Librerías Borland ®

Funciones y estructuras

Steven R. Davidson steven@conclase.net y

Con Clase: http://www.conclase.net

Librerías de Borland® (para C y C++)

Hemos observado que mucha gente, especialmente en la lista, usa los compiladores de Borland®. Aparte de usar este compilador, también existe una demanda para saber acerca de las librerías no estándar ofrecidas por Borland®.

Tenemos que advertir que estas librerías no son estándar, por lo que los programas que hagan uso de ellas no serán necesariamente portables a otras plataformas ni a otros compiladores.

Sin embargo, estas páginas podrán usarse como consulta para ver el funcionamiento de cada función individual. Para que la consulta sea más fácil, se incluye un índice alfabético de funciones, y un índice de ficheros de cabecera.

Como se ha hecho hasta ahora en el resto de los cursos, se irán añadiendo y actualizando páginas a medida que estén preparadas, de modo que no sea necesario esperar a que todo esté terminado para empezar las consultas.

Nota: algunas descripciones de funciones, estructuras y macros han sido extraídas de la ayuda de los compiladores de Borland y del libro: "Borland® C++ Programmer's Guide to Graphics" de James W. McCord.

Librería conio Borland ® C

Contiene los prototipos de las funciones, macros, y constantes para preparar y manipular la consola en modo texto en el entorno de MS-DOS®.

Funciones

cgets	clreol	clrscr	cprintf
cputs	cscanf	delline	getch
getche	getpass	gettext	gettextinfo
gotoxy	highvideo	inport	insline
kbhit	lowvideo	movetext	normvideo
outport	putch	puttext	setcursortype
textattr	textbackground	textcolor	ungetch
wherex	wherey	window	

Macros

colores	inp	inportb	inpw
modos	outp	outportb	outpw

Estructuras

text_info

Librería graphics Borland® C

Contiene los prototipos de las funciones para preparar y manipular la parte gráfica en el entorno de MS-DOS®.

Funciones

arc	bar	bar3d	circle
cleardevice	clearviewport	closegraph	detectgraph
drawpoly	ellipse	fillellipse	fillpoly
floodfill	getarccoords	getaspectratio	getbkcolor
getcolor	getdefaultpalette	getdrivername	getfillpattern
getfillsettings	getgraphmode	getimage	getlinesettings
getmaxcolor	getmaxmode	getmaxx	getmaxy
getmodename	getmoderange	getpalette	getpalettesize
getpixel	gettextsettings	getviewsettings	getx
gety	graphdefaults	grapherrormsg	graphfreemem
graphgetmem	graphresult	imagesize	initgraph
installuserdriver	installuserfont	line	linerel
lineto	moverel	moveto	outtext
outtextxy	pieslice	putimage	putpixel
rectangle	registerbgidriver	registerbgifont	restorecrtmode
sector	setactivepage	setallpalette	setaspectratio
setbkcolor	setfillpattern	setfillstyle	setgraphbufsize
setgraphmode	setlinestyle	setpalette	setrgbpalette
settextjustify	settextstyle	setusercharsize	setviewport
setvisualpage	setwritemode	textheight	textwidth

Macros

colores	drivers	enlazar	errores
fuentes	linea	modos	put_op
trama			

Estructuras

arccoordstype fillsettingstype linesettingstype palettetype textsettingstype viewporttype

Indice de funciones

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

-A-

Función	Librería	Fichero de cabecera C
arc	graphics	graphics.h

-B-

Función	Librería	Fichero de cabecera C
bar	graphics	graphics.h
bar3d	graphics	graphics.h

-C-

Función	Librería	Fichero de cabecera C
cgets	conio	conio.h
circle	graphics	graphics.h
cleardevice	graphics	graphics.h
clearviewport	graphics	graphics.h
closegraph	graphics	graphics.h
clreol	conio	conio.h
clrscr	conio	conio.h
cprintf	conio	conio.h
cputs	conio	conio.h
cscanf	conio	conio.h

-D-

Función	Librería	Fichero de cabecera C
delline	conio	conio.h
detectgraph	graphics	graphics.h
drawpoly	graphics	graphics.h

-E-

Función	Librería	Fichero de cabecera C
ellipse	graphics	graphics.h

-F-

Función	Librería	Fichero de cabecera C
fillellipse	graphics	graphics.h
fillpoly	graphics	graphics.h
floodfill	graphics	graphics.h

-G-

Función	Librería	Fichero de cabecera C
getarccoords	graphics	graphics.h
getaspectratio	graphics	graphics.h
getbkcolor	graphics	graphics.h
getch	conio	conio.h
getche	conio	conio.h
getcolor	graphics	graphics.h
getdefaultpalette	graphics	graphics.h
getdrivername	graphics	graphics.h
getfillpattern	graphics	graphics.h
getfillsettings	graphics	graphics.h
getgraphmode	graphics	graphics.h
getimage	graphics	graphics.h
getlinesettings	graphics	graphics.h
getmaxcolor	graphics	graphics.h
getmaxmode	graphics	graphics.h
getmaxx	graphics	graphics.h
getmaxy	graphics	graphics.h
getmodename	graphics	graphics.h
getmoderange	graphics	graphics.h
getpalette	graphics	graphics.h
getpalettesize	graphics	graphics.h
getpass	conio	conio.h

getpixel	graphics	graphics.h
gettext	conio	conio.h
gettextinfo	conio	conio.h
gettextsettings	graphics	graphics.h
getviewsettings	graphics	graphics.h
getx	graphics	graphics.h
gety	graphics	graphics.h
gotoxy	conio	conio.h
graphdefaults	graphics	graphics.h
grapherrormsg	graphics	graphics.h
graphfreemem	graphics	graphics.h
graphgetmem	graphics	graphics.h
graphresult	graphics	graphics.h

-H-

Función	Librería	Fichero de cabecera C
highvideo	conio	conio.h

-I-

Función	Librería	Fichero de cabecera C
imagesize	graphics	graphics.h
initgraph	graphics	graphics.h
inport	conio	conio.h
insline	conio	conio.h
installuserdriver	graphics	graphics.h
installuserfont	graphics	graphics.h

-K-

Función	Librería	Fichero de cabecera C
kbhit	conio	conio.h

-L-

Función	Librería	Fichero de cabecera C

line	graphics	graphics.h
linerel	graphics	graphics.h
lineto	graphics	graphics.h
lowvideo	conio	conio.h

-M-

Función	Librería	Fichero de cabecera C
moverel	graphics	graphics.h
movetext	conio	conio.h
moveto	graphics	graphics.h

-N-

Función	Librería	Fichero de cabecera C
normvideo	conio	conio.h

-O-

Función	Librería	Fichero de cabecera C
outport	conio	conio.h
outtext	graphics	graphics.h
outtextxy	graphics	graphics.h

-P-

Función	Librería	Fichero de cabecera C
pieslice	graphics	graphics.h
putch	conio	conio.h
putimage	graphics	graphics.h
putpixel	graphics	graphics.h
puttext	conio	conio.h

-R-

Función	Librería	Fichero de cabecera C
rectangle	graphics	graphics.h
registerbgidriver	graphics	graphics.h

registerbgifont	graphics	graphics.h
restorecrtmode	graphics	graphics.h

- S -

Función	Librería	Fichero de cabecera C
sector	graphics	graphics.h
setactivepage	graphics	graphics.h
setallpalette	graphics	graphics.h
setaspectratio	graphics	graphics.h
setbkcolor	graphics	graphics.h
setcursortype	conio	conio.h
setfillpattern	graphics	graphics.h
setfillstyle	graphics	graphics.h
setgraphbufsize	graphics	graphics.h
setgraphmode	graphics	graphics.h
setlinestyle	graphics	graphics.h
setpalette	graphics	graphics.h
setrgbpalette	graphics	graphics.h
settextjustify	graphics	graphics.h
settextstyle	graphics	graphics.h
setusercharsize	graphics	graphics.h
setviewport	graphics	graphics.h
setvisualpage	graphics	graphics.h
setwritemode	graphics	graphics.h

- T -

Función	Librería	Fichero de cabecera C
textattr	conio	conio.h
textbackground	conio	conio.h
textcolor	conio	conio.h
textheight	graphics	graphics.h
textwidth	graphics	graphics.h

- U -

Función	Librería	Fichero de cabecera C
ungetch	conio	conio.h

- W -

Función	Librería	Fichero de cabecera C
wherex	conio	conio.h
wherey	conio	conio.h
window	conio	conio.h

Función arc Borland® C

Librería: graphics

```
void far arc(int x, int y,
   int comienzo_angulo, int final_angulo, int radio);
```

Esta función creará un arco circular. El arco tiene como centro el punto especificado por los argumentos **x** e **y**, y es dibujado con el radio especificado: **radio**. El arco no está rellanado, pero es dibujado usando el color actual. El arco comienza al ángulo especificado por el argumento **comienzo_angulo** y es dibujado en la dirección contraria al de las agujas del reloj hasta llegar al ángulo especificado por el argumento **final_angulo**. La función *arc* usa el este (extendiéndose hacia la derecha del centro del arco en la dirección horizontal) como su punto de 0 grados. La función setlinestyle puede usarse para establecer el grosor del arco. La función *arc*, sin embargo, ignorará el argumento **trama** de la función setlinestyle.

Valor de retorno:

La función *arc* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int radio;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/
/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   for( radio = 25; radio < 175; radio += 25 )
      arc( 320, 175, 45, 135, radio );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función bar Borland® C

Librería: graphics

```
void far bar(int izquierda, int superior,
   int derecha, int inferior);
```

Esta función dibujará una barra rectangular y rellenada de dos dimensiones. La esquina superior izquierda de la barra rectangular está definida por los argumentos **izquierda** y **superior**. Estos argumentos corresponden a los valores *x* e *y* de la esquina superior izquierda. Similarmente, los argumentos **derecha** e **inferior** definen la esquina inferior derecha de la barra. La barra no tiene borde, pero es rellenada con la trama de relleno actual y el color de relleno como es establecido por la función setlinestyle.

Valor de retorno:

La función *bar* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x, y, color, relleno;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   x = 20;
   y = 20;
   color = 1;
   fill = 1;

   do {
      setfillstyle( fill, color );
      bar( x, y, x+40, 320 );
      x += 40;
      y += 10;
      color = (color+1) % 16;
      fill = (fill+1) % 12;
   } while( x < 620 );

   getch();    /* Pausa */
```

```
    closegraph();

    return 0;
}
```

Función bar3d Borland® C

Librería: graphics

```
void far bar3d(int izquierda, int superior,
   int derecha, int inferior, int profundidad, int banderin_tapa);
```

Esta función creará una barra rectangular y rellenada de tres dimensiones. La esquina superior izquierda de la barra rectangular más frontal está definida por los argumentos **izquierda** y **superior**. Estos argumentos corresponden a los valores *x* e *y* de la esquina superior izquierda del rectángulo más frontal. Similarmente, los argumentos **derecha** e **inferior** definen la esquina inferior derecha del rectángulo más frontal. La barra tiene borde, en todas las tres dimensiones, rellenada con el color y estilo de línea actuales. El rectángulo más frontal es rellenado usando la trama de relleno actual y el color de relleno como es establecido por la función setlinestyle. El argumento **banderin_tapa** es usado para especificar si es o no es posible apilar varias barras encima de cada una. Si **banderin_tapa** tiene un valor distinto a cero, entonces la barra está "tapada". Si **banderin_tapa** tiene un valor de cero, entonces la barra no está "tapada", permitiendo otras barras ser apiladas encima de ésta.

Valor de retorno:

La función *bar3d* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int color, relleno;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   color = 10;
   relleno = 11;

   setfillstyle( relleno, color );
   bar3d( 100, 50, 300, 150, 25, 1 );

   getch();    /* Pausa */
   closegraph();

   return 0;
```

```
}
```

Función cgets Borland® C

Librería: conio

```
char *cgets(char *cadena);
```

Esta función leerá una cadena de caracteres desde la consola, guardando la cadena (y su longitud) en el lugar apuntado por ***cadena**. La función *cgets* leerá caracteres hasta que encuentre una combinación de retorno de línea y nueva línea (CR/LF), o hasta que el número máximo de caracteres permitidos hayan sido leídos. Si se lee una combinación CR/LF, entonces es sustituido por un carácter nulo '\0' antes de ser guardado la cadena.

