcion llama a la subfuncion 2 de la interrupcion numero 10. Dicha subfuncion
133  * mueve el cursor a la posicion que se le indique
134  *
135  * @param x Indica la columna donde se quiere colocar el cursor
136  * @param y Indice la fila donde se quiere colocar el cursor
137  */
138 void gotoxy(int x, int y){
139     // Tenemos dos registros, uno donde almacenamos los parametros de la interrupcion
140     // seleccionada y otro donde se guardan los datos devueltos por dicha interrupcion
141     union REGS inregs, outregs;
142 
143     // En el registro ah introducimos la subfuncion de la rutina que queremos, en este caso, es "colocar el
cursor en una posicion determinada"
144     inregs.h.ah = 0x02;
145 
146     // En el registro dx introducimos la fila y la columna donde queremos colocar el cursor
147     inregs.h.bh = 0x00;
148     inregs.h.dh = y;
149     inregs.h.dl = x;
150 
151     // Mediante la funcion int86 llamamos a la interrupcion 0x16 que se asocia con la rutina de
"comunicacion con la tarjeta de video"
152     int86(0x10, &inregs, &outregs);
153 }
154 
155 
156 /**
157  * @brief Fija el aspecto del cursor, debe admitir tres valores: INVISIBLE, NORMAL y GRUESO
158  *
159  * La funcion llama a la subfuncion 1 de la interrupcion numero 10. Dicha subfuncion
160  * fija el aspecto del cursor en funcion de los numeros de linea que se le indiquen
161  *
162  * @param tipo_cursor Indica el tipo de cursor que desea establecer. Las opciones son INVISIBLE, NORMAL o
GRUESO
163  */
164 void setcursortype(enum types tipo_cursor){
165     // Tenemos dos registros, uno donde almacenamos los parametros de la interrupcion
166     // seleccionada y otro donde se guardan los datos devueltos por dicha interrupcion
167     union REGS inregs, outregs;
168 
169     // En el registro ah introducimos la subfuncion de la rutina que queremos, en este caso, es "fijar el
tamaño del cursor en modo texto"
170     inregs.h.ah = 0x01;
171 
172     switch(tipo_cursor){
173         case NORMAL:
174             inregs.h.ch = 010;
175             inregs.h.cl = 010;
176             break;
177         case GRUESO:
178             inregs.h.ch = 000;
179             inregs.h.cl = 010;
180             break;
181         case INVISIBLE:
182             inregs.h.ch = 010;
183             inregs.h.cl = 000;
184             break;
185     }
186 
187     // Mediante la funcion int86 llamamos a la interrupcion 0x10 que se asocia con la rutina de
```

```
"comunicacion con la tarjeta de video"
188       int86(0x10, &inregs, &outregs);
189   }
190
191   /**
192   * @brief Fija el modo de video deseado
193   *
194   * La funcion llama a la subfuncion 0 de la interrupcion numero 10. Dicha subfuncion
195   * fija el modo de video que se le indique.
196   *
197   * @param modo Es un caracter que indica el modo que se desea. Dicho caracter se debe traducir al decimal
adecuado
198   */
199   void setvideomode(BYTE modo){
200       // Tenemos dos registros, uno donde almacenamos los parametros de la interrupcion
201       // seleccionada y otro donde se guardan los datos devueltos por dicha interrupcion
202       union REGS inregs, outregs;
203       int modo_traducido;
204
205       // En el registro ah introducimos la subfuncion de la rutina que queremos, en este caso, es
"seleccionar modo de video"
206       inregs.h.ah = 0x00;
207
208       // Traducimos el caracter del modo en el decimal adecuado
209       modo_traducido = traducir_caracter(modo);
210
211       // Introducimos en el registro al el modo de video que deseamos establecer
212       inregs.h.al = modo_traducido;
213
214       ajustar_resolucion(modo_traducido);
215
216       // Mediante la funcion int86 llamamos a la interrupcion 0x10 que se asocia con la rutina de
"comunicacion con la tarjeta de video"
217       int86(0x10, &inregs, &outregs);
218   }
219
220   void ajustar_resolucion(int nuevo_modo){
221       switch(nuevo_modo){
222           case '0':
223               ANCHO = 39;
224               ALTO = 24;
225               break;
226           case '1':
227               ANCHO = 39;
228               ALTO = 24;
229               break;
230           case '2':
231               ANCHO = 79;
232               ALTO = 24;
233               break;
234           case '3':
235               ANCHO = 39;
236               ALTO = 24;
237               break;
238           case '4':
239               ANCHO = 319;
240               ALTO = 199;
241               break;
242           case '5':
243               ANCHO = 319;
244               ALTO = 199;
245               break;
246           case '6':
247               ANCHO = 639;
248               ALTO = 199;
249               break;
```

