

PRÁCTICA 1

ENTRADA/SALIDA UTILIZANDO INTERRUPCIONES CON LENGUAJE C



RUBÉN HIDALGO TROYANO
MARÍA MEGÍAS MOYANO

ÍNDICE

| | |
|---|----|
| 1. Introducción | 3 |
| 2. pausa() | 3 |
| 3. cgetchar() | 4 |
| 4. cputchar() | 4 |
| 5. setcursortype() | 4 |
| 6. setvideomode() (Tamaño de pantalla) | 5 |
| 7. getvideomode() (Tamaño de pantalla) | 5 |
| 8. imprimir_modo_video() | 6 |
| 9. get_text_color() | 6 |
| 10. get_text_background_color() | 7 |
| 11. textcolor() | 7 |
| 12. textbackground() | 8 |
| 13. cambiartextoyfondo() | 8 |
| 14. restaurarcolores() | 9 |
| 15. cputcharcolortexto() | 9 |
| 16. cputcharcolorfondo() | 9 |
| 17. cputcharcolortextoyfondo() | 10 |
| 18. gotoxy() | 10 |
| 19. gotoposicioncorrecta() | 10 |
| 20. getX() | 11 |
| 21. getY() | 11 |
| 22. clrscr() | 12 |
| 23. clrscr2() | 12 |
| 24. mostrarcolores() (Función auxiliar) | 12 |
| 25. pedircolorfondo() | 13 |
| 26. pedircolortexto() | 13 |
| 27. modovideo() (Texto-Gráfico) | 13 |
| 28. pixel() | 14 |
| 29. rectangulo_vacio() | 14 |
| 30. rectangulo_lleno() | 15 |
| 31. dibujar_conejo() | 15 |
| 32. dibujar_gato() | 15 |
| 33. dibujar_pollo() | 16 |
| 34. dibujar_bob_esponja() | 16 |
| 35. dibujar_pez() | 16 |
| 36. dibujar_rombo() | 17 |
| 37. dibujar_cuadrado() | 17 |
| 38. ejecución del programa | 17 |

1. Introducción

Para cada función, pondremos dos capturas de pantalla:

- ★ La primera mostrará la implementación de la función en c.
- ★ La segunda mostrará un fragmento del main() en el que ejecutamos dicha función.

Para cada función, usaremos únicamente las bibliotecas “stdio.h” y “dos.h”. Además, definimos la variable del tipo *unsigned char* como “BYTE” para cuando debemos llevar a cabo un cambio entre modo texto (en el que BYTE tendrá valor 3) y modo gráfico para dibujar con píxeles (en el que BYTE tendrá valor 4).

```
#include <stdio.h>
#include <dos.h>
#define BYTE unsigned char

BYTE MODOTEXTO = 3;
BYTE MODOGRAFICO = 4;
```

Tras las funciones, se mostrará el funcionamiento del programa tras compilarlo y ejecutarlo usando capturas de pantalla en DosBox para que todo sea más visual y entendible.

APUNTES:

- ★ Para una mejor fluidez de las definiciones, a la hora de nombrar una función llamada `ejemplo_funcion(int _)`, la llamaremos simplemente `ejemplo_funcion()`, de forma que sea más fácil su comprensión. Mencionaremos sus distintos parámetros de entrada durante su definición, pudiendo usar las capturas de pantalla para su visualización.
- ★ Siempre que una función devuelve un valor, lo hace mediante el registro `outregs.h.XY`, indicando X el registro e Y si se trata de la parte baja. Por ello, para decir dónde se encuentra el valor a devolver, haremos mención solamente al registro y su parte.

2. pausa()

La función “`mi_pausa()`” sirve para llamar a una interrupción del S.O. cuya función es detener el programa hasta insertar un carácter.

Para ello, debemos usar la función en número hexadecimal 8h (`inregs.h.ah=8`) de la interrupción 21h (`int86(0x21, &inregs, &outregs)`).

```
void mi_pausa(){
    union REGS inregs, outregs;
    inregs.h.ah = 8;
    int86(0x21, &inregs, &outregs);
}

printf("\nCursor invisible: ");
setcursortype(0);
mi_pausa();

printf("\nCursor grueso: ");
setcursortype(2);
mi_pausa();

printf("\nCursor normal: ");
setcursortype(1);
mi_pausa();
```

3. cgetchar()

En la función “mi_getchar()”, usamos la función 1h de la interrupción 21h para pedir al usuario que nos dé un carácter desde teclado, quedando dicho carácter almacenado en la parte alta del registro AX.

En nuestro ejemplo de implementación en main(), dicho valor quedará guardado en la variable de tipo entero llamada tmp.

```
int mi_getchar(){
    union REGS inregs, outregs;
    int character;

    inregs.h.ah = 1;
    int86(0x21, &inregs, &outregs);

    character = outregs.h.al;
    return character;
}

int tmp;

printf("\nPulsa una tecla... ");
tmp = mi_getchar();
```

4. cputchar()

Dentro de la función “mi_putchar()”, usamos la función 2h de la interrupción 21h para pedir mostrar por pantalla un carácter recibido como parámetro de entrada con el color de texto y fondo actuales. Para ello, asignamos el carácter de entrada a la parte baja del registro DX.

En este ejemplo de main(), se mostrará el valor de la variable tmp almacenado anteriormente con mi_getchar().

```
void mi_putchar(char c){
    union REGS inregs, outregs;

    inregs.h.ah = 2;
    inregs.h.dl = c;
    int86(0x21, &inregs, &outregs);
}

printf("\nHas pulsado: ");
mi_putchar( (char)tmp );
```

5. setcursortype()

Con la función “setcursortype()”, pedimos un entero como parámetro de entrada. La función usa la función 1h de la interrupción 10h para modificar el cursor de pantalla. En este caso, dependiendo del valor del entero, podemos obtener los siguientes tipos de cursor:

- Cursor invisible: El entero posee valor 0, la parte alta del registro CX tendrá el valor 010 y la parte baja el valor 000.
- Cursor grueso: El entero vale 2, por lo que la parte alta del registro CX se configura con 000 y la parte baja con 010.
- Cursor normal: El entero es 1, por lo que la parte alta y baja del registro CX toman el valor 010.

```

void setcursortype(int tipo_cursor){
    union REGS inregs, outregs;
    inregs.h.ah = 0x01;
    switch(tipo_cursor){
        case 0: //invisible
            inregs.h.ch = 010;
            inregs.h.cl = 000;
            break;
        case 1: //normal
            inregs.h.ch = 010;
            inregs.h.cl = 010;
            break;
        case 2: //grueso
            inregs.h.ch = 000;
            inregs.h.cl = 010;
            break;
    }
    int86(0x10, &inregs, &outregs);
}

printf("\nCursor invisible: ");
setcursortype(0);
mi_pausa();

printf("\nCursor grueso: ");
setcursortype(2);
mi_pausa();

printf("\nCursor normal: ");
setcursortype(1);
mi_pausa();

```

6. setvideomode() (Tamaño de pantalla)

En la función 'mi_modos_video()', usamos la función 0h de la interrupción 10h, asignando una variable de entrada de tipo unsigned char a la parte alta del registro AX, indicando de esta forma a qué modo de vídeo queremos cambiar.