Antes de que la función *cgets* es llamada, asigna a **cadena[0]** la longitud máxima de la cadena a ser leída. Al retornar, **cadena[1]** es asignado el número de caracteres leídos. Los caracteres leídos comienzan a partir de **cadena[2]** (incluido) y termina con el carácter nulo. Por esta razón, ***cadena** debe ser como mínimo **cadena[0]** más 2 bytes de longitud.

Valor de retorno:

La función *cgets* retorna la cadena de caracteres a partir de **cadena[2]**, si tiene éxito.

Ejemplo:

```
#include <conio.h>

int main() {
   char cadena[23];
   char *cad;

   cadena[0] = 21;    /* 20 caracteres del usuario y el carácter nulo final */
   cprintf( "Escriba un mensaje: " );
   cad = cgets( cadena );

   cprintf( "\r\nEscribiste: \"%s\"\r\nEl mensaje tiene %d caracteres.\r\n\r\n",
      cad, cadena[1] );

   return 0;
}
```

Función circle Borland® C

Librería: graphics

```
void far circle(int x, int y, int radio);
```

Esta función se usas para dibujar un círculo. Los argumentos **x** e **y** definen el centro del círculo, mientras que el argumento **radio** define el radio del círculo. El círculo no es rellenado pero es dibujado usando el color actual. El grosor de la circunferencia puede ser establecido por la función setlinestyle; sin embargo, el estilo de la línea es ignorado por la función *circle*. La proporción anchura-altura para el modo actual es considerado cuando se calcula el círculo. Por esta razón, alterando los valores de defecto x e y de los factores de anchura-altura afectará el círculo (ya no seguirá siendo redondo).

Valor de retorno:

La función *circle* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int relleno;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   relleno = 1;

   setlinestyle( SOLID_LINE, relleno, THICK_WIDTH );
   circle( 300, 200, 80 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función cleardevice Borland® C

Librería: graphics

```
void far cleardevice(void);
```

Esta función es usada para despejar una pantalla gráfica. La función *cleardevice* usa el color de fondo actual, como es establecido por la función setbkcolor, para rellenar la pantalla. La posición del cursor gráfico es la esquina superior izquierda de la pantalla - posición (0,0) - después de que la pantalla haya sido borrado.

Valor de retorno:

La función *cleardevice* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int relleno, color;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   relleno = 1;
   color = 1;

   setlinestyle( SOLID_LINE, relleno, THICK_WIDTH );
   circle( 300, 200, 80 );
   getch();    /* Pausa */

   setbkcolor( color );
   cleardevice();
   setlinestyle( SOLID_LINE, relleno, THICK_WIDTH );
   circle( 400, 200, 20 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función clearviewport Borland® C

Librería: graphics

```
void far clearviewport(void);
```

Esta función es usada para rellenar la pantalla actual del usuario con el color de fondo actual. El color de fondo puede ser establecido con la función setbkcolor. La posición del cursor gráfico es la esquina superior izquierda de la pantalla actual del usuario. Esta posición es (0,0) según la pantalla actual del usuario.

Valor de retorno:

La función *clearviewport* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int color;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setviewport( 150, 150, 350, 350, 0 );
   for( color = 0; color<16; color++ ) {
      circle( 100, 100, 60 );
      getch();
      setbkcolor( color );
      clearviewport();
   }

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función closegraph Borland® C

Librería: graphics

```
void far closegraph(void);
```

Esta función es usada para cerrar el sistema gráfico como es iniciada por la función initgraph. La función *closegraph* libera toda la memoria usada por el sistema gráfico y luego restaura el modo de vídeo al modo de texto que estaba en uso anteriormente a la llamada a la función initgraph.

Valor de retorno:

La función *closegraph* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   circle( 300, 200, 80 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función clreol Borland® C

Librería: conio

```
void clreol(void);
```

Esta función despeja todos los caracteres desde la posición del cursor hasta el final de la línea dentro de la ventana de texto actual, sin mover la posición del cursor.

Valor de retorno:

La función *clreol* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>
#include <stdio.h>

int main() {
   clrscr();
   printf( "Ejemplo de \"clreol\"\n\n" );
   printf( "Esto es un mensaje\nescrito en varias líneas.\n" );
   printf( "Usaremos \"gotoxy\"para colocar el cursor\n
   printf( "en una de estas líneas, para borrarla.\n" );
   printf( "Pulsa una tecla para continuar...\n" );
   getch();
   gotoxy( 1, 4 );
   clreol();
   getch();

   return 0;
}
```

Función clrscr Borland® C

Librería: conio

```
void clrscr(void);
```

Esta función despeja la ventana de texto actual y coloca el cursor en la esquina superior izquierda: posición (1,1).

Valor de retorno:

La función *clrscr* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>
#include <stdio.h>

int main() {
   printf( "Ejemplo de \"clrscr\"\n\n" );
   printf( "Pulsa una tecla para continuar...\n" );
   getch();
   clrscr();

   return 0;
}
```

Función cprintf Borland® C

Librería: conio

```
int cprintf(const char *formato, ...);
```

Muestra texto en pantalla según el formato descrito. Esta función es similar a la función printf, pero con la excepción de que la función *cprintf* no convertirá los caracteres de nueva línea (\n) en la pareja de caracteres de retorno de línea/nueva línea (\r\n). Los caracteres de tabulación (\t) no serán expandidos a espacios. La cadena de texto con formato será enviado directamente a la ventana de texto actual en la pantalla. Esto se realiza mediante una escritura directa a la memoria de la pantalla o mediante una llamada a la BIOS, dependiendo del valor de la variable global **_directvideo**.

Valor de retorno:

La función *cprintf* retorna el número de caracteres mostrados en pantalla.

Ejemplo:

```
#include <conio.h>

int main() {
   cprintf( "Ejemplo de \"cprintf\"\r\n\r\n" );
   cprintf( "Comenzamos en esta línea, pero con \'\\n\'\n" );
   cprintf( "Nos saltamos a la siguiente línea sin volver al comienzo de línea\r\n"
);
   cprintf( "Pulsa una tecla para continuar..." );
   getch();

   return 0;
}
```

Función cputs Borland® C

Librería: conio

```
int cputs(const char *cadena);
```

Muestra la cadena, que finaliza con el carácter nulo, apuntada por el argumento ***cadena** en la ventana de texto actual. Esta función es similar a la función puts, pero con dos excepciones: la función *cputs* no convertirá los caracteres de nueva línea (\n) en la pareja de caracteres de retorno de línea/nueva línea (\r\n) tampoco añadirá el carácter de nueva línea al final del texto. Esto se realiza mediante una escritura directa a la memoria de la pantalla o mediante una llamada a la BIOS, dependiendo del valor de la variable global **_directvideo**.

Valor de retorno:

La función *cputs* retorna el último carácter mostrad en pantalla.

Ejemplo:

```
#include <conio.h>

int main() {
   cputs( "Ejemplo de \"cputs\"\r\n\r\n" );
   cputs( "Comenzamos en esta línea, pero sin \'\\n\'" );
   cprintf( "Seguimos en esta línea\r\n" );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();

   return 0;
}
```

Función cscanf Borland® C

Librería: conio

```
int cscanf(const char *formato, ...);
```

Recoge el texto y lo procesa según el formato dado por el argumento ***formato**. Esta función es similar a la función scanf., la diferencia está en que la función *cscanf* lee los datos desde la consola que son automáticamente mostrados.

Valor de retorno:

La función *cscanf* retorna el número de elementos entrados que hayan sido escaneados, convertidos, y guardados con éxito; el valor retornado no incluye elementos que no hayan sido guardados. Si no se han guardado elementos leídos, el valor de retorno es 0. Si *cscanf* intenta leer al final-de-fichero, el valor retornado es EOF.

Ejemplo:

```
#include <conio.h>

int main() {
   char nombre[25];
   int total;

   cprintf( "Escribe tu nombre:\r\n" );
   /* Intenta borrar unos caracteres escritos */
   total = cscanf( "%s", nombre );
   cprintf( "Tu nombre es \"%s\"\r\n", nombre );
   cprintf( "Número total de elementos guardados: %d\r\n", total );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();

   return 0;
}
```

Función delline Borland® C

Librería: conio

```
void delline(void);
```

Borra la línea donde se encuentre el cursor y mueve todas las líneas inferiores a una línea anterior. La función *delline* funciona en la ventana de texto activa.

Valor de retorno:

La función *delline* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>

int main() {
   clrscr();
   cprintf( "Ejemplo de \"delline\"\r\n\r\n" );
   cprintf( "Esta línea será borrada\r\n" );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();
   gotoxy( 1, 3 );
   delline();

   return 0;
}
```

Función detectgraph Borland® C

Librería: graphics

```
void far detectgraph(int far *driver, int far *modo);
```

Esta función es usada para detectar el adaptador gráfico y el modo óptimo para usar con el sistema en uso. Si la función *detectgraph* no puede detectar ningún dispositivo gráfico, el argumento ***driver** es asignado grNotDetected (-2). Una llamada a graphresult resultará en un valor de retorno de -2, o grNotDetected.

Existen varios valores que indican los diferentes dispositivos gráficos que pueden ser usados por el argumento ***driver**. Un valor de 0, o DETECT, inicia la funcionalidad de autodetección, el cual determina el driver óptimo a usar.

Para cada dispositivo existen varios valores que indican los diferentes modos gráficos que pueden ser usados por el argumento ***modo**. Sin embargo, si el argumento ***driver** es asignado el valor de 0, o DETECT, el argumento ***modo** es automáticamente establecido al modo de resolución mas alto para el driver.

Valor de retorno:

La función *detectgraph* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver, gmodo;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   detectgraph( &gdriver, &gmodo, );
   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   circle( 300, 200, 80 );

   getch();    /* Pausa */
   closegraph();

   printf( "Driver: %d\tModo: %d\n\n", gdriver, gmodo );

   return 0;
}
```

Función drawpoly Borland® C

Librería: graphics

```
void far drawpoly(int numpuntos, int far *puntos);
```

Esta función es usada para crear un polígono con un número especificado de puntos. El argumento **numpuntos** es usado para definir el número de puntos en el polígono. Para la función *drawpoly*, el número de puntos debe ser el número actual de puntos más 1 para poder crear un polígono cerrado. En otras palabras, el primer punto debe ser igual al último punto. El argumento ***puntos** apunta a un array de números de longitud **numpuntos** multiplicado por 2. Los dos primeros miembros del array identifica las coordenadas *x* e *y* del primer punto, respectivamente, mientras que los dos siguientes especifican el siguiente punto, y así sucesivamente. La función *drawpoly* dibuja el perímetro del polígono con el estilo de línea y color actuales, pero no rellena el polígono.