```
        case '7':
            ANCHO = 79;
            ALTO = 24;
            break;
        case '13':
            ANCHO = 319;
            ALTO = 199;
            break;
        case '14':
            ANCHO = 639;
            ALTO = 199;
            break;
        case '15':
            ANCHO = 639;
            ALTO = 349;
            break;
        case '16':
            ANCHO = 639;
            ALTO = 349;
            break;
        case '17':
            ANCHO = 639;
            ALTO = 479;
            break;
        case '18':
            ANCHO = 639;
            ALTO = 479;
            break;
        case '19':
            ANCHO = 319;
            ALTO = 199;
            break;
    }
}

/**
 * @brief Traduce un caracter al decimal adecuado para ser insertado en un registro
 *
 * La funcion usa un switch para relacionar cada caracter con el decimal adecuado
 *
 * @param caracter_original Es el caracter que deseamos traducir
 * @return un entero que representa el decimal del caracter traducido
 */
int traducir_caracter(BYTE caracter_original){
    int decimal;

    switch(caracter_original){
        case '0':
            decimal = 0;
            break;
        case '1':
            decimal = 1;
            break;
        case '2':
            decimal = 2;
            break;
        case '3':
            decimal = 3;
            break;
        case '4':
            decimal = 4;
            break;
        case '5':
            decimal = 5;
            break;
        case '6':
```

```
316             decimal = 6;
317             break;
318         case '7':
319             decimal = 7;
320             break;
321         case 'd':
322             decimal = 13;
323             break;
324         case 'e':
325             decimal = 14;
326             break;
327         case 'f':
328             decimal = 15;
329             break;
330         case '10':
331             decimal = 16;
332             break;
333         case '11':
334             decimal = 17;
335             break;
336         case '12':
337             decimal = 18;
338             break;
339         case '13':
340             decimal = 19;
341             break;
342         default:
343             decimal = 3;
344             break;
345     }
346 
347     return decimal;
348 }
349 
350 /**
351 * @brief Obtiene el modo de video actual
352 *
353 * La funcion llama a la subfuncion 15 de la interrupcion numero 10. Dicha subfuncion nos devuelve el
354 * en el registro al el modo de video usado actualmente
355 *
356 * @return un entero que representa el modo de video actual
357 * @note la relacion entre los enteros y los modos de video se pueden observar en el fichero main.c
358 */
359 int getvideomode(){
360     // Tenemos dos registros, uno donde almacenamos los parametros de la interrupcion
361     // seleccionada y otro donde se guardan los datos devueltos por dicha interrupciona
362     union REGS inregs, outregs;
363 
364     // En el registro ah introducimos la subfuncion de la rutina que queremos, en este caso, es "averiguar
el modo de video"
365     inregs.h.ah = 0xF;
366 
367     // Mediante la funcion int86 llamamos a la interrupcion 0x10 que se asocia con la rutina de
"comunicacion con la tarjeta de video"
368     int86(0x10, &inregs, &outregs);
369 
370     return outregs.h.al;
371 }
372 
373 /**
374 * @brief Modifica el color de primer plano con que se mostrarán los caracteres
375 *
376 * La funcion modifica la variable global FG_COLOR, la cual almacena el color en el que
377 * se escribe el texto por consola.
378 */
379 void textcolor(int color){
```