En este ejemplo de uso en main(), primero pasamos a modo 40x25 y posteriormente regresamos al modo estipulado por defecto, 80x25.

```

void mi_modos_video(unsigned char modo){
    union REGS inregs, outregs;
    inregs.h.ah = 0x00;
    inregs.h.al = modo;
    int86(0x10, &inregs, &outregs);
    return;
}

mi_modos_video(1); // modo 1 -> 40x25 (Letras grandes)
printf("\nEn C. Pulsa una tecla... ");
mi_pausa();
mi_modos_video(3); // modo 3 -> 80x25 (Letras pequeñas)

```

7. getvideomode() (Tamaño de pantalla)

La finalidad de la función 'mi_get_modos_video()' es obtener el tamaño del modo de vídeo que estamos utilizando en ese momento. Para llevar a cabo su cometido, utilizará la función 0h de la interrupción 10h para obtener el valor de la parte alta del registro AX.

En el ejemplo mostrado de implementación en main(), almacenamos el valor obtenido gracias a la función en la variable de tipo unsigned char llamada modo_video o simplemente no la almacenamos y lo usamos como parámetro de entrada de la función imprimir_modos_video().

```

unsigned char mi_get_modos_video() {
    union REGS inregs, outregs;

    inregs.h.ah = 0x0F;
    int86(0x10, &inregs, &outregs);

    return outregs.h.al;
}

mi_modos_video(1);
printf("\nEn C. Pulsa una tecla... ");
modo_video = mi_get_modos_video();
imprimir_modos_video(modo_video);
mi_pausa();
mi_modos_video(3);
imprimir_modos_video(mi_get_modos_video());
printf("\nEn C. Pulsa una tecla...\n");
mi_pausa();

```

8.imprimir_modovideo()

La función 'imprimir_modovideo()' actúa como auxiliar de mi_get_modovideo(), ya que interpreta el modo de vídeo obtenido y lo traduce a su correspondiente resolución de pantalla.

En otras palabras, toma el valor devuelto por mi_get_modovideo() y, mediante una estructura switch, muestra en pantalla la resolución correspondiente a ese modo de vídeo.

```
mi_modovideo(1);
printf("\nEn C. Pulsa una tecla... ");

modovideo = mi_get_modovideo();

imprimir_modovideo(modovideo);

mi_pausa();

mi_modovideo(3);

imprimir_modovideo(mi_get_modovideo());

printf("\nEn C. Pulsa una tecla...\n");

mi_pausa();

void imprimir_modovideo(unsigned char modovideo) {
    switch (modovideo) {
        case 0x01:
            printf("Modo de video: 40x25\n");
            break;
        case 0x03:
            printf("Modo de video: 80x25\n");
            break;
        case 0x07:
            printf("Modo de video: 80x25 Blanco y Negro\n");
            break;
        case 0x13:
            printf("Modo de video: 320x200 con 256 colores\n");
            break;
        default:
            printf("Modo de video desconocido: %d\n", modovideo);
            break;
    }
}
```

9. get_text_color()

La función 'get_text_color()' obtiene el color actual del texto. Para ello, utiliza la función 8h de la interrupción 10h indicando que se llevará a cabo en esta página de vídeo, la principal (parte alta del registro BX a 0). Devuelve el valor contenido en la parte alta del registro AX, a la cuál se le aplica una máscara para obtener solamente los 4 últimos dígitos, debido a que los 4 primeros representan el color de fondo.

Esta máscara, 0x0F, expresa una función AND del valor del registro ah con el valor 00001111, de forma que tan sólo nos quedaremos con los 4 últimos dígitos que representan el color de texto, poniendo a 0 aquellos que representan al color de fondo. Ejemplo de máscara: 1010 1101 & 0000 1111 = 0000 1101

```
unsigned char get_text_color() {
    union REGS inregs, outregs;
    inregs.h.ah = 0x08;
    inregs.h.bh = 0x00;
    int86(0x10, &inregs, &outregs);
    return (outregs.h.ah & 0x0F);
}

cambiarColor(4);
printf("Texto en color rojo\n");
mi_pausa();
color_texto = get_text_color();
color_fondo = get_text_background_color();
printf("Color de TEXTO actual: %d\n", color_texto);
printf("Color de FONDO actual: %d\n", color_fondo);
mi_pausa();
```

10. get_text_background_color()

La función ‘get_text_background_color()’ obtiene el color actual del fondo de texto. Para ello, utiliza la función 8h de la interrupción 10h indicando que se llevará a cabo en esta página de vídeo, la principal (parte alta del registro BX a o). Devuelve el valor contenido en la parte alta del registro AX, a la cuál se le aplica un desplazamiento de 4 dígitos para obtener solamente los 4 primeros dígitos, debido a que los 4 últimos representan el color de texto.

Este desplazamiento, >>4, mueve 4 veces todos los bits a la derecha, eliminando el bit más a la derecha y añadiendo un 0 a la izquierda, de forma que únicamente nos queden los primeros 4 bits que representan al fondo y eliminando los 4 bits que indican el color de texto. Ejemplo: 1010 1101 >> 2 = 00101 011

1010 1101 >> 4 = 0000 1010

```
unsigned char get_text_background_color() {  
    union REGS inregs, outregs;  
    inregs.h.ah = 0x08;  
    inregs.h.bh = 0x00;  
    int86(0x10, &inregs, &outregs);  
    return (outregs.h.ah >> 4);  
}  
  
cambiarColor(4);  
printf("Texto en color rojo\n");  
mi_pausa();  
color_texto = get_text_color();  
color_fondo = get_text_background_color();  
printf("Color de TEXTO actual: %d\n", color_texto);  
printf("Color de FONDO actual: %d\n", color_fondo);  
mi_pausa();
```

11. textcolor()

La función ‘cambiarColorTexto()’, utiliza la función 9h de la interrupción 10h para cambiar el color de las letras pero que el fondo sea negro (debido a que al pasar un número entre 0 y 15, los 4 primeros bits que representan el color de fondo siempre estarán a 0). Indicamos en el registro AL (parte alta de AX) que el carácter a escribir sea un espacio, no para que solamente se imprima ese espacio sino para evitar que se llene todo de caracteres basura al hacer el cambio de color.