Valor de retorno:

La función *drawpoly* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int puntos[8] = { 300, 50, 500, 300, 100, 300, 300, 50 };

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   drawpoly( 4, puntos );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función ellipse Borland® C

Librería: graphics

```
void far ellipse(int x, int y, int comienzo_angulo,
   int final_angulo, int x_radio, int y_radio);
```

Esta función es usada para dibujar un arco elíptico en el color actual. El arco elíptico está centrado en el punto especificado por los argumentos **x** e **y**. Ya que el arco es eilptíco el argumento **x_radio** especifica el radio horizontal y el argumento **y_radio** especifica el radio vertical. El arco elíptico comienza con el ángulo especificado por el argumento **comienzo_angulo** y se extiende en un sentido contrario a las agujas del reloj al ángulo especificado por el argumento **final_angulo**. La función *ellipse* considera este - el eje horizontal a la derecha del centro del elipse - ser 0 grados. El arco elíptico es dibujado con el grosor de línea actual como es establecido por la función setlinestyle. Sin embargo, el estilo de línea es ignorado por la función *ellipse*.

Valor de retorno:

La función *ellipse* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   ellipse( 300, 150, 45, 225, 100, 50 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función fillellipse Borland® C

Librería: graphics

```
void far fillellipse(int x, int y,
   int x_radio, int y_radio);
```

Esta función es usada para dibujar y rellenar una elipse. El centro de la elipse es especificado por los argumentos **x** e **y**. El argumento **x_radio** especifica el radio horizontal y el argumento **y_radio** especifica el radio vertical de la elipse. La elipse es dibujado con el perímetro en el color actual y rellenada con el color de relleno y la trama de relleno actuales.

Valor de retorno:

La función *fillellipse* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int trama, color;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   trama = SOLID_FILL;
   color = 4;
   setfillstyle( trama, color );

   fillellipse( 300, 150, 100, 50 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función fillpoly Borland® C

Librería: graphics

```
void far fillpoly(int numpuntos, int far *puntos);
```

Esta función es usada para crear un polígono rellenado. El argumento **numpuntos** es usado para definir el número de puntos en el polígono. Al contrario que la función drawpoly, la función automáticamente cierra el polígono. El argumento ***puntos** apunta a un array de números de longitud **numpuntos** multiplicado por 2. Los dos primeros miembros del array identifica las coordenadas *x* e *y* del primer punto, respectivamente, mientras que los dos siguientes especifican el siguiente punto, y así sucesivamente. La función *fillpoly* dibuja el perímetro del polígono con el estilo de línea y color actuales. Luego, el polígono es rellenado con la trama de relleno y color de relleno actuales.

Valor de retorno:

La función *fillpoly* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int trama, color;
   int puntos[6] = { 300, 50, 500, 300, 100, 300 };

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   trama = SLASH_FILL;
   color = 4;
   setfillstyle( trama, color );

   fillpoly( 3, puntos );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función floodfill Borland® C

Librería: graphics

```
void far floodfill(int x, int y, int borde);
```

Esta función es usada para rellenar un área cerrado con el color de relleno y trama de relleno actuales. Los argumentos **x** e **y** especifican el punto de comienzo para el algoritmo de relleno. El argumento **borde** especifica el valor del color del borde del área. Para que la función *fillpoly* funcione como es esperado, el área a ser rellenado debe estar rodeada por el color especificado por el argumento **borde.** Cuando el punto especificado por los argumentos **x** e **y** se encuentra dentro del área a ser rellenada, el interior será rellenado. Si se encuentra fuera del área, el exterior será rellenada.

Nota: Esta función no funciona con el driver IBM-8514.

Valor de retorno:

La función *floodfill* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int trama, color;
   int puntos[8] = { 300, 50, 500, 300, 100, 300, 300, 50 };

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setcolor( 10 );
   drawpoly( 4, puntos );

   trama = SLASH_FILL;
   color = 4;
   setfillstyle( trama, color );
   floodfill( 400, 250, 10 );

   getch();    /* Pausa */
   closegraph();
```

```
    return 0;
}
```

Función getarccoords Borland® C

Librería: graphics

```
void far getarccoords(struct arccoordstype far *coordenadas_arco);
```

Esta función es usada para recoger las coordenadas del centro, y los puntos del comienzo y final de la última llamada con éxito a la función arc. El argumento ***coordenadas_arco** apunta a la estructura de tipo **arccoordstype** que guarda la información recogida. La síntaxis de la estructura arccoordstype es:

```
struct arccoordstype {
   int x, y;
   int xstart, ystart;
   int xend, yend;
};
```

Los miembros **x** e **y** definen el centro del arco. Los miembros **xstart** e **ystart** definen las coordenadas *x* e *y* del punto de comienzo del arco. Similarmente, los miembros **xend** e **yend** definen las coordenadas *x* e *y* del punto de final del arco.

Valor de retorno:

La función *getarccoords* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int radio;
   struct arccoordstype info_arco;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   for( radio=25; radio<=100; radio+=25 ) {
      arc( 300, 150, 45, 315, radio );
      getarccoords( &info_arco );
      moveto( info_arco.xstart, info_arco.ystart );
      lineto( info_arco.xend, info_arco.yend );
```

```
    }

    getch();    /* Pausa */
    closegraph();

    return 0;
}
```

Función getaspectratio Borland® C

Librería: graphics

```
void far getaspectratio(int far *x_proporcion,
   int far *y_proporcion);
```

Esta función es usada para obtener la proporción anchura-altura del modo gráfico actual. La proporción anchura-altura puede definirse como la proporción de la anchura del píxel del modo gráfico y la altura del píxel. Esta proporción, usando los modos gráficos existentes, es siempre menor o igual que 1. El valor para determinar la proporción anchura-altura con respecto al eje horizontal es retornado en el argumento ***x_proporcion**. Similarmente, el valor para el eje vertical es retornado en el argumento ***y_proporcion**. El argumento ***y_proporcion** es asignado 10000, el cual es retornado cuando se llama a la función *getaspectratio*. El argumento ***x_proporcion** es casi siempre menor que el valor de ***y_proporcion**. Esto es debido al hecho de que la mayoría de los modos gráficos tiene píxels más altos que anchos. La única excepción es en los modos de VGA que produce píxels cuadrados; es decir, **x_proporcion** = **y_proporcion**.

Valor de retorno:

La función *getaspectratio* no retorna ningún valor, directamente.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x_proporcion, y_proporcion;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   getaspectratio( &x_proporcion, &y_proporcion );
   circle( 300, 150, 50 );

   getch();    /* Pausa */
   closegraph();

   printf( "Proporción anchura-altura.\nFactor x: %d\tFactor y: %d\n",
      x_proporcion, y_proporcion );

   return 0;
```

```
}
```

Función getbkcolor Borland® C

Librería: graphics

```
int far getbkcolor(void);
```

Esta función es usada para obtener el valor del color de fondo actual. El color de fondo, por defecto, es el color 0. Sin embargo, este valor puede cambiar con una llamada a la función setbkcolor.

Existen varios valores para ciertos colores de fondo.

Valor de retorno:

La función *getbkcolor* retorna el valor del color de fondo actual.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int color;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setbkcolor( 4 );
   circle( 300, 150, 50 );

   color = getbkcolor();

   getch();    /* Pausa */
   closegraph();

   printf( "Color de fondo: %d\n", color );

   return 0;
}
```

Función getch Borland® C

Librería: conio

```
int getch(void);
```

Lee un solo carácter directamente desde el teclado, sin mostrar tal carácter en pantalla.

Valor de retorno:

La función *getch* retorna el carácter leído desde el teclado.

Ejemplo:

```
#include <conio.h>

int main() {
   clrscr();
   cprintf( "Ejemplo de \"getch\"\r\n\r\n" );
   cprintf( "Pulsa una tecla: " );
   cprintf( "\'%c\'\r\n", getch() );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();

   return 0;
}
```

Función getche Borland® C

Librería: conio

```
int getche(void);
```

Lee un solo carácter directamente desde el teclado, mostrando tal carácter en pantalla, a través de la BIOS o por directamente a vídeo.

Valor de retorno:

La función *getche* retorna el carácter leído del teclado.

Ejemplo:

```
#include <conio.h>

int main() {
   clrscr();
   cprintf( "Ejemplo de \"getch\"\r\n\r\n" );
   cprintf( "Pulsa una tecla: " );
   cprintf( "\'%c\'\r\n", getch() );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();

   return 0;
}
```

Función getcolor Borland® C

Librería: graphics

```
int far getcolor(void);
```

Esta función obtiene el valor del color actual. El color actual es el color usado para dibujar líneas, arcos, etc.. Este color no es el mismo que el color de relleno. El valor del color obtenido es interpretado según el modo que esté en uso.

Existen varios valores para ciertos colores de fondo.

Valor de retorno:

La función *getcolor* retorna el valor del color actual.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int color;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setcolor( 4 );
   circle( 300, 150, 50 );

   color = getcolor();

   getch();    /* Pausa */
   closegraph();

   printf( "Color del perímetro: %d\n", color );

   return 0;
}
```

Función getdefaultpalette Borland® C

Librería: graphics

```
struct palettetype far *getdefaultpalette(void);
```

Esta función es usada para obtener una estructura que define la paleta según el dispositivo en la inicialización - esto es, cuando se llama a initgraph. La estructura palettetype se define de la siguiente manera:

```
#define MAXCOLORS 15

struct palettetype {
   unsigned char size;
   signed char colors[MAXCOLORS+1];
}
```

El campo **size** indica el tamaño de la paleta. El campo **colors** contiene los valores numéricos que representan los colores que ofrece el dispositivo en su paleta de colores.

Valor de retorno:

La función *getdefaultpalette* retorna un puntero a una estructura del tipo **palettetype**.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h&t;

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct palettetype *palette = NULL;
   int i;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   palette = getpalettetype();
   circle( 300, 150, 50 );

   getch();    /* Pausa */
   closegraph();
```

```
   printf( "Paleta\n\nTamaño: %d\nColores: %d",
      palette->size, palette->colors[0] );
   for( i=1; i<palette->size; i++ )
      printf( ", %d", palette->colors[i] );
   printf( "\n" );

   return 0;
}
```

Función getdrivername Borland® C

Librería: graphics

```
char *far getdrivername(void);
```

Esta función es usada para obtener una cadena de caracteres que contiene el nombre del dispositivo gráfico actual. Este función debería ser llamada después de que un dispositivo haya sido definido e inicializado - esto es, después de llamar a initgraph.