```
380     FG_COLOR = color;
381 }
382 
383 /**
384 * @brief Modifica el color de fondo con que se mostrarán los caracteres
385 *
386 * La funcion modifica la variable global BG_COLOR, la cual almacena el color en el que
387 * se escribe el fondo de los caracteres por consola.
388 */
389 void textbackground(int color){
390     BG_COLOR = color;
391 }
392 
393 /**
394 * @brief Borra toda la pantalla
395 *
396 * Esta funcion borra toda la pantalla haciendo que la funcion scrollup deslice la pantalla
397 * hacia arriba tantas lineas como alto tenga la consola. Ademas, situa el cursor en la posicion 0,0.
398 *
399 * @see scrollup(int lineas, int color, int lsi, int csi, int lid, int cid)
400 */
401 void clrscr(){
402     int color = BG_COLOR << 4 | FG_COLOR;
403 
404     scrollup(ALTO+1, color, 0, 0, ALTO, ANCHO);
405     gotoxy(0,0);
406 }
407 
408 /**
409 * @brief Borra una línea desde la posición actual del cursor hasta el final de la misma
410 *
411 * Esta funcion borra desde la posicion actual del cursor hasta el final de la linea (end of line, eof)
412 * haciendo uso de una de las caracteristicas de scrollup; y es que, cuando se llama a scrollup con el
413 * numero de lineas establecido en 0, se borra desde la (linea, columna) hasta la (linea, columna) que se
414 * desee
415 *
416 * @see scrollup(int lineas, int color, int lsi, int csi, int lid, int cid)
417 */
418 void clreol(){
419     int linea = wherey();   // Linea actual
420     int columna = wherex(); // Columna actual
421 
422     int color = BG_COLOR << 4 | FG_COLOR;
423 
424     scrollup(0, color, linea, columna, linea, ANCHO);
425 }
426 
427 /**
428 * @brief Desplaza toda la pantalla una línea hacia arriba
429 *
430 * La funcion llama a la subfuncion 6 de la interrupcion numero 10. Dicha subfuncion
431 * desplaza toda la pantalla hacia arriba el numero de lineas que se le indiquen. Si
432 * se le indican 0 lineas, borra la pantalla entre los puntos que se quieran (mediante lsi, csi, lid, cid.
433 *
434 * @param lineas Numero de lineas que se desean desplazar. 0 para borrar.
435 * @param color Color de relleno para los espacios en blanco. Los tres primeros bits indican el color de
fondo
436 * y los tres ultimos el color del texto.
437 * @param lsi Linea superior izquierda. Linea inicial desde la que se desea comenzar a desplazar/borrar.
438 * @param csi Columna superior izquierda. Columna inicial desde la que se desea comenzar a desplazar/borrar.
439 * @param lid Linea inferior derecha. Linea final hasta la que se desea desplazar/borrar.
440 * @param cid Columna inferior derecha. Columna final hasta la que se desea desplazar/borrar.
441 */
442 void scrollup(int lineas, int color, int lsi, int csi, int lid, int cid){
443     // Tenemos dos registros, uno donde almacenamos los parametros de la interrupcion
444     // seleccionada y otro donde se guardan los datos devueltos por dicha interrupcion
```