Indicamos el color a la parte baja de BX.

Por último, indicar que aquí y en otras funciones (CambiarColorFondo(), CambiarColorTextoyFondo() y clrscr2()) podríamos no usar inregs y outregs como tal, al no haber uso de outregs, pudiendo llevar a cabo esta implementación más simple de una sola variable:

```
void cambiarColorFondo(int colorFondo) {  
    union REGS r;  
  
    r.h.ah = 0x09;  
    r.h.al = ' ';  
    r.h.bh = 0;  
    r.h.bl = (colorFondo << 4) | 0;  
    int86(0x10, &r, &r);  
}
```

Sin embargo, para que todas las funciones con interrupciones BIOS sean más homogéneas, todas serán implementadas con el uso de inregs y outregs.

```

void cambiarColorTexto(int color) {
    union REGS inregs, outregs;

    inregs.h.ah = 0x09;
    inregs.h.al = ' ';
    inregs.h.bh = 0;
    inregs.h.bl = color;

    int86(0x10, &inregs, &outregs);
}

cambiarColorTexto(4);
printf("Texto en color rojo\n");
mi_pausa();

```

12. textbackground()

La función 'cambiarColorTexto()', utiliza la función 9h de la interrupción 10h para cambiar el color de fondo y que las letras sean negras (suponemos que no se va a cambiar el fondo de nuevo a negro, ya que para ello está la función restaurarColores()). Indicamos en el registro AL (parte alta de AX) que el carácter a escribir sea un espacio, no para que solamente se imprima ese espacio sino para evitar que se llene todo de caracteres basura al hacer el cambio de color.

Indicamos el color a la parte baja de BX, llevando a cabo un desplazamiento de 4 bits a la izquierda ya que el color de fondo es representado por los 4 primeros bits, quedándose los últimos 4 con ceros para que el color de texto sea negro.

```

void cambiarColorFondo(int colorFondo) {
    union REGS inregs, outregs;

    inregs.h.ah = 0x09;
    inregs.h.al = ' ';
    inregs.h.bh = 0;
    inregs.h.bl = (colorFondo << 4);
    int86(0x10, &inregs, &outregs);
}

cambiarColorFondo(5);
printf("Fondo en color magenta\n");
mi_pausa();

```

13. cambiartextoyfondo()

La función 'cambiarColorTexto()', utiliza la función 9h de la interrupción 10h para cambiar el color de fondo y el color de texto.

La función toma el color pasado en su segundo parámetro de entrada y lo inserta en la parte baja del registro BX, haciendo el desplazamiento de cuatro bits para que cambie el color de fondo y los últimos 4 bits queden a 0. A continuación, hace una operación OR con el valor del primer parámetro de entrada, es decir, con el entero que representa el color del texto (cuyos 4 primeros bits están a 0), quedando así el color de fondo en los primeros 4 bits y el de texto en los últimos 4.

```

void cambiarColorTextoYFondo(int colorTexto, int colorFondo) {
    union REGS inregs, outregs;

    inregs.h.ah = 0x09;
    inregs.h.al = ' ';
    inregs.h.bh = 0;
    inregs.h.bl = (colorFondo << 4) | colorTexto;

    int86(0x10, &inregs, &outregs);
}

cambiarColorTextoYFondo(9, 7);
printf("Texto en color azul claro con fondo blanco\n");
mi_pausa();

```


14. restaurarcolores()

La función `restaurarColores()` restablece los colores originales de texto y fondo llamando a `cambiarColorTextoYFondo()`, la cual utiliza la función `9h` de la interrupción `10h` de BIOS para modificar los atributos de color en pantalla. Para ello, toma los valores originales del color del texto y el fondo, los combina en un solo byte (donde los 4 bits más altos representan el fondo y los 4 más bajos el texto) y los aplica a la pantalla.

```
void restaurarColores(unsigned char color_texto_original, unsigned char color_fondo_original) {
    cambiarColorTextoYFondo(color_texto_original, color_fondo_original);
    printf("Colores restaurados a los valores originales.\n");
}

restaurarColores(color_texto_original, color_fondo_original);
mi_pausa();
```

15. cputcharcolortexto()

La función `cputchar_color_texto()` permite imprimir un carácter en pantalla con un color de texto específico sin modificar el color de fondo. Para ello, utiliza la interrupción `10h` de BIOS con la función `09h`, que permite escribir un carácter con atributos de color. Se pasa el carácter en `AL`, el color en `BL`, y `CX` se establece en 1 para imprimir el carácter una única vez.

```
void cputchar_color_texto(char c, int colorTexto) {
    union REGS r;
    r.h.ah = 0x09;
    r.h.al = c;
    r.h.bh = 0;
    r.h.bl = colorTexto;
    r.x.cx = 1;
    int86(0x10, &r, &r);
}

printf("Caracter en rojo: ");
cputchar_color_texto('R', 4);
printf("\n");
```

16. cputcharcolorfondo()

La función `cputchar_fondo()` imprime un solo carácter en pantalla con un color de fondo específico, manteniendo el texto en blanco. Para lograrlo, utiliza la interrupción `10h` de BIOS con la función `09h`, que permite escribir un carácter con atributos de color. El color de fondo se almacena en los 4 bits más altos del byte de atributo (`BL`), desplazándolo 4 bits a la izquierda, mientras que los 4 bits bajos se configuran en `0x0F` para que los últimos 4 bits se traduzcan en 15 y el texto sea siempre blanco brillante.

```
void cputchar_fondo(char c, int colorFondo) {
    union REGS r;
    r.h.ah = 0x09;
    r.h.al = c;
    r.h.bh = 0;
    r.h.bl = (colorFondo << 4) | 0x0F;
    r.x.cx = 1;
    int86(0x10, &r, &r);
}

printf("Caracter con fondo azul: ");
cputchar_fondo('F', 1);
printf("\n");
```