Valor de retorno:

La función *getdrivername* retorna una cadena de caracteres conteniendo el nombre del dispositivo gráfico.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char *nombre;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   strcpy( nombre, getdrivername() );
   circle( 300, 150, 50 );

   getch();    /* Pausa */
   closegraph();

   printf( "Nombre del dispositivo gráfico: %s\n", nombre );

   return 0;
}
```

Función getfillpattern Borland® C

Librería: graphics

```
void far getfillpattern(char far *trama);
```

Esta función es usada para obtener una trama de relleno definido por el usuario, como es definida por la función setfillpattern, y la guarda en memoria. El argumento ***trama** es un puntero a una serie de ocho bytes que representa una trama de relleno de bits de 8 x 8. Cada byte representa una fila de ocho bits, donde cada bit está encendido o no (1 ó 0). Un bit de 0 indica que el píxel correspondiente será asignado el color de relleno actual. Un bit de 0 indica que el píxel correspondiente no será alterado.

Valor de retorno:

La función *getfillpattern* no retorna ningún valor, directamente.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char trama1[8] = { 0x33, 0xEE, 0x33, 0xEE, 0x33, 0xEE, 0x33, 0xEE };
   char trama2[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   getfillpattern( trama2 );
   bar( 50, 50, 150, 150 );

   setfillpattern( trama1, 9 );
   bar( 160, 50, 260, 150 );

   setfillpattern( trama2, 4 );
   bar( 105, 160, 205, 260 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función getfillsettings Borland® C

Librería: graphics

```
void far getfillsettings(struct fillsettingstype far *info);
```

Esta función es usada para obtener la información de tramas de relleno. El argumento ***info** apunta a una estructura de tipo fillsettingstype, el cual es actualizado cuando se llama a la función *getfillsettings*. La estructura es:

```
struct fillsettingstype {
   int pattern;
   int color;
};
```

El campo **pattern** es la trama y el campo **color** es el color de relleno de la trama.

Existen trece valores ya definidos para tramas.

Valor de retorno:

La función *getfillsettings* no retorna ningún valor, directamente.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct fillsettingstype info;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   getfillsettings( &info );
   bar( 50, 50, 350, 300 );

   getch();    /* Pausa */
   closegraph();
```

```
   printf( "Trama de relleno: %d\tColor de relleno: %d\n",
      info.pattern, info.color );

   return 0;
}
```

Función getgraphmode Borland® C

Librería: graphics

```
int far getgraphmode(void);
```

Esta función es usada para obtener el valor del modo gráfico actual. El dispositivo actual debe ser considerado cuando se interprete el valor de retorno. Esta función sólo debería ser llamada después de que el sistema gráfico haya sido inicialiazado con la función initgraph.

Existen varios valores para los modos de cada dispositivo.

Valor de retorno:

La función *getgraphmode* retorna el modo gráfico como es establecido por initgraph o setgraphmode.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int modo;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   modo = getgraphmode();
   bar( 50, 50, 350, 300 );

   getch();    /* Pausa */
   closegraph();

   printf( "Modo gráfico: %d\n", modo );

   return 0;
}
```

Función getimage Borland® C

Librería: graphics

```
void far getimage(int izquierda, int superior,
   int derecha, int inferior, void far *imagen);
```

Esta función es usada para guardar una porción rectangular de la pantalla para un uso posterior. La esquina superior izquierda del área rectangular que ha de ser guardada está definida por los argumentos **izquierda** y **superior**. Estos argumentos representan las coordenades *x* e *y* de la esquina superior izquierda, respectivamente. Los argumentos **derecha** e **inferior** definen la esquina inferior derecha de la imagen rectangular. Estos argumentos definen las coordenades *x* e *y* de la esquina inferior derecha, respectivamente. El argumento ***image** apunta al búfer de memoria donde la imagen está guardada.

Valor de retorno:

La función *getimage* no retorna ningún valor, directamente.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   void *imagen;
   int imagentam;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setfillstyle( SLASH_FILL, 5 );
   bar( 50, 50, 350, 300 );

   imagentam = imagesize( 50, 50, 100, 100 );
   imagen = malloc( imagentam );
   getimage( 50, 50, 100, 100, imagen );

   putimage( 400, 50, imagen, COPY_PUT );
   putimage( 400, 110, imagen, COPY_PUT );

   getch();    /* Pausa */
   closegraph();
```

```
   free( imagen );

   return 0;
}
```

Función getlinesettings Borland® C

Librería: graphics

```
void far getlinesettings(struct linesettingstype far *info);
```

Esta función obtiene la información actual para las líneas. Esta información es guardada en una estructura de tipo linesettingstype que es apuntado por el argumento ***info**. El estilo de línea, trama, y grosor actuales son guardados en esta estructura. La síntaxis para la estructura **linesettingstype**:

```
struct linesettingstype {
   int linestyle;
   unsigned upattern;
   int thickness;
}
```

El campo **linestyle** es el estilo de la línea recta. El campo **upattern** es la trama de la línea del usuario solamente cuando el campo **linestyle** es igual a **USERBIT_LINE**, ó 4. Cuando esto sea el caso, el miembro **upattern** contiene una trama de línea definido por el usuario de 16 bits. Un bit 1 en esta trama indica que el píxel correspondiente será asignado el color actual. Un bit 0 indica que el píxel correspondiente no será alterado. El campo **thickness** es el grosor de la línea.

Existen varios valores para los diferentes estilos y grosores de líneas rectas.

Valor de retorno:

La función *getlinesettings* no retorna ningún valor, directamente.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct linesettingstype info;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setlinestyle( DOTTED_LINE, 0xFF33, THICK_WIDTH );
```

```
   circle( 350, 250, 50 );

   getlinesettings( &info );

   getch();    /* Pausa */
   closegraph();

   printf( "Líneas rectas.\nEstilo: %d\tTrama: %X\tGrosor: %d\n",
      info.linestyle, info.upattern, info.thickness  );

   return 0;
}
```

Función getmaxcolor Borland® C

Librería: graphics

```
int far getmaxcolor(void);
```

Esta función es usada para obtener el valor más alto de color en la paleta actual. La paleta en uso depende del dispositivo y modo inicializados. Para los modos de 16 colores, el valor de retorno es 15. Similarmente, para los modos de dos colores, el valor de retorno es 1.

Valor de retorno:

La función *getmaxcolor* retorna el valor máximo del color en la paleta en uso.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int color_max;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   color_max = getmaxcolor();

   closegraph();

   printf( "Color máximo: %d\n", color_max );

   return 0;
}
```

Función getmaxx Borland® C

Librería: graphics

```
int far getmaxx(void);
```

Esta función es usada para obtener la coordenada máxima de la pantalla en la dirección horizontal. Este valor suele ser la resolución horizontal máxima menos 1.

Valor de retorno:

La función *getmaxx* retorna la coordenada máxima de la pantalla en la dirección horizontal.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x_max;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   x_max = getmaxx();

   closegraph();

   printf( "X máxima: %d\n", x_max );

   return 0;
}
```

Función getmaxy Borland® C

Librería: graphics

```
int far getmaxy(void);
```

Esta función es usada para obtener la coordenada máxima de la pantalla en la dirección vertictal. Este valor suele ser la resolución vertictal máxima menos 1.

Valor de retorno:

La función *getmaxy* retorna la coordenada máxima de la pantalla en la dirección vertictal.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x_max, y_max;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   x_max = getmaxx();
   y_max = getmaxy();

   closegraph();

   printf( "X máxima: %d\tY máxima: %d\n", x_max, y_max );

   return 0;
}
```

Función getmodename Borland® C

Librería: graphics

```
char *far getmodename(int num_modo);
```

Esta función es usada para obtener el nombre del modo gráfico especificado por el argumento **num_modo**.

Valor de retorno:

La función *getmodename* retorna el nombre del modo gráfico que está contenido en todos los dispositivos gráficos.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char *nombre;
   int num_modo;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   num_modo = getgraphmode();
   strcpy( nombre, getmodename( num_modo ) );

   closegraph();

   printf( "X máxima: %d\tY máxima: %d\n", x_max, y_max );

   return 0;
}
```

Función getmoderange Borland® C

Librería: graphics

```
void far getmoderange(int driver,
   int far *modo_bajo, int far *modo_alto);
```

Esta función es usada para obtener los valores altos y bajos del modo gráfico del dispositivo especificado por el argumento **driver**. El valor más bajo del modo es retornado en ***modo_bajo**, y el valor más alto del modo es retornado en ***modo_alto**. Si el dispositivo especificado es inválido, el valor de -1 es retornado en ambos argumentos: ***modo_bajo** y ***modo_alto**. Sin embargo, si el argumento **driver** es asignado -1, los modos alto y bajo del dispositivo actual son retornados.

Valor de retorno:

La función *getmoderange* no retorna ningún valor, directamente.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int modo_bajo, modo_alto;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   getmoderange( gdriver, &modo_bajo, &modo_alto );

   closegraph();

   printf( "Alcance de modos, de %d á %d\n", modo_bajo, modo_alto );

   return 0;
}
```

Función getpalette Borland® C

Librería: graphics

```
void far getpalette(struct palettetype far *paleta);
```

Esta función es usada para obtener la información de la paleta actual. El argumento ***paleta** apunta a una estructura del tipo palettetype donde la información de la paleta es guardada. La estructura **palettetype** se define de la siguiente manera:

```
#define MAXCOLORS 15

struct palettetype {
   unsigned char size;
   signed char colors[MAXCOLORS+1];
}
```

El campo **size** indica el número de colores en la paleta. El campo **colors** contiene los valores numéricos que representan los colores que ofrece el dispositivo en su paleta de colores.

Valor de retorno:

La función *getpalette* no retorna ningún valor, directamente.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct palettetype palette;
   int i;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   getpalette( &palette );

   closegraph();

   printf( "Paleta\n\nTamaño: %d\nColores: %d", palette.size, palette.colors[0] );
   for( i=1; i<palette.size; i++ )
      printf( ", %d", palette.colors[i] );
   printf( "\n" );
```

```
    return 0;
}
```

Función getpalettesize Borland® C

Librería: graphics

```
int far getpalettesize(void);
```

Esta función es usada para obtener el número de entradas de paleta válidas para la paleta actual, considerando el modo gráfico en uso.

Valor de retorno:

La función *getpalettesize* retorna el número de colores en la paleta actual. Para modos de 16 colores, la función *getpalettesize* retorna 16.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int num_colores;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   num_colores = getpalettesize();

   closegraph();

   printf( "Paleta\n\nNúmero de colores: %d\n", num_colores );

   return 0;
}
```

Función getpass Borland® C

Librería: conio

```
char *getpass(const char *mensaje);
```

Lee una contraseña desde la consola del sistema después de mostrar un mensaje, el cual es una cadena de caracteres (terminada en un carácter nulo) apuntada por el argumento **mensaje** y desactivando la salida de texto.

Valor de retorno:

La función *getpass* retorna un puntero estático a la cadena de caracteres con el carácter nulo al final conteniendo la contraseña. Esta cadena contiene hasta ocho caracteres, sin contar el carácter nulo. Cada vez que la función *getpass* es llamada, la cadena de caracteres es sobrescrita.

Ejemplo:

```
#include <conio.h>

int main() {
   char *contra;

   clrscr();
   cprintf( "Ejemplo de \"getpass\"\r\n\r\n" );
   contra = getpass( "Introduzca la contraseña: " );
   cprintf( "La contraseña escrita: \'%s\'\r\n", contra );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();

   return 0;
}
```

Función getpixel Borland® C

Librería: graphics

```
unsigned far getpixel(int x, int y);
```

Esta función es usada para obtener el valor del color del píxel especificado por los argumentos **x** e **y**. Estos argumentos especifican las coordenadas de la pantalla del píxel a ser evaluado. Cuando se evalúa el valor del color retornado, el modo gráfico en uso debe ser considerado.

Existen varios valores para describir colores.