```
445     union REGS inregs, outregs;
446
447     // En el registro ah introducimos la subfuncion de la rutina que queremos, en este caso, es "desplazar
zona de pantalla hacia arriba"
448     inregs.h.ah = 0x06;
449
450     // En el resto de registros introducimos todos los parametros descritos anteriormente
451     inregs.h.al = lineas;
452     inregs.h.bh = color;
453     inregs.h.ch = lsi;
454     inregs.h.cl = csi;
455     inregs.h.dh = lid;
456     inregs.h.dl = cid;
457
458     // Mediante la funcion int86 llamamos a la interrupcion 0x10 que se asocia con la rutina de
"comunicacion con la tarjeta de video"
459     int86(0x10, &inregs, &outregs);
460
461     // Una vez realizado el desplazamiento de la pantalla, desplazamos el cursor un mismo numero de lineas
462     gotoxy(wherex(),wherey()-lineas);
463 }
464
465 /**
466  * @brief Desplaza toda la pantalla una línea hacia abajo
467  *
468  * La funcion llama a la subfuncion 7 de la interrupcion numero 10. Dicha subfuncion
469  * desplaza toda la pantalla hacia abajo el numero de lineas que se le indiquen. Si
470  * se le indican 0 lineas, borra la pantalla entre los puntos que se quieran (mediante lsi, csi, lid, cid.
471  *
472  * @param lineas Numero de lineas que se desean desplazar. 0 para borrar.
473  * @param color Color de relleno para los espacios en blanco. Los tres primeros bits indican el color de
fondo
474  * y los tres ultimos el color del texto.
475  * @param lsi Linea superior izquierda. Linea inicial desde la que se desea comenzar a desplazar/borrar.
476  * @param csi Columna superior izquierda. Columna inicial desde la que se desea comenzar a desplazar/borrar.
477  * @param lid Linea inferior derecha. Linea final hasta la que se desea desplazar/borrar.
478  * @param cid Columna inferior derecha. Columna final hasta la que se desea desplazar/borrar.
479  */
480 void scrolldown(int lineas, int color, int lsi, int csi, int lid, int cid){
481     // Tenemos dos registros, uno donde almacenamos los parametros de la interrupcion
482     // seleccionada y otro donde se guardan los datos devueltos por dicha interrupcion
483     union REGS inregs, outregs;
484
485     // En el registro ah introducimos la subfuncion de la rutina que queremos, en este caso, es "desplazar
zona de pantalla hacia abajo"
486     inregs.h.ah = 0x07;
487
488     inregs.h.al = lineas;
489     inregs.h.bh = color;
490     inregs.h.ch = lsi;
491     inregs.h.cl = csi;
492     inregs.h.dh = lid;
493     inregs.h.dl = cid;
494
495     // Mediante la funcion int86 llamamos a la interrupcion 0x10 que se asocia con la rutina de
"comunicacion con la tarjeta de video"
496     int86(0x10, &inregs, &outregs);
497 }
498
499 /**
500  * @brief Escribe un caracter en pantalla con el color indicado actualmente
501  *
502  * La funcion llama a la subfuncion 9 de la interrupcion numero 10. Dicha subfuncion
503  * imprime un caracter por pantalla con el color indicado y el numero de veces que se desee.
504  * Finalmente avanza el cursor.
505  *
```