17. cputcharcolortextoyfondo()

La función `cputchar_color_fondo()` imprime un carácter en pantalla con un color de texto y un color de fondo personalizados. Para ello, utiliza la interrupción 10h de BIOS con la función 09h, que permite escribir un carácter con atributos de color. El color de fondo se almacena en los 4 bits más altos del byte de atributo (BL), desplazándose 4 bits a la izquierda, mientras que el color de texto se modifica usando una máscara en el color pasado por parámetro, quedándonos con los 4 bits menos significativos, y aplicando una operación OR con el color desplazado de fondo.

```
void cputchar_color_fondo(char c, int colorTexto, int colorFondo) {
    union REGS r;
    r.h.ah = 0x09;
    r.h.al = c;
    r.h.bh = 0;
    r.h.bl = (colorFondo << 4) | (colorTexto & 0x0F);
    r.x.cx = 1;
    int86(0x10, &r, &r);
}

printf("Caracter amarillo sobre fondo magenta: ");
cputchar_color_fondo('Z', 14, 5);
printf("\n");
mi_pausa();
```

18. gotoxy()

La función `gotoxy()` mueve el cursor a una posición específica en la pantalla de texto en modo DOS utilizando la interrupción 10h de BIOS con la función 02h. Antes de ejecutarla, verifica que x e y estén dentro de los límites válidos (1-80 para columnas y 1-25 para filas), en caso contrario las ajusta a su límite más cercano, ajusta las coordenadas a base cero (internamente, las columnas van de 0 a 79 y las filas de 0 a 24) y las almacena en los registros. Esto permite un control preciso sobre la ubicación del cursor, facilitando la organización de la salida en pantalla.

```
void gotoxy(int x, int y) {
    union REGS inregs, outregs;

    if (x < 1) x = 1;
    if (x > 80) x = 80;
    if (y < 1) y = 1;
    if (y > 25) y = 25;

    inregs.h.ah = 0x02;
    inregs.h.bh = 0x00;
    inregs.h.dh = y - 1;
    inregs.h.dl = x - 1;

    int86(0x10, &inregs, &outregs);
}

printf("Introduce la columna (1-80): ");
scanf("%d", &x);
printf("Introduce la fila (1-25): ");
scanf("%d", &y);

gotoxy(x, y);
```

19. gotoposicioncorrecta()

La función `go_to_posicion_correcta()` mueve el cursor a la esquina inferior izquierda de la pantalla de texto en modo DOS, posicionándolo en la fila 24 y columna 0, utilizando la interrupción 10h de BIOS con la función 02h. Los valores de la fila y columna se almacenan en los registros antes de ejecutar la interrupción. La fila 24, usando base cero, representa la última línea visible de una pantalla de 25 líneas, y la columna 0 coloca el cursor al inicio de esa línea, lo que es útil para gestionar la disposición del texto en la consola o imprimir mensajes importantes.

```

void go_to_posicion_correcta() {
    union REGS inregs, outregs;

    inregs.h.ah = 0x02;
    inregs.h.bh = 0x00;
    inregs.h.dh = 24;
    inregs.h.dl = 0;

    int86(0x10, &inregs, &outregs);

    go_to_posicion_correcta();
    printf("Regresamos a la posicion inferior izquierda\n");
    printf("Texto antes de borrar la pantalla...\n");
    mi_pausa();

    clrscr2();
}

```

20. getX()

La función `getX()` obtiene la posición actual de la columna en la que se encuentra el cursor en la pantalla de texto en modo DOS. Para ello, utiliza la interrupción `10h` de BIOS con la función `03h`, que permite obtener la posición actual del cursor.

Antes de llamar a la interrupción, se configura `AH = 0x03` y `BH = 0x00` (para la primera página de vídeo). Tras la ejecución, el valor de la columna actual se almacena en `DL`. Como las coordenadas de la BIOS comienzan en 0, se retorna `DL + 1` para ajustar el valor al rango convencional (1-80), asegurando así que el resultado sea comprensible para el usuario.

```

int getX() {
    union REGS inregs, outregs;

    inregs.h.ah = 0x03;
    inregs.h.bh = 0x00;
    int86(0x10, &inregs, &outregs);

    return outregs.h.dl + 1;
}

printf("Introduce la columna (1-80): ");
scanf("%d", &x);
printf("Introduce la fila (1-25): ");
scanf("%d", &y);

gotoxy(x, y);
xx = getX();
yy = getY();
mi_pausa();
printf("El cursor esta en la posicion X: %d, Y: %d.", x, y);
mi_pausa();

```

21. getY()

La función `getY()` obtiene la posición actual del cursor en la pantalla de texto en modo DOS, específicamente la fila en la que se encuentra, utilizando la interrupción `10h` de BIOS con la función `03h`. Configura los registros para solicitar la posición del cursor en la primera página de video y luego llama a la interrupción. El valor de la fila se devuelve en el registro `DH` y, como las coordenadas en BIOS comienzan desde 0, la función ajusta el valor sumando 1 antes de retornarlo, convirtiendo así el valor a una base 1 (rango 1-25) para facilitar su interpretación.

```

int getY() {
    union REGS inregs, outregs;

    inregs.h.ah = 0x03;
    inregs.h.bh = 0x00;
    int86(0x10, &inregs, &outregs);

    return outregs.h.dh + 1;
}

printf("Introduce la columna (1-80): ");
scanf("%d", &x);
printf("Introduce la fila (1-25): ");
scanf("%d", &y);

gotoxy(x, y);
xx = getX();
yy = getY();
mi_pausa();
printf("El cursor esta en la posicion X: %d, Y: %d.", x, y);
mi_pausa();

```

22. clrscr()

La función `clrscr()` borra toda la pantalla de texto en modo DOS utilizando la interrupción 10h de BIOS con la función 06h, que desplaza la pantalla hacia arriba y elimina su contenido. Configura los registros para borrar toda el área visible (80x25 caracteres) y luego imprime el mensaje "Hemos borrado toda la pantalla" como confirmación.

```
void clrscr() {
    union REGS inregs, outregs;

    inregs.h.ah = 0x06;
    inregs.h.al = 0;
    inregs.h.bh = 0x07;
    inregs.x.cx = 0x0000;
    inregs.x.dx = 0x184F;

    int86(0x10, &inregs, &outregs);

    printf("Hemos borrado toda la pantalla\n");
}

printf("Texto antes de borrar la pantalla...\n");
mi_pausa();

clrscr();

mi_pausa();
```