Valor de retorno:

La función *getpixel* retorna el número del color del píxel especificado.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x, y, color;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   x = 300;
   y = 100;

   setfillstyle( SOLID_FILL, 2 );
   fillellipse( 300, 160, 50, 150 );
   color = getpixel( x, y );

   getch();
   closegraph();

   printf( "Colores\n\nEl color del píxel (%d,%d): %d\n", x, y, color );

   return 0;
}
```

Función gettext Borland® C

Librería: conio

```
int gettext(int izquierda, int superior, int derecha,
   int inferior, void *destino);
```

Guarda el contenido en un rectángulo de texto en pantalla definido por los argumentos **izquierda** y **superior**, que describen la esquina superior izquierda y por los argumentos **derecha** e **inferior**, que describen la esquina inferior derecha, en el área de memoria apuntada por el argumento **destino**. Todas las coordenadas son coordenadas absolutas de pantalla; no son relativas a la ventana. La esquina superior izquierda es (1,1). La función *gettext* lee el contenido en este rectángulo en memoria secuencialmente de izquierda a derecha y de arriba a abajo. Cada posición en pantalla necesita 2 bytes de memoria. El primer byte es el carácter en la celda y el segundo es el atributo de vídeo de la celda. El espacio requerido para un rectángulo *b* columnas anchas y *h* filas altas está definida como:
bytes = (h filas) x (w anchas) x 2.

Valor de retorno:

La función *gettext* retorna 1 si la operación tiene éxito. Si ocurre un error, como es el caso de acceder fuera de la pantalla, entonces retorna el valor de 0.

Ejemplo:

```
#include <conio.h>
#include <stdlib.h>

#define ANCHURA 15
#define ALTURA 5

int main() {
   void *destino;

   destino = malloc( ALTURA*ANCHURA*2 );
   clrscr();
   cprintf( "Ejemplo de \"gettext\" y \"puttext\"\r\n\r\n" );
   cprintf( "El rectángulo será de un área relativamente pequeña.\r\n" );
   cprintf( "Las dimensiones son: %d (de ancho) x %d (de alto).\r\n",
      ANCHURA, ALTURA );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();

   cprintf( "\"gettext\" ha retornado: %d\r\n",
      gettext( 1, 1, 1+ANCHURA, 1+ALTURA, destino ) );
   cprintf( "Mostremos lo que hay guardado en memoria:" );
   puttext( 1, 9, 1+ANCHURA, 9+ALTURA, destino );
   getch();
   clrscr();
```

```
   free( destino );

   return 0;
}
```

Función gettextinfo Borland® C

Librería: conio

```
void gettextinfo(struct text_info *ti);
```

Obtiene la información de vídeo del modo texto. Esta información es guardada en una estructura apuntada por el argumento ***ti**. La estructura text_info se define de esta manera:

```
struct text_info {
   unsigned char winleft;        /* Coordenada izquierda de la ventana */
   unsigned char wintop;         /* Coordenada superior de la ventana */
   unsigned char winright;       /* Coordenada derecha de la ventana */
   unsigned char winbottom;      /* Coordenada inferior de la ventana */
   unsigned char attribute;      /* Atributo de texto */
   unsigned char normattr;       /* Atributo normal*/
   unsigned char currmode;       /* Modo en Uso: BW40, BW80, C40, C80, ó C4350 */
   unsigned char screenheight;   /* Altura de la pantalla de texto */
   unsigned char screenwidth;    /* Anchura de la pantalla de texto */
   unsigned char curx;           /* Coordenada X de la ventana en uso */
   unsigned char cury;           /* Coordenada Y de la ventana en uso */
};
```

Valor de retorno:

La función *gettextinfo* no retorna ningún valor, directamente.

Ejemplo:

```
#include <conio.h>

int main() {
   struct text_info *ti;

   gettextinfo( ti );
   clrscr();
   cprintf( "Ejemplo de \"gettextinfo\"\r\n\r\n" );
   cprintf( "Dimensiones de la ventana: " );
   cprintf( "(%d,%d) á (%d,%d)\r\n", ti->winleft, ti->wintop,
      ti->winright, ti->winbottom );
   cprintf( "Atributo: %d  Normal: %d\r\n", ti->attribute, ti->normattr );
   cprintf( "Modo en uso: %d\r\n", ti->currmode );
   cprintf( "Dimensiones de la pantalla: %d x %d\r\n",
      ti->screenwidth, ti->screenheight );
   cprintf( "Coordenadas de la ventana: (%d,%d)\r\n", ti->curx, ti->cury );
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Función gettextsettings Borland® C

Librería: graphics

```
void far gettextsettings(struct textsettingstype far *info);
```

Esta función es usada para obtener información acerca de la fuente gráfica actual. Esta información es guardada en una estructura de tipo textsettingstype, la cual es apuntada por el argumento ***info**. Esta estructura contiene información de la fuente actual en uso, la orientación del texto, el tamaño del carácter, y la justificación horizontal y vertical. La síntaxis de la estructura **textsettingstype** es la siguiente:

```
struct textsettingstype {
   int font;
   int direction;
   int charsize;
   int horiz;
   int vert;
};
```

Existen varios valores para describir el tipo, la orientación, y justificación de fuentes.

Valor de retorno:

La función *gettextsettings* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct textsettingstype info;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   gettextsettings( &info );

   closegraph();

   printf( "Texto\n\nFuente: %d\tSentido: %d\tTamaño: %d\n"
```

```
            "Justificación:\nHorizontal: %d, Vertical: %d\n",
            info.font, info.direction, info.charsize, info.horiz, info.vert);

    return 0;
}
```

Función getviewsettings Borland® C

Librería: graphics

```
void far getviewsettings(struct viewporttype far *info);
```

Esta función es usada para obtener información acerca del área gráfica actual. Esta información es guardada en una estructura de tipo viewporttype, la cual es apuntada por el argumento ***info**. Esta estructura contiene información acerca de las esquinas superior izquierda e inferior derecha, también como el banderín de recorte del área gráfica. La síntaxis de la estructura **viewporttype** es la siguiente:

```
struct viewporttype {
   int left, top;
   int right, bottom;
   int clip;
};
```

Valor de retorno:

La función *getviewsettings* no retorna ningún valor, directamente.

Ejemplo:

```
#include <graphics.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct viewporttype info;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   getviewsettings( &info );

   closegraph();

   printf( "Pantalla\n\nIzquierda: %d\tSuperior: %d\tDerecha: %d\t"
           "Inferior: %d\tBanderín: %d\n",
           info.left, info.top, info.right, info.bottom, info.clip );

   return 0;
}
```

Función getx Borland® C

Librería: graphics

```
int far getx(void);
```

Esta función es usada para obtener la posición, en la dirección horizontal, del cursor gráfico. El valor retornado especifica el lugar del píxel horizontal del cursor gráfico (la coordenada *x*), relativo a la pantalla del usuario actual.

Valor de retorno:

La función *getx* retorna la coordenada *x* del cursor gráfico.

Ejemplo:

```
#include <graphics.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x, y;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   moveto( 300, 150 );
   x = getx();
   y = gety();

   closegraph();

   printf( "Cursor gráfico\n\nX: %d\tY: %d\n", x, y );

   return 0;
}
```

Función gety Borland® C

Librería: graphics

```
int far gety(void);
```

Esta función es usada para obtener la posición, en la dirección vertical, del cursor gráfico. El valor retornado especifica el lugar del píxel vertical del cursor gráfico (la coordenada *y*), relativo a la pantalla del usuario actual.

Valor de retorno:

La función *gety* retorna la coordenada *y* del cursor gráfico.

Ejemplo:

```
#include <graphics.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x, y;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   moveto( 300, 150 );
   x = getx();
   y = gety();

   closegraph();

   printf( "Cursor gráfico\n\nX: %d\tY: %d\n", x, y );

   return 0;
}
```

Función gotoxy Borland® C

Librería: conio

```
void gotoxy(int x, int y);
```

Mueve el cursor de la ventana de texto a la posición según las coordenadas especificadas por los argumentos **x** e **y**. Si las coordenadas no son válidas entonces la llamda a la función *gotoxy* es ignorada. Los argumentos no pueden ser 0.

Valor de retorno:

La función *gotoxy* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>

int main() {
   clrscr();
   cprintf( "Ejemplo de \"gotoxy\"\r\n\r\n" );
   cprintf( "1ª línea" );
   cprintf( "2ª línea" );
   gotoxy( 5, 20 );
   cprintf( "3ª línea" );
   gotoxy( 20, 1 );
   cprintf( "4ª línea" );
   gotoxy( 1, 15 );
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Función graphdefaults Borland® C

Librería: graphics

```
void far graphdefaults(void);
```

Esta función es usada para reiniciar todos los datos gráficos a sus valores originales, o por defecto. La función *graphdefaults* reinicia la pantalla del usuario para que cubra la pantalla entera, mueve el cursor a la posición (0,0), y reinicia la paleta actual a sus colores por defecto. También reinicia el color de fondo y el actual a sus valores por defecto, reinicia el estilo y trama de relleno a sus valores por defecto, y reinicia la fuente y justificación de texto.

Valor de retorno:

La función *graphdefaults* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setcolor( 4 );
   setviewport( 250, 150, 350, 250, 1 );
   graphdefaults();
   circle( 300, 200, 50 );

   getch();
   closegraph();

   return 0;
}
```

Función grapherrormsg Borland® C

Librería: graphics

```
char *far grapherrormsg(int codigo_error);
```

Esta función es usada para obtener una cadena de caracteres conteniendo el mensaje de error para un código de error especificado. El argumento **codigo_error** especifica el valor del código de error. La función graphresult debe ser usada para obtener el código de error usado para el argumento **codigo_error**.

Valor de retorno:

La función *grapherrormsg* retorna una cadena de caracteres.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int codigo_error;
   char *mensaje_error;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setgraphmode( 40 );    /* Creamos un ERROR */
   codigo_error = graphresult();
   strcpy( mensaje_error, grapherrormsg( codigo_error ) );

   closegraph();

   printf( "ERROR: \"%s\" (%d)\n", mensaje_error, codigo_error );

   return 0;
}
```

Función _graphfreemem Borland® C

Librería: graphics

```
void far _graphfreemem(void far *ptr, unsigned tamanyo);
```

Esta función es usada por la librería gráfica para desadjudicar memoria previamente reservada mediante una llamada a la función _graphgetmem. Esta función es llamada por la librería gráfica cuando se quiere liberar memoria. Por defecto, la función simplemente llama a *free*, pero se puede controlar la administración de memoria de la librería gráfica. La forma de hacer esto es simplemente creando la definición de la función, con el mismo prototipo mostrado aquí.

Valor de retorno:

La función *_graphfreemem* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

void far _graphfreemem( void far *ptr, unsigned tamanyo ) {
   printf( "__graphfreemem ha sido llamado para "
      "desadjudicar %d bytes en memoria\n" );
   printf( "para el montón (heap) interno\n", tamanyo );
   printf( "Pulse cualquier tecla...\n\n" );
   getch();
   free( ptr );
}

void far * far _graphgetmem( unsigned tamanyo ) {
   printf( "__graphgetmem ha sido llamado para "
      "adjudicar %d bytes en memoria\n" );
   printf( "para el montón (heap) interno\n", tamanyo );
   printf( "Pulse cualquier tecla...\n\n" );
   getch();
   return malloc( tamanyo );
}

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
```

```
   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   circle( 200, 100, 50 );

   getch();
   closegraph();

   return 0;
}
```

Función _graphgetmem Borland® C

Librería: graphics

```
void far * far _graphgetmem(unsigned tamanyo);
```

Esta función es usada por la librería gráfica para adjudicar memoria gráfica para búfers internos, dispositivos gráficos, y fuentes. Esta función tiene como intención ser llamada por la librería gráfica cuando se quiere adjudicar memoria. Por defecto, la función simplemente llama a *malloc*, pero se puede controlar la administración de memoria de la librería gráfica. La forma de hacer esto es simplemente creando la definición de la función, con el mismo prototipo mostrado aquí.