```
506  * @param caracter Caracter que se desea imprimir
507  * @param repeticiones Numero de veces que se desea imprimir el caracter
508  * @see avanzar_cursor
509  */
510  void cputchar(const char caracter, int repeticiones){
511      // Tenemos dos registros, uno donde almacenamos los parametros de la interrupcion
512      // seleccionada y otro donde se guardan los datos devueltos por dicha interrupcion
513      union REGS inregs, outregs;
514      int color = BG_COLOR << 4 | FG_COLOR;
515
516      // En el registro ah introducimos la subfuncion de la rutina que queremos, en este caso, es "escribir
un caracter en pantalla"
517      inregs.h.ah = 0x09;
518
519      // En el resto de registros se introducen el caracter, el color y el numero de veces que se desea
imprimir el caracter
520      inregs.h.al = caracter;
521      inregs.h.bl = color;
522      inregs.h.bh = 0x00;
523      inregs.x.cx = repeticiones;
524
525      // Mediante la funcion int86 llamamos a la interrupcion 0x10 que se asocia con la rutina de
"comunicacion con la tarjeta de video"
526      int86(0x10, &inregs, &outregs);
527
528      // Finalmente avanzamos el cursor
529      avanzar_cursor();
530  }
531
532  /**
533  * @brief Obtiene el caracter de teclado y lo muestra en pantalla
534  *
535  * La funcion llama a la subfuncion 0 de la interrupcion numero 10. Dicha subfuncion
536  * lee un caracter desde el teclado y lo almacena en el registro al. En caso de ser
537  * una tecla especial queda almacenado en el ah.
538  *
539  * @return Decimal que representa el caracter introducido por teclado
540  */
541  int getche(){
542      // Tenemos dos registros, uno donde almacenamos los parametros de la interrupcion
543      // seleccionada y otro donde se guardan los datos devueltos por dicha interrupcion
544      union REGS inregs, outregs;
545      int caracter;
546
547      // En el registro ah introducimos la subfuncion de la rutina que queremos, en este caso, es "leer un
caracter desde el teclado"
548      inregs.h.ah = 0x00;
549
550      // Mediante la funcion int86 llamamos a la interrupcion 0x16 que se asocia con la rutina de "gestion
del teclado"
551      int86(0x16, &inregs, &outregs);
552
553      // Si al es 0 significa que es un caracter especial. 1 en caso contrario
554      if(outregs.h.al != 0)
555          caracter = outregs.h.al;
556      else
557          caracter = outregs.h.ah;
558
559      cputchar(caracter, 1);
560
561      return caracter;
562  }
563
564
565  /**
566  * @brief Obtiene el caracter de teclado
```

```
 *
 * La funcion llama a la subfuncion 0 de la interrupcion numero 10. Dicha subfuncion
 * lee un caracter desde el teclado y lo almacena en el registro al. En caso de ser
 * una tecla especial queda almacenado en el ah.
 *
 * @return Decimal que representa el caracter introducido por teclado
 */
int getch(){
    // Tenemos dos registros, uno donde almacenamos los parametros de la interrupcion
    // seleccionada y otro donde se guardan los datos devueltos por dicha interrupcion
    union REGS inregs, outregs;
    int caracter;

    // En el registro ah introducimos la subfuncion de la rutina que queremos, en este caso, es "leer un caracter desde el teclado"
    inregs.h.ah = 0x00;

    // Mediante la funcion int86 llamamos a la interrupcion 0x16 que se asocia con la rutina de "gestion del teclado"
    int86(0x16, &inregs, &outregs);

    // Si al es 0 significa que es un caracter especial. 1 en caso contrario
    if(outregs.h.al != 0)
        caracter = outregs.h.al;
    else
        caracter = outregs.h.ah;

    return caracter;
}

/**
 * @brief Imprime una cadena de caracteres en pantalla
 *
 * La funcion hace uso de la funcion cputchar para ir imprimiendo por pantalla
 * uno a uno los caracteres de la cadena que se desea imprimir. Lo que hacemos es que
 * vamos leyendo los caracteres de la cadena introducida hasta llegar al caracter
 * de fin de linea ('\0'). Si el caracter es distinto del de salto de linea (\n)
 * lo imprimimos y avanzamos al siguiente caracter. Si el caracter leido es el de
 * salto de linea, situamos el cursor al inicio de la siguiente linea y avanzamos
 * el indice para leer el siguiente caracter
 *
 * @param cadena Cadena que se desea imprimir
 * @return decimal que representa al ultimo caracter leido
 */
int cputs(const char * cadena){
    int i = 0;  // Indice para recorrer la cadena
    int linea = wherey();   // Linea actual
    int columna = wherex(); // Columna actual

    while(cadena[i] != '\0'){
        if(cadena[i] != '\n')
            cputchar(cadena[i],1);
        else {
            columna = 0;

            if(linea < ALTO)
                linea = linea + 1;
            else {
                scrollup(1, getcolor(), 0, 0, ALTO, ANCHO);
                linea = ALTO;
            }

            gotoxy(columna, linea);
        }

        i = i+1;    // Avanzamos el indice
```

```
631     }
632
633     return cadena[i];
634
```