23. clrscr2()

La función `clrscr2()` simula el borrado de la pantalla en la consola, pero de una manera más simple que la función `clrscr()`. En lugar de usar la interrupción 10h de BIOS, simplemente imprime 25 saltos de línea (`\n`), lo que desplaza el texto actual fuera de la vista, como si se hubiera borrado la pantalla. Luego, imprime el mensaje "Hemos borrado toda la pantalla" para confirmar la acción. Aunque no borra realmente el contenido de la pantalla, crea la ilusión de que la pantalla ha sido limpiada al mover todo el contenido fuera de la vista del usuario.

```
void clrscr2() {
    int i;
    for (i = 0; i < 25; i++) {
        printf("\n");
    }
    printf("Hemos borrado toda la pantalla\n");
}

dibujar_conejo(5, 5);
go_to_posicion_correcta();
mi_pausa();
clrscr2();
```

24. mostrarcolores() (Función auxiliar)

La función `mostrarColores()` imprime una lista de los colores disponibles (del 0 al 15) con sus nombres correspondientes, como negro, azul, verde, rojo y versiones más brillantes de estos colores. Esto permite al usuario conocer los valores numéricos disponibles al modificar colores en la consola.

```
void mostrarColores() {
    printf("Colores disponibles (0-15):\n");
    printf("0 - Negro\n");
    printf("1 - Azul\n");
    printf("2 - Verde\n");
    printf("3 - Aqua\n");
    printf("4 - Rojo\n");
    printf("5 - Magenta\n");
    printf("6 - Amarillo\n");
    printf("7 - Blanco\n");
    printf("8 - Gris\n");
    printf("9 - Azul claro\n");
    printf("10 - Verde claro\n");
    printf("11 - Aqua claro\n");
    printf("12 - Rojo claro\n");
    printf("13 - Magenta claro\n");
    printf("14 - Amarillo claro\n");
    printf("15 - Blanco brillante\n");
}

mostrarColores();
color_texto = pedirColorTexto();
mostrarColores();
color_fondo = pedirColorFondo();
cambiarColorTextoYFondo(color_texto, color_fondo);
printf("Color y fondo seleccionados\n");
mi_pausa();
```

25. pedircolorfondo()

La función pedirColorFondo() solicita al usuario que ingrese un valor para el color de fondo (entre 0 y 15). Si el valor ingresado es menor que 0, se ajusta a 0; si es mayor que 15, se ajusta a 15. Finalmente, la función retorna el valor del color de fondo válido ingresado o ajustado.

```
int pedirColorFondo() {
    int colorFondo;
    printf("Introduce el color del fondo (0-15): ");
    scanf("%d", &colorFondo);

    if (colorFondo < 0) colorFondo = 0;
    if (colorFondo > 15) colorFondo = 15;

    return colorFondo;
}

mostrarColores();
color_texto = pedirColorTexto();
mostrarColores();
color_fondo = pedirColorFondo();

rectangulo_lleno(x1, y1, x2, y2, color_texto, color_fondo);
```

26. pedircolortexto()

La función pedirColorTexto() solicita al usuario que ingrese un valor para el color del texto (entre 0 y 15). Si el valor ingresado es menor que 0, se ajusta a 0; si es mayor que 15, se ajusta a 15. Finalmente, la función retorna el valor del color de texto válido ingresado o ajustado.

```
int pedirColorTexto() {
    int colorTexto;
    printf("Introduce el color del texto (0-15): ");
    scanf("%d", &colorTexto);

    if (colorTexto < 0) colorTexto = 0;
    if (colorTexto > 15) colorTexto = 15;

    return colorTexto;
}

mostrarColores();
color_texto = pedirColorTexto();
mostrarColores();
color_fondo = pedirColorFondo();
cambiarColorTextoYFondo(color_texto, color_fondo);
printf("Color y fondo seleccionados\n");
mi_pausa();
```

27. modovideo() (Texto-Gráfico)

La función modovideo() cambia el modo de vídeo en el sistema utilizando la interrupción 10h de BIOS con la función ooh. El parámetro modo especifica el modo de vídeo que se desea activar. La función configura los registros AH y AL para indicar la operación de cambio de modo y el modo deseado, y luego llama a la interrupción int86(0x10, &inregs, &outregs) para realizar el cambio. Esto permite cambiar entre diferentes modos de vídeo, como el modo texto o modo gráfico.

```
// establece el modo de vídeo: 3-texto, 4-gráfico
void modovideo(BYTE modo){
    union REGS inregs, outregs;
    inregs.h.al = modo;
    inregs.h.ah = 0x00;
    int86(0x10, &inregs, &outregs);
}

modovideo(MODOGRAFICO);

pixel(10,40,0);
pixel(10,50,1);
pixel(15,60,2);
pixel(20,70,3);

for(i=0; i<100; i++){
    pixel(i,i, i%4 );
}

mi_pausa();
modovideo(MODOTEXTO);
```

28. pixel()

La función `pixel()` dibuja un píxel en la pantalla en las coordenadas (x, y) especificadas, utilizando un color determinado por el parámetro C. Para lograrlo, utiliza la interrupción 10h de BIOS con la función `oCh`, que permite poner un píxel en una posición específica de la pantalla en modo gráfico. Los registros se configuran de la siguiente manera: CX y DX contienen las coordenadas (x, y) del píxel, AL contiene el color (C), y AH se establece en `0x0C` para indicar la operación de dibujo del píxel. Después, la interrupción `int86(0x10, &inregs, &outregs)` se ejecuta para realizar la acción.

```
// pone un pixel en la coordenada X,Y de color C
void pixel(int x, int y, BYTE C){
    union REGS inregs, outregs;
    inregs.x.cx = x;
    inregs.x.dx = y;
    inregs.h.al = C;
    inregs.h.ah = 0x0C;
    int86(0x10, &inregs, &outregs);
}

pixel(10,40,0);
pixel(10,50,1);
pixel(15,60,2);
pixel(20,70,3);

for(i=0; i<100; i++){
    pixel(i,i, i%4 );
}
```

29. rectangulo_vacio()