Valor de retorno:

La función *_graphgetmem* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>

void far _graphfreemem( void far *ptr, unsigned tamanyo ) {
   printf( "__graphfreemem ha sido llamado para "
      "desadjudicar %d bytes en memoria\n" );
   printf( "para el montón (heap) interno\n", tamanyo );
   printf( "Pulse cualquier tecla...\n\n" );
   getch();
   free( ptr );
}

void far * far _graphgetmem( unsigned tamanyo ) {
   printf( "__graphgetmem ha sido llamado para "
      "adjudicar %d bytes en memoria\n" );
   printf( "para el montón (heap) interno\n", tamanyo );
   printf( "Pulse cualquier tecla...\n\n" );
   getch();
   return malloc( tamanyo );
}

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */
```

```
   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   circle( 200, 100, 50 );

   getch();
   closegraph();

   return 0;
}
```

Función graphresult Borland® C

Librería: graphics

```
int far graphresult(void);
```

Esta función obtiene y retorna el código de error para la última llamada sin éxito. Además, reinicia el nivel de error a 0, o **grOk**.

Existen varios valores de códigos de error.

Valor de retorno:

La función *graphresult* retorna el código de error de la última llamada gráfica sin éxito.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int codigo_error;
   char *mensaje_error;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setgraphmode( 40 );    /* Creamos un ERROR */
   codigo_error = graphresult();
   strcpy( mensaje_error, grapherrormsg( codigo_error ) );

   closegraph();

   printf( "ERROR: \"%s\" (%d)\n", mensaje_error, codigo_error );

   return 0;
}
```

Función highvideo Borland® C

Librería: conio

```
void highvideo(void);
```

Selecciona los caracteres con una mayor intensidad mediante activando el bit de la mayor intensidad del color de primer plano en uso. La función *highvideo* no afecta cualesquiera de los caracteres actualmente en pantalla, pero sí afecta aquéllas mostradas por funciones que usan el vídeo directamente para la salida en modo texto después de llamar a la función *highvideo*.

Valor de retorno:

La función *highvideo* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>

int main() {
   clrscr();
   cprintf( "Ejemplo de \"highvideo\" y \"lowvideo\"\r\n\r\n" );
   cprintf( "Este texto tiene su propia intensidad.\r\n" );
   cprintf( "Ahora cambiaremos la intensidad.\r\n" );
   highvideo();
   cprintf( "La intensidad a partir de ahora es mayor.\r\n" );
   cprintf( "Ahora lo cambiaremos a una intensidad menor.\r\n" );
   lowvideo();
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Función imagesize Borland® C

Librería: graphics

```
unsigned far imagesize(int izquierda, int superior,
   int derecha, int inferior);
```

Esta función es usada para determinar el tamaño del búfer necesitado para almacenar una imagen con la función getimage. Los argumentos **izquierda** y **superior** definen las coordenadas *x* e *y* de la esquina superior izquierda de la imagen rectangular. Similarmente, los argumentos **derecha** y **inferior** definen las coordenadas *x* e *y* de la esquina inferior derecha de la imagen rectangular.

Valor de retorno:

La función *imagesize* retorna el número actual de bytes necesarios si el tamaño requerido es menor que 64 Kb menos 1 byte. Si esto no es el caso, el valor retornado es 0xFFFF, ó -1.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   void *imagen;
   int imagentam;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setfillstyle( SLASH_FILL, 5 );
   bar( 50, 50, 350, 300 );

   imagentam = imagesize( 50, 50, 100, 100 );
   imagen = malloc( imagentam );
   getimage( 50, 50, 100, 100, imagen );

   putimage( 400, 50, imagen, COPY_PUT );
   putimage( 400, 110, imagen, COPY_PUT );

   getch();    /* Pausa */
   closegraph();

   free( imagen );
```

```
    return 0;
}
```

Función initgraph Borland® C

Librería: graphics

```
void far initgraph(int far *driver,
   int far *modo, int far *path);
```

Esta función es usada para cargar o validar un dispositivo gráfico y cambiar el sistema de vídeo a modo gráfico. La función *initgraph* debe ser llamada antes de cualesquier funciones que generan una salida gráfica sean usadas.

Existen varios valores a ser usados para el argumento ***driver**. Si ***driver** es asignado a DETECT, ó 0, la función detectgraph es llamada, y un dispositivo y modo gráfico apropiados son seleccionados. Asignando a ***driver** cualquier otro valor predefinido inicia la carga del dispositivo gráfico correspondiente.

Existen varios valores a ser usados para el argumento ***modo**. Estos valores deberían corresponder al dispositivo especificado en el argumento ***driver**.

El argumento ***path** especificad el directorio donde los dispositivos gráficos están localizados. La función *initgraph* buscará el dispositivo primeramente en este directorio. Si no es encontrado, la función buscará en el directorio de inicio. Cuando el argumento ***path** es NULL, solamente el directorio de inicio es buscado.

Otra forma para evitar cargando el dispositivo desde el disco cada vez que el programa es ejecutado es ligarlo o enlazarlo al dispositivo apropiado en un programa ejecutable.

Valor de retorno:

La función *initgraph* no retorna ningún valor. Sin embargo, cuando la función *initgraph* es llamada, el código de error interno es activado. Si la función *initgraph* termina con éxito, el código es asignado un 0. Si no, el código es asignado así:

-2 grNotDetected	La tarjeta gráfica no se encontró
-3 grFileNotFound	El fichero del dispositivo no se encontró
-4 grInvalidDriver	El fichero del dispositivo es inválido
-5 grNoLoadMem	No hay suficiente memoria para cargar el dispositivo

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
```

```
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   circle( 300, 200, 80 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función inport Borland® C

Librería: conio

```
int inport(int id_puerto);
```

Lee 1 byte de la parte baja de 1 palabra (word) desde el puerto de entrada indicado por el argumento **id_puerto**; lee el byte alto desde **id_puerto**+1. La función *inport* funciona de la misma manera que la instrucción 80x86 *IN*.

Valor de retorno:

La función *inport* retorna el valor leído de una palabra (word) de tamaño desde el puerto apuntado por el argumento **id_puerto** e **id_puerto**+1.

Ejemplo:

```
#include <conio.h>

int main() {
   int valor, id_puerto=0;    /* Puerto de serie 0 */

   valor = inport( id_puerto );
   clrscr();
   cprintf( "Ejemplo de \"inport\"\r\n\r\n" );
   cprintf( "Leemos 1 word desde el puerto %d: 0x%X.\r\n", id_puerto, valor );
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Función insline Borland® C

Librería: conio

```
void insline(void);
```

Inserta una línea vacía en la ventana de texto en la posición del cursor usando el color de fondo de texto en uso. Todas las líneas debajo de la vacía son mudadas una línea más abajo, y la línea inferior es mudada fuera de la ventana.

Valor de retorno:

La función *insline* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>

int main() {
   clrscr();
   cprintf( "Ejemplo de \"inline\"\r\n\r\n" );
   cprintf( "Añadiremos una línea vacía depués de ésta.\r\n" );
   insline();
   cprintf( "\r\nPulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Función installuserdriver Borland® C

Librería: graphics

```
int far installuserdriver(char far *nombre,
   int huge (*detectar)(void));
```

Esta función permite al usuario añadir dispositivos adicionales de otras compañías o grupos a la tabla interna BGI de los dispositivos. El argumento ***nombre** define el nombre del fichero nuevo del dispositivo .BGI. El parámetro ***detectar** es un puntero a una función opcional para autodetectar que puede ser o no ser provisto con el dispositivo nuevo. La función de autodetectación espera no recibir ningún parámetro y retorna un valor entero.

Valor de retorno:

La función *installuserdriver* retorna el parámetro del número del dispositivo que hubiese sido pasado a la función initgraph para seleccionar un dispositivo nuevo.

Ejemplo:

```
/* Este programa no funcionará, ya que se
** necesitaría otra tarjeta gráfica
** desconocida por las librerías gráficas de BGI.
** Esto sólo es para poner un ejemplo.
*/
#include <graphics.h>

int huge detectarSMGGA( void ) {
   int driver, modo, modo_sugerirdo=0;

   detectgraph( &driver, &modo );
   if( SMGGA == driver )   return modo_sugerido;

   return grError;
}

int main() {
   int gdriver, gmodo;
```

```
/* Intentamos instalar nuestra tarjeta gráfica:
** Súper Mega Guay Graphics Array (SMGGA)
** Ya sé que suena muy cursi, pero esto sólo es un ejemplo :)
*/
   gdriver = installuserdriver( "SMGGA", detectarSMGGA );

/* Forzamos a que use nuestra función para autodetectar */
   gdriver = DETECT;
   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   closegraph();

   return 0;
}
```

Función installuserfont Borland® C

Librería: graphics

```
int far installuserfont(char far *nombre);
```

Esta función carga un fichero de fuente escalable que no está provisto con el sistema BGI. El parámetro ***nombre** especifica el nombre del fichero fuente a cargar, en el directorio de inicio. El sistema gráfico puede tener hasta veinte fuentes instaladas a la vez.

Valor de retorno:

La función *installuserfont* retorna el número de identificación de la fuente que es usada para selccionar la fuente nueva a través de la función settextstyle. Si la tabla interna de fuentes está llena, un valor de -11 (grError) es retornado, indicando un error.

Ejemplo:

```
/* Este programa no funcionará, ya que se
** necesitaría tener una fuente nueva
** y desconocida por las librerías gráficas de BGI.
** Esto sólo es para poner un ejemplo.
*/
#include <graphics.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int fuente_SMGF;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

/* Intentamos instalar nuestra fuente nueva:
** Súper Mega Chula Fuente (SMGF)
** Ya sé que suena muy cursi, pero esto sólo es un ejemplo :)
*/
   if( (fuente_SMGF = installuserfont( "SMGF.CHR" )) != grError )
      settextstyle( fuente_SMGF, HORIZ_DIR, 4 );
   else
      settextstyle( DEFAULT_FONT, HORIZ_DIR, 4 );

   closegraph();

   return 0;
```

```
}
```

Función kbhit Borland® C

Librería: conio

```
int kbhit(void);
```

Revisa si una tecla pulsada está disponible. Cualesquier pulsadas disponibles pueden ser recogidas con las funciones getch o getche.

Valor de retorno:

La función *kbhit* retorna 0 si no se ha registrado una pulsada de tecla; si hay una disponible, entonces el valor retornado es distinto a cero.

Ejemplo:

```
#include <conio.h>

int main() {
   clrscr();
   cprintf( "Ejemplo de \"kbhit\"\r\n\r\n" );
   cprintf( "El programa está a la espera de registrar una tecla pulsada.\r\n" );
   while( !kbhit() );
   cprintf( "El carácter es \'%c\'\r\n", getch() );
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Función line Borland® C

Librería: graphics

```
void far line(int x1, int y1, int x2, int y2);
```

Esta función es usada para conectar dos puntos con una línea recta. El primer punto es especificado por los argumentos **x1** e **y1**. El segundo punto es especificado por los argumentos **x2** e **y2**. La línea se dibuja usando el estilo de línea actual, el grosor, y el color actual. La posición del cursor gráfico no es afectado por la función *line*.

Valor de retorno:

La función *line* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   line( 20, 40, 350, 100 );
   line( 400, 30, 50, 250 );

   getch();
   closegraph();

   return 0;
}
```

Función linerel Borland® C

Librería: graphics

```
void far linerel(int dx, int dy);
```

Esta función es usada para dibujar una línea recta a una distancia y dirección predeterminadas desde la posición actual del cursor gráfico. El argumento **dx** especifica el número relativo de píxels para atravesar en la dirección horizontal. El argumento **dy** especifica el número relativo de píxels para atravesar en la dirección vertical. Estos argumentos pueden ser tanto valores positivos como negativos. La línea se dibuja usando el estilo de línea actual, el grosor, y el color actual desde la posición actual del cursor gráfico a través de la distancia relativa especificada. Cuando la línea esté terminada, la posición del cursor gráfico es actualizado al último punto de la línea.

Valor de retorno:

La función *linerel* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   moveto( 20, 20 );
   linerel( 20, 40 );
   linerel( 40, 30 );

   getch();
   closegraph();

   return 0;
}
```

Función lineto Borland® C

Librería: graphics

```
void far lineto(int x, int y);
```

Esta función es usada para dibujar una línea recta desde la posición actual del cursor gráfico hasta el punto especificado por los argumentos **x** e **y**. La línea se dibuja usando el estilo de línea actual, el grosor, y el color actual. Después de que la línea recta haya sido dibujado, la posición del cursor gráfico es actualizado a la posición especificado por los argumentos **x** e **y** (el punto final de la línea).

Valor de retorno:

La función *lineto* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   moveto( 20, 20 );
   lineto( 40, 60 );
   lineto( 80, 90 );

   getch();
   closegraph();

   return 0;
}
```

Función lowvideo Borland® C

Librería: conio

```
void lowvideo(void);
```

Selecciona los caracteres con una menor intensidad mediante activando el bit de la menor intensidad del color de primer plano en uso. La función *lowvideo* no afecta cualesquiera de los caracteres actualmente en pantalla, pero sí afecta aquéllas mostradas por funciones que usan el vídeo directamente para la salida en modo texto después de llamar a la función *lowvideo*.

Valor de retorno:

La función *lowvideo* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>

int main() {
   clrscr();
   cprintf( "Ejemplo de \"highvideo\" y \"lowvideo\"\r\n\r\n" );
   cprintf( "Este texto tiene su propia intensidad.\r\n" );
   cprintf( "Ahora cambiaremos la intensidad.\r\n" );
   highvideo();
   cprintf( "La intensidad a partir de ahora es mayor.\r\n" );
   cprintf( "Ahora lo cambiaremos a una intensidad menor.\r\n" );
   lowvideo();
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Función moverel Borland® C

Librería: graphics

```
void far moverel(int dx, int dy);
```

Esta función es usada para mover la posición del cursor gráfico a una distancia relativa como los argumentos **dx** y **dy**. El argumento **dx** define la distancia relativa a moverse en la dirección horizontal. El argumento **dy** define la distancia relativa a moverse en la dirección vertical. Estos valores pueden ser positivos o negativos. No se dibuja ya que el cursor es mudado.

Valor de retorno:

La función *moverel* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   moveto( 20, 20 );
   linerel( 20, 40 );
   moverel( 50, 50 );
   linerel( 40, 30 );

   getch();
   closegraph();

   return 0;
}
```

Función movetext Borland® C

Librería: conio

```
int movetext(int izquierda, int superior, int derecha,
int inferior, int destino_izquierda, int destino_superior);
```

Copia el contenido en un rectángulo de texto en pantalla definido por los argumentos **izquierda** y **superior**, que describen la esquina superior izquierda y por los argumentos **derecha** e **inferior**, que describen la esquina inferior derecha, a otro rectángulo de iguales dimensiones. La esquina superior izquierda del nuevo rectángulo está especificada por los argumentos **destino_izquierda** y **destino_superior**. Todas las coordenadas son coordenadas absolutas de pantalla; no son relativas a la ventana. Los rectángulos que ocupan el mismo área son mudados acordemente. La función *movetext* usa la salida de vídeo directamente.

Valor de retorno:

La función *movetext* retorna un valor distinto a 0, si la operación tiene éxito. Si ocurre un error, como es el caso de acceder fuera de la pantalla, entonces retorna el valor de 0.

Ejemplo:

```
#include <conio.h>

#define ANCHURA 25
#define ALTURA 2

int main() {
   clrscr();
   cprintf( "Ejemplo de \"movetext\"\r\n\r\n" );
   cprintf( "El rectángulo será de un área relativamente pequeña.\r\n" );
   cprintf( "Copiaremos esta línea...\r\n...y ésta también.\r\n" );
   cprintf( "Las dimensiones son: %d (de ancho) x %d (de alto).\r\n", ANCHURA, ALTURA
);
   cprintf( "\"movetext\" ha retornado: %d\r\n", movetext( 1, 4, ANCHURA, 3+ALTURA,
5, 15 ) );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();
   clrscr();

   return 0;
}
```

Función moveto Borland® C

Librería: graphics

```
void far moveto(int x, int y);
```

Esta función es usada para colocar el cursor gráfico al punto especificado por los argumentos **x** e **y**. Ya que el cursor es movido desde su posición anterior al punto especificado por los argumentos **x** e **y**, no hay dibujo alguno.

Valor de retorno:

La función *moveto* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   moveto( 20, 20 );
   lineto( 40, 60 );
   lineto( 80, 90 );

   getch();
   closegraph();

   return 0;
}
```

Función normvideo Borland® C

Librería: conio

```
void normvideo(void);
```

Selecciona los caracteres con una intensidad normal mediante seleccionando el atributo del texto (primer plano y de fondo) al valor que tenía anteriormente al comienzo del programa. La función *normvideo* no afecta cualesquiera de los caracteres actualmente en pantalla, pero sí afecta aquéllas mostradas por funciones que usan el vídeo directamente para la salida en modo texto después de llamar a la función *normvideo*.

Valor de retorno:

La función *normvideo* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>

int main() {
   clrscr();
   cprintf( "Ejemplo de \"normvideo\"\r\n\r\n" );
   cprintf( "Este texto tiene su propia intensidad.\r\n" );
   cprintf( "Ahora cambiaremos la intensidad.\r\n" );
   normvideo();
   cprintf( "La intensidad a partir de ahora es mayor.\r\n" );
   cprintf( "Ahora lo cambiaremos a una intensidad menor.\r\n" );
   lowvideo();
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Función outport Borland® C

Librería: conio

```
int outport(int id_puerto, int valor);
```

Escribe el último byte de 1 palabra (word) al puerto de salida indicado por el argumento **id_puerto**; escribe el primer byte a **id_puerto**+1. La función *outport* funciona de la misma manera que la instrucción 80x86 *OUT*.

Valor de retorno:

La función *outport* retorna el valor escrito de una palabra (word) de tamaño al puerto apuntado por el argumento **id_puerto** e **id_puerto**+1.

Ejemplo:

```
#include <conio.h>

int main() {
   int valor=0xFFAA, id_puerto=0;    /* Puerto de serie 0 */

   outport( id_puerto, valor );
   clrscr();
   cprintf( "Ejemplo de \"outport\"\r\n\r\n" );
   cprintf( "Escribimos 1 word (2 bytes) al puerto %d: 0x%X.\r\n", id_puerto, valor
);
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Función outtext Borland® C

Librería: graphics

```
void far outtext(char far *cadena_texto);
```

Esta función es usada para mostar una cadena de caracteres. El argumento ***cadena_texto** define la cadena de texto a ser mostrado. La cadena es mostrado donde está el cursor gráfico actualmente usando el color actual y fuente, dirección, valores, y justificaciones de texto. La posición del cursor permanece sin ser cambiado al menos que la justificación horizontal actual es LEFT_TEXT y la orientación del texto es HORIZ_DIR. Cuando esto sea el caso, la posición del cursor es colocada horizontalmente a la anchura del píxel de la cadena de texto. Además, cuando se use la fuente por defecto, cualquier texto que se extiende a fuera del área gráfica actual es truncado.

Aunque la función *outtext* está diseñada para texto sin formato, texto con formato puede ser mostrada a través del uso de un búfer de caracteres y la función sprintf.

Valor de retorno:

La función *outtext* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char mensaje[40];
   char nombre[25];

   printf( "Escribe tu nombre: " );
   scanf( "%s", nombre );
   sprintf( mensaje, "Hola %s!", nombre );

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   outtext( mensaje );
   outtextxy( 300, 150, mensaje );

   getch();
   closegraph();
```

```
    return 0;
}
```

Función outtextxy Borland® C

Librería: graphics

```
void far outtextxy(int x, int y, char far *cadena_texto);
```

Esta función es usada para mostar una cadena de caracteres. El argumento ***cadena_texto** define la cadena de texto a ser mostrado. La cadena es mostrada en la posición descrita por los argumentos **x** e **y** usando el color actual y fuente, dirección, valores, y justificaciones de texto. Cuando se use la fuente por defecto, cualquier texto que se extiende fuera del área gráfica actual es truncado.

Aunque la función *outtextxy* está diseñada para texto sin formato, texto con formato puede ser mostrada a través del uso de un búfer de caracteres y la función sprintf.

Valor de retorno:

La función *outtextxy* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char mensaje[40];
   char nombre[25];

   printf( "Escribe tu nombre: " );
   scanf( "%s", nombre );
   sprintf( mensaje, "Hola %s!", nombre );

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   outtext( mensaje );
   outtextxy( 300, 150, mensaje );

   getch();
   closegraph();

   return 0;
}
```

Función pieslice Borland® C

Librería: graphics

```
void far pieslice(int x, int y,
   int comienzo_angulo, int final_angulo, int radio);
```

Esta función es usada para dibujar y rellenar un una cuña circular. La cuña circular está centrada en el punto especificado por los argumentos **x** e **y**. La porción circular de la cuña comienza con el ángulo especificado por el argumento **comienzo_angulo** y se extiende en un sentido contrario a las agujas del reloj al ángulo especificado por el argumento **final_angulo**. La función *pieslice* considera este - el eje horizontal a la derecha del centro - como su punto de referencia de 0 grados. El perímetro de la cuña es dibujado con el color actual y es rellenado con la trama y color de relleno actual.

Valor de retorno:

La función *pieslice* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   pieslice( 300, 150, 45, 225, 50 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función putch Borland® C

Librería: conio

```
int putch(int c);
```

Muesrta un carácter, especificado por el argumento **c**, directamente a la ventana de texto en uso. La función *putch* usa el vídeo directamente para mostrar caracteres. Esto se realiza mediante una escritura directa a la memoria de la pantalla o mediante una llamada a la BIOS, dependiendo del valor de la variable global *_directvideo*. Esta función no convierte los caracteres de nueva línea (\n) en la pareja de caracteres de retorno de línea/nueva línea (\r\n).

Valor de retorno:

La función *putch* retorna el carácter mostrado, si tiene éxito; si ocurre un error, entonces retorna EOF.

Ejemplo:

```
#include <conio.h>
#include <stdio.h>

int main() {
   char mensaje[8]="Borland";
   int i=0;

   clrscr();
   cprintf( "Ejemplo de \"putch\"\r\n\r\n" );
   cprintf( "El mensaje es: \"" );
   while( putch( mensaje[i++] ) != EOF );
   cprintf( "\"\r\nPulsa una tecla para continuar..." );
   getch();

   return 0;
}
```

Función putimage Borland® C

Librería: graphics

```
void far putimage(int izquierda, int superior,
   void far *imagen, int accion);
```

Esta función coloca una imagen que fue previamente guardada con la función getimage en la pantalla. La esquina superior izquierda donde será colocada la imagen está definida por los argumentos **izquierda** y **superior**. Estos argumentos representan las coordenades *x* e *y* de la esquina superior izquierda, respectivamente. El argumento ***image** apunta al búfer de memoria donde la imagen está guardada. La imagen se coloca en la pantalla con la acción defindia en el argumento **accion**. Los valores y consonantes usados por el argumento **accion** se describen a continuación:

Constante	Valor	Significado
COPY_PUT	0	Sobrescribir los píxels existentes
XOR_PUT	1	Operación OR Exclusivo con los píxels
OR_PUT	2	Operación OR Inclusivo con los píxels
AND_PUT	3	Operación AND con los píxels
NOT_PUT	4	Invertir la imagen

Valor de retorno:

La función *putimage* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdlib.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   void *imagen;
   int imagentam;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setfillstyle( SLASH_FILL, 5 );
   bar( 50, 50, 350, 300 );
```

```
    imagentam = imagesize( 50, 50, 100, 100 );
    imagen = malloc( imagentam );
    getimage( 50, 50, 100, 100, imagen );

    putimage( 400, 50, imagen, COPY_PUT );
    putimage( 400, 110, imagen, COPY_PUT );

    getch();    /* Pausa */
    closegraph();

    free( imagen );

    return 0;
}
```

Función putpixel Borland® C

Librería: graphics

```
void far putpixel(int x, int y, int color);
```

Esta función es usada para asignar el valor del color a un píxel en particular. La posición del píxel en cuestión está especificado por los argumentos **x** e **y**. El argumento **color** especifica el valor del color del píxel.