La función `rectangulo_vacio()` dibuja un rectángulo vacío (es decir, su perímetro) en la pantalla utilizando los caracteres +, -, y |, con los colores de texto y fondo especificados. Los parámetros (x1, y1) y (x2, y2) definen las coordenadas de las esquinas superior izquierda e inferior derecha del rectángulo, mientras que `color1` y `color2` definen el color del texto y el color de fondo, respectivamente.

```
void rectangulo_vacio(int x1, int y1, int x2, int y2, int color1, int color2) {
    int colorTexto = color1;
    int colorFondo = color2;

    gotoxy(x1, y1);
    cputchar_color_fondo('+', colorTexto, colorFondo);
    for (x = x1 + 1; x < x2; x++) {
        gotoxy(x, y1);
        cputchar_color_fondo('-', colorTexto, colorFondo);
    }

    gotoxy(x2, y1);
    cputchar_color_fondo('+', colorTexto, colorFondo);

    for (y = y1 + 1; y < y2; y++) {
        gotoxy(x1, y);
        cputchar_color_fondo('|', colorTexto, colorFondo);
        gotoxy(x2, y);
        cputchar_color_fondo('|', colorTexto, colorFondo);
    }

    gotoxy(x1, y2);
    cputchar_color_fondo('+', colorTexto, colorFondo);
    for (x = x1 + 1; x < x2; x++) {
        gotoxy(x, y2);
        cputchar_color_fondo('-', colorTexto, colorFondo);
    }
    gotoxy(x2, y2);
    cputchar_color_fondo('+', colorTexto, colorFondo);

    printf("Introduce las coordenadas de la esquina superior izquierda (x1, y1): ");
    scanf("%d %d", &x1, &y1);

    printf("Introduce las coordenadas de la esquina inferior derecha (x2, y2): ");
    scanf("%d %d", &x2, &y2);

    if(x1>x2){
        auxiliar=x1;
        x1=x2;
        x2=auxiliar;
    }

    if(y1>y2){
        auxiliar=y1;
        y1=y2;
        y2=auxiliar;
    }

    mostrarColores();
    color_texto = pedirColorTexto();
    mostrarColores();
    color_fondo = pedirColorFondo();

    rectangulo_vacio(x1, y1, x2, y2, color_texto, color_fondo);

    mi_pausa();
    go_to_posicion_correcta();
}
```


30. rectangulo_lleno()

La función `rectangulo_lleno()` dibuja un rectángulo lleno en la pantalla, con bordes formados por los caracteres `+`, `-`, y `|`, y con el interior relleno de espacios. Los parámetros `(x1, y1)` y `(x2, y2)` definen las coordenadas de las esquinas superior izquierda e inferior derecha del rectángulo, mientras que `color1` y `color2` especifican el color del texto y del fondo, respectivamente.

```
void rectangulo_lleno(int x1, int y1, int x2, int y2, int color1, int color2) {
    int x, y;

    int colorTexto = color1;
    int colorFondo = color2;

    gotoxy(x1, y1);
    cputchar_color_fondo('+', colorTexto, colorFondo);
    for (x = x1 + 1; x < x2; x++) {
        gotoxy(x, y1);
        cputchar_color_fondo('-', colorTexto, colorFondo);
    }
    gotoxy(x2, y1);
    cputchar_color_fondo('+', colorTexto, colorFondo);

    for (y = y1 + 1; y < y2; y++) {
        gotoxy(x1, y);
        cputchar_color_fondo('|', colorTexto, colorFondo);
        for (x = x1 + 1; x < x2; x++) {
            gotoxy(x, y);
            cputchar_color_fondo(' ', colorTexto, colorFondo);
        }
        gotoxy(x2, y);
        cputchar_color_fondo('|', colorTexto, colorFondo);
    }

    gotoxy(x1, y2);
    cputchar_color_fondo('+', colorTexto, colorFondo);
    for (x = x1 + 1; x < x2; x++) {
        gotoxy(x, y2);
        cputchar_color_fondo('-', colorTexto, colorFondo);
    }
    gotoxy(x2, y2);
    cputchar_color_fondo('+', colorTexto, colorFondo);
}

printf("Introduce las coordenadas de la esquina superior izquierda (x1, y1): ");
scanf("%d %d", &x1, &y1);

printf("Introduce las coordenadas de la esquina inferior derecha (x2, y2): ");
scanf("%d %d", &x2, &y2);

if(x1>x2){
    auxiliar=x1;
    x1=x2;
    x2=auxiliar;
}

if(y1>y2){
    auxiliar=y1;
    y1=y2;
    y2=auxiliar;
}

mostrarColores();
color_texto = pedirColorTexto();
mostrarColores();
color_fondo = pedirColorFondo();

rectangulo_lleno(x1, y1, x2, y2, color_texto, color_fondo);

mi_pausa();

go_to_posicion_correcta();
mi_pausa();
```

31. dibujar_conejo()

La función `dibujar_conejo()` dibuja un dibujo en ascii 32 de un conejo en la consola, comenzando en la posición especificada por `(base_x, base_y)`. Utiliza `gotoxy()` para mover el cursor a las posiciones correspondientes y luego imprime cada línea del dibujo utilizando `printf()`.

```
void dibujar_conejo(int base_x, int base_y) {
    gotoxy(base_x, base_y);
    printf(" \\\ ");
    gotoxy(base_x, base_y + 1);
    printf(" .. ");
    gotoxy(base_x, base_y + 2);
    printf("c( ) ");
    gotoxy(base_x, base_y + 3);
    printf(" oo ");
}

dibujar_conejo(5, 5);
mi_pausa();

go_to_posicion_correcta();
mi_pausa();
```

32. dibujar_gato()

La función `dibujar_gato()` dibuja un gato en ascii 32 en la consola, comenzando en la posición especificada por `(x, y)`. Utiliza `gotoxy()` para mover el cursor a las posiciones correspondientes y luego imprime cada línea del dibujo utilizando `printf()`.

```
void dibujar_gato(int x, int y) {
    gotoxy(x, y);
    printf(" /\_/\ ");
    gotoxy(x, y + 1);
    printf("( o.o )");
    gotoxy(x, y + 2);
    printf("> ^ < ");
}

dibujar_gato(30, 2);
dibujar_pollo(10, 10);
dibujar_bob_esponja(40, 12);
dibujar_pez(5, 3);

go_to_posicion_correcta();
mi_pausa();
```

33. dibujar_pollo()

La función `dibujar_pollo()` dibuja un pollo en ascii 32 en la consola, comenzando en la posición especificada por (x, y). Utiliza `gotoxy()` para mover el cursor a las posiciones correspondientes y luego imprime cada línea del dibujo utilizando `printf()`.

```
void dibujar_pollo(int x, int y) {  
    gotoxy(x, y);  
    printf("  _");  
    gotoxy(x, y + 1);  
    printf("<(o )__");  
    gotoxy(x, y + 2);  
    printf(" ( _/ ");  
    gotoxy(x, y + 3);  
    printf(" ^^ ");  
}  
  
dibujar_gato(30, 2);  
dibujar_pollo(10, 10);  
dibujar_bob_esponja(40, 12);  
dibujar_pez(5, 3);  
  
go_to_posicion_correcta();  
mi_pausa();
```

34. dibujar_bob_esponja()

La función `dibujar_bob_esponja()` dibuja una figura que representa a Bob Esponja en ascii 32 en la consola, comenzando en la posición especificada por (x, y). Utiliza `gotoxy()` para mover el cursor a las posiciones correspondientes y luego imprime cada línea del dibujo utilizando `printf()`.

```
void dibujar_bob_esponja(int x, int y) {  
    gotoxy(x, y);  
    printf(" +-----+ ");  
    gotoxy(x, y + 1);  
    printf(" | o o | ");  
    gotoxy(x, y + 2);  
    printf(" | o | ");  
    gotoxy(x, y + 3);  
    printf(" | '-' | ");  
    gotoxy(x, y + 4);  
    printf(" +-----+ ");  
}  
  
dibujar_gato(30, 2);  
dibujar_pollo(10, 10);  
dibujar_bob_esponja(40, 12);  
dibujar_pez(5, 3);  
  
go_to_posicion_correcta();  
mi_pausa();
```

35. dibujar_pez()

La función `dibujar_pez()` dibuja un pez simple en ascii 32 en la consola, comenzando en la posición especificada por (x, y). Utiliza `gotoxy()` para mover el cursor a las posiciones correspondientes y luego imprime el dibujo del pez utilizando `printf()`. El dibujo consiste en una sola línea que representa un pez, con la forma `><(((o>))`, donde los símbolos forman la cabeza, cuerpo y cola del pez.

```
void dibujar_pez(int x, int y) {  
    gotoxy(x, y);  
    printf("><(((o>))");  
}  
  
dibujar_gato(30, 2);  
dibujar_pollo(10, 10);  
dibujar_bob_esponja(40, 12);  
dibujar_pez(5, 3);  
  
go_to_posicion_correcta();  
mi_pausa();
```


36. dibujar_rombo()

La función `dibujar_rombo()` dibuja un rombo en la consola en modo gráfico utilizando la función `pixel()`, que establece un píxel en las coordenadas especificadas con un color determinado. Comienza en la posición (`base_x`, `base_y`) y luego dibuja el rombo al colocar píxeles en varias posiciones relativas a esa coordenada central.

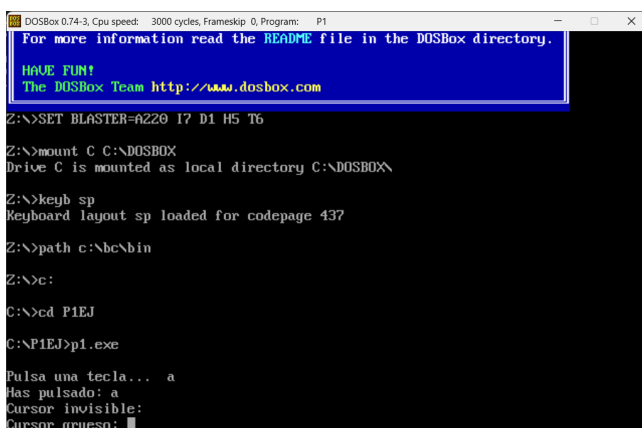
```
void dibujar_rombo(int cx, int cy, int tamano) {  
    int i, j;  
    for (i = -tamano; i <= tamano; i++) {  
        for (j = -tamano; j <= tamano; j++) {  
            if (abs(i) + abs(j) <= tamano) {  
                pixel(cx + i, cy + j, 1);  
            }  
        }  
    }  
}  
  
modovideo(MODOGRAFICO);  
  
dibujar_rombo(100, 60, 50);  
mi_pausa();
```

37. dibujar_cuadrado()

La función `dibujar_cuadrado()` dibuja un cuadrado en la consola en modo gráfico utilizando la función `pixel()`, que coloca un píxel en las coordenadas especificadas con un color determinado (en este caso, el color 2), dibujando un cuadrado de dimensión tamaño x tamaño (tamano x tamano al no aceptar la ñ).

```
void dibujar_cuadrado(int base_x, int base_y, int tamano) {  
    int i, j;  
    for (i = base_x; i < base_x + tamano; i++) {  
        for (j = base_y; j < base_y + tamano; j++) {  
            pixel(i, j, 2);  
        }  
    }  
}  
  
modovideo(MODOGRAFICO);  
  
dibujar_cuadrado(100, 50, 100);  
mi_pausa();  
  
modovideo(MODOTEXT);
```

38. ejecución del programa



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: P1  
For more information read the README file in the DOSBox directory.  
HAVE FUN!  
The DOSBox Team http://www.dosbox.com  
Z:\>SET BLASTER=A220 I7 D1 H5 T6  
Z:\>mount C C:\DOSBOX  
Drive C is mounted as local directory C:\DOSBOX\  
Z:\>keyb sp  
Keyboard layout sp loaded for codepage 437  
Z:\>path c:\bc\bin  
Z:\>c:  
C:\>cd P1EJ  
C:\P1EJ>p1.exe  
Pulsa una tecla... a  
Has pulsado: a  
Cursor invisible:  
Cursor grueso: █
```



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip: 0, Program: P1
Modo de video: 80x25

En C. Pulsa una tecla...
Color de TEXTO actual: 7
Color de FONDO actual: 0
Texto en color rojo
Color de TEXTO actual: 4
Color de FONDO actual: 0
Fondo en color magenta
Color de TEXTO actual: 0
Color de FONDO actual: 5
Texto en color azul claro con fondo blanco
Color de TEXTO actual: 9
Color de FONDO actual: 7
```

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip: 0, Program: P1
Fondo en color magenta
Color de TEXTO actual: 0
Color de FONDO actual: 5
Texto en color azul claro con fondo blanco
Color de TEXTO actual: 9
Color de FONDO actual: 7
Colores restaurados a los valores originales.
Colores disponibles (0-15):
0 - Negro
1 - Azul
2 - Verde
3 - Aqua
4 - Rojo
5 - Magenta
6 - Amarillo
7 - Blanco
8 - Gris
9 - Azul claro
10 - Verde claro
11 - Aqua claro
12 - Rojo claro
13 - Magenta claro
14 - Amarillo claro
15 - Blanco brillante
Introduce el color del texto (0-15):
```

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip: 0, Program: P1
12 - Rojo claro
13 - Magenta claro
14 - Amarillo claro
15 - Blanco brillante
Introduce el color del texto (0-15): 6
Colores disponibles (0-15):
0 - Negro
1 - Azul
2 - Verde
3 - Aqua
4 - Rojo
5 - Magenta
6 - Amarillo
7 - Blanco
8 - Gris
9 - Azul claro
10 - Verde claro
11 - Aqua claro
12 - Rojo claro
13 - Magenta claro
14 - Amarillo claro
15 - Blanco brillante
Introduce el color del fondo (0-15): 7
Color y fondo seleccionados
```

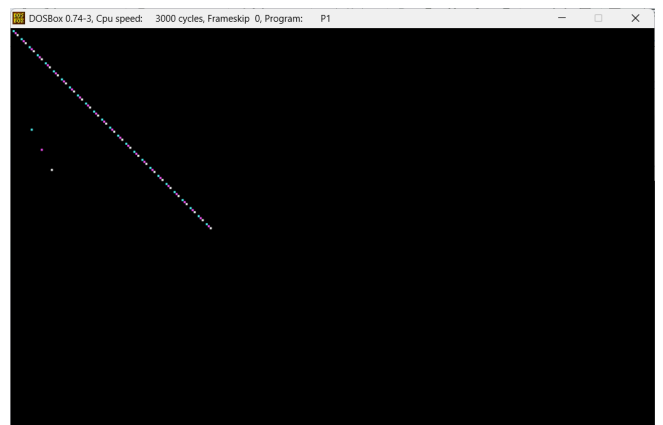
```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip: 0, Program: P1
Color y fondo seleccionados
Colores restaurados a los valores originales.
Caracter en rojo: R
Caracter con fondo azul: F
Caracter amarillo sobre fondo magenta: Z
Pulsa una tecla... A
Colores disponibles (0-15):
0 - Negro
1 - Azul
2 - Verde
3 - Aqua
4 - Rojo
5 - Magenta
6 - Amarillo
7 - Blanco
8 - Gris
9 - Azul claro
10 - Verde claro
11 - Aqua claro
12 - Rojo claro
13 - Magenta claro
14 - Amarillo claro
15 - Blanco brillante
Introduce el color del texto (0-15): 6_
```

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip: 0, Program: P1
12 - Rojo claro
13 - Magenta claro
14 - Amarillo claro
15 - Blanco brillante
Introduce el color del texto (0-15): 6
Colores disponibles (0-15):
0 - Negro
1 - Azul
2 - Verde
3 - Aqua
4 - Rojo
5 - Magenta
6 - Amarillo
7 - Blanco
8 - Gris
9 - Azul claro
10 - Verde claro
11 - Aqua claro
12 - Rojo claro
13 - Magenta claro
14 - Amarillo claro
15 - Blanco brillante
Introduce el color del fondo (0-15): 7
A
```

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip: 0, Program: P1
Hemos borrado toda la pantalla
```

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip: 0, Program: P1
El cursor esta en la posicion X: 16, Y: 17.

Hemos borrado toda la pantalla
Introduce la columna (1-80): 16
Introduce la fila (1-25): 17
```



```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: P1
Introduce las coordenadas de la esquina superior izquierda (x1, y1): 5
6
Introduce las coordenadas de la esquina inferior derecha (x2, y2): 15
16
Colores disponibles (0-15):
0 - Negro
1 - Azul
2 - Verde
3 - Agua
4 - Rojo
5 - Magenta
6 - Amarillo
7 - Blanco
8 - Gris
9 - Azul claro
10 - Verde claro
11 - Agua claro
12 - Rojo claro
13 - Magenta claro
14 - Amarillo claro
15 - Blanco brillante
Introduce el color del texto (0-15):
```

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: P1
11 - Agua claro
12 - Rojo claro
13 - Magenta claro
14 - Amarillo claro
15 - Blanco brillante
Introduce el color del texto (0-15): 6
Colores disponibles (0-15):
0 - Negro
1 - Azul
2 - Verde
3 - Agua
4 - Rojo
5 - Magenta
6 - Amarillo
7 - Blanco
8 - Gris
9 - Azul claro
10 - Verde claro
11 - Agua claro
12 - Rojo claro
13 - Magenta claro
14 - Amarillo claro
15 - Blanco brillante
Introduce el color del fondo (0-15): 9
```

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: P1
Hemos borrado toda la pantalla
Introduce las coordenadas de la esquina superior izquierda (x1, y1): 15
16
Introduce las coordenadas de la esquina inferior derecha (x2, y2): 5
6
Colores disponibles (0-15):
0 - Negro
1 - Azul
2 - Verde
3 - Agua
4 - Rojo
5 - Magenta
6 - Amarillo
7 - Blanco
8 - Gris
9 - Azul claro
10 - Verde claro
11 - Agua claro
12 - Rojo claro
13 - Magenta claro
14 - Amarillo claro
15 - Blanco brillante
Introduce el color del texto (0-15):
```

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: P1
11 - Agua claro
12 - Rojo claro
13 - Magenta claro
14 - Amarillo claro
15 - Blanco brillante
Introduce el color del texto (0-15): 6
Colores disponibles (0-15):
0 - Negro
1 - Azul
2 - Verde
3 - Agua
4 - Rojo
5 - Magenta
6 - Amarillo
7 - Blanco
8 - Gris
9 - Azul claro
10 - Verde claro
11 - Agua claro
12 - Rojo claro
13 - Magenta claro
14 - Amarillo claro
15 - Blanco brillante
Introduce el color del fondo (0-15): 9
```

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: P1
( \ )
( . )
c ( )
oo

Hemos borrado toda la pantalla
```

```
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Program: P1
><(((||>
( o.o )
> ^ <

<(o )
^ ^

+-----+
| 0 0 |
|  o  |
| ' ' |
+-----+

Hemos borrado toda la pantalla
```