Existen varios valores para describir colores.

Valor de retorno:

La función *putpixel* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int t;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   for( t=0; t<200; t++ )
     putpixel( 100+t, 50+t, t%16 );

   getch();
   closegraph();

   return 0;
}
```

Función puttext Borland® C

Librería: conio

```
int puttext(int izquierda, int superior,
   int derecha, int inferior, void *origen);
```

Imprime el contenido en un rectángulo de texto en pantalla definido por los argumentos **izquierda** y **superior**, que describen la esquina superior izquierda y por los argumentos **derecha** e **inferior**, que describen la esquina inferior derecha, en el área de memoria apuntada por el argumento **origen**. Todas las coordenadas son coordenadas absolutas de pantalla; no son relativas a la ventana. La esquina superior izquierda es (1,1). La función *puttext* coloca el contenido de este rectángulo en memoria secuencialmente de izquierda a derecha y de arriba a abajo. Cada posición en pantalla contiene 2 bytes de memoria. El primer byte es el carácter en la celda y el segundo es el atributo de vídeo de la celda. El espacio requerido para un rectángulo *b* columnas anchas y *h* filas altas está definida como:
bytes = (h filas) x (w anchas) x 2.
La función *puttext* usa la salida directa de vídeo.

Valor de retorno:

La función *puttext* retorna un valor distinto a 0, si la operación tiene éxito. Si ocurre un error, como es el caso de acceder fuera de la pantalla, entonces retorna el valor de 0.

Ejemplo:

```
#include <conio.h>
#include <stdlib.h>

#define ANCHURA 15
#define ALTURA 5

int main() {
   void *destino;

   destino = malloc( ALTURA*ANCHURA*2 );
   clrscr();
   cprintf( "Ejemplo de \"gettext\" y \"puttext\"\r\n\r\n" );
   cprintf( "El rectángulo será de un área relativamente pequeña.\r\n" );
   cprintf( "Las dimensiones son: %d (de ancho) x %d (de alto).\r\n", ANCHURA, ALTURA
);
   cprintf( "Pulsa una tecla para continuar..." );
   getch();

   cprintf( "\"gettext\" ha retornado: %d\r\n", gettext( 1, 1, 1+ANCHURA, 1+ALTURA,
destino ) );
   cprintf( "Mostremos lo que hay guardado en memoria:" );
   puttext( 1, 9, 1+ANCHURA, 9+ALTURA, destino );
   getch();
   clrscr();

   free( destino );

   return 0;
}
```

Función rectangle Borland® C

Librería: graphics

```
void far rectangle(int izquierda,
   int superior, int derecha, int inferior);
```

Esta función dibujará un rectángulo sin rellenar su interior usando el color actual. La esquina superior izquierda del rectángulo está definida por los argumentos **izquierda** y **superior**. Estos argumentos corresponden a los valores *x* e *y* de la esquina superior izquierda. Similarmente, los argumentos **derecha** e **inferior** definen la esquina inferior derecha del rectángulo. El perímetro del rectángulo es dibujado usando el estilo y grosor de línea actuales.

Valor de retorno:

La función *rectangle* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   rectangle( 20, 20, 400, 300 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función registerbgidriver Borland® C

Librería: graphics

```
int registerbgidriver(void (*driver)(void));
```

Esta función es usada para cargar y registrar un dispositivo gráfico. El argumento ***driver** apunta a un dispositivo. Un fichero de dispositivo registrado puede ser tanto cargado desde el disco o convertido en un formato .OBJ y ligado (o enlazado) dentro del programa. Registrando el dispositivo de esta manera, el fichero .EXE no depende de un fichero externo de dispositivo para poder ejecutarse.

Valor de retorno:

La función *registerbgidriver* retorna número del dispositivo cuando tiene éxito. Un código de error, un número negativo, es retornado si el dispositivo especificado es inválido.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:
*/
   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );


/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );
*/
   rectangle( 20, 20, 400, 300 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función registerbgifont Borland® C

Librería: graphics

```
int registerbgifont(void (*fuente)(void));
```

Esta función es usada para informar al sistema que la fuente apuntada por el argumento ***fuente** fue incluida durante el enlazo. Un fichero de fuente registrado ha de ser convertido en un fichero objeto .OBJ y ligado (o enlazado) dentro del programa. Registrando la fuente de esta manera, el fichero .EXE no depende de un fichero externo de fuentes para poder ejecutarse.

Nota: La fuente de defecto es la única que está disponible en el programa, ya que forma parte del sistema gráfico; no es necesario ligarlo al programa.

Valor de retorno:

La función *registerbgifont* retorna número del dispositivo cuando tiene éxito. Un código de error, un número negativo, es retornado si el dispositivo especificado es inválido.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos y fuente para que formen parte de graphics.lib
** entonces usa estas sentencias:
*/
   registerbgidriver( EGAVGA_driver );
   registerbgifont( sansserif_font );
   initgraph( &gdriver, &gmodo, "" );


/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );
*/

   outtext( "Esto es una prueba con la fuente \"Sans Serif\"" );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función restorecrtmode Borland® C

Librería: graphics

```
void far restorecrtmode(void);
```

Esta función es usada para reiniciar el modo gráfico del vídeo al modo en uso anterior a la incialización del sistema gráfico. Esta función suele ser usada en conjunción con la función setgraphmode para cambiar entre ambos modos de texto y de gráficos.

Valor de retorno:

La función *restorecrtmode* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos y fuente para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   outtext( "Esto es una prueba para cambiar entre modo gráfico..." );
   getch();

   restorecrtmode();
   printf( "...y en modo texto.\nPulsa una tecla para volver\n" );
   getch();

   setgraphmode( gmodo );

   rectangle( 200, 100, 400, 250 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función sector Borland® C

Librería: graphics

```
void far sector(int x, int y,
   int comienzo_angulo, int final_angulo, int x_radio, int y_radio);
```

Esta función es usada para dibujar una cuña elíptica. El centro de la cuña elíptica es especificado por los argumentos **x** e **y**. El argumento **x_radio** especifica el radio horizontal y el argumento **y_radio** especifica el radio vertical de la cuña elíptica. La cuña elíptica comienza al ángulo especificado por el argumento **comienzo_angulo** y es dibujado en la dirección contraria al de las agujas del reloj hasta llegar al ángulo especificado por el argumento **final_angulo**. La cuña elíptica es dibujado con el perímetro en el color actual y rellenada con el color de relleno y la trama de relleno actuales.

Valor de retorno:

La función *sector* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setfillstyle( SOLID_FILL, 6 );

   sector( 300, 150, 45, -45, 150, 50 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función setactivepage Borland® C

Librería: graphics

```
void far setactivepage(int pagina);
```

Esta función es usada para especificar un número de página que representa una sección de memoria del vídeo donde todos los datos gráficos para mostrar son enviados. Está sección de memoria se denomina una página activa. El argumento **pagina** especifica el número de la página activa. Para usar esta función con eficacia, el adaptador de vídeo usado debe ser EGA o VGA y tener suficiente memoria para soportar múltiples páginas para gráficos. Esta función es usada con la función setvisualpage para dibujar páginas no visuales y para crear animación.

Valor de retorno:

La función *setactivepage* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int visual=1;

   printf( "Instrucciones:\nPulsa el espacio para cambiar de "
      "página, cualquier otra tecla para salir\n" );
   printf( "(Pulsa cualquier tecla para entrar en modo gráfico)\n" );
   getch();
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setactivepage( 0 );
   setfillstyle( SOLID_FILL, 6 );
   sector( 300, 150, 45, 315, 150, 50 );

   setactivepage( 1 );
   setfillstyle( SOLID_FILL, 6 );
   sector( 300, 150, 90, 270, 150, 50 );

   while( getch() == ' ' ) {
      setvisualpage( visual );
```

```
      visual = 0==visual ? 1 : 0;
   }

   closegraph();

   return 0;
}
```

Función setallpalette Borland® C

Librería: graphics

```
void far setallpalette(struct palettetype far *paleta);
```

Esta función es usada para asignar la paleta actual a la paleta definida en la estructura del tipo **palettetype** que es apuntado por el argumento ***paleta**. Todos los colores de la paleta actual son asignados a aquéllos definidos en la estructura palettetype. La síntaxis de la estructura **palettetype** es:

```
#define MAXCOLORS 15

struct palettetype {
   unsigned char size;
   signed char colors[MAXCOLORS+1];
}
```

El campo **size** indica el número de colores de la paleta actual. El campo **colors** es un array que contiene los valores numéricos que representan los colores que ofrece el dispositivo en su paleta de colores. Si la entrada de cualquier elemento del array es -1, el valor del color de ese elemento no cambiará.

Nota: Recuerda que todos los cambios hechos a la paleta tiene un efecto visual inmediato y que la función setallpalette *no debería usarse con el dispositivo IBM-8514.*

Valor de retorno:

La función *setallpalette* no retorna ningún valor; sin embargo, si los valores pasados son inválidos, entonces la función graphresult retorna grError (-11) y la paleta no es alterada.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct palettetype palette;
   int size, temp, i, y=0;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );
```

```
   getpalette( &palette );
   size = palette.size;
   for( i=0; i<size; i++ ) {
      y += 30;
      setcolor( palette.colors[i] );
      line( 20, y, 520, y );
   }

   getch();    /* Pausa */
   for( i=0; i<size/2; i++ ) {
      temp = palette.colors[i];
      palette.colors[i] = palette.colors[size-1-i];
      palette.colors[size-1-i] = temp;
   }
   setallpalette( &palette );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función setaspectratio Borland® C

Librería: graphics

```
void far setaspectratio(int x_proporcion, int y_proporcion);
```

Esta función es usada para modificar la proporción anchura-altura del modo gráfico actual. La proporción anchura-altura puede definirse como la proporción de la anchura del píxel del modo gráfico y la altura del píxel. Esta proporción es usada por el sistema gráfico para calcular círculos y arcos. Por ello, alterando la proporción anchura-altura afectará la visualización de estas funciones. La función getaspectratio puede ser usada para obtener las opciones por defecto del modo actual anteriormente a ser modificados.

Valor de retorno:

La función *setaspectratio* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int x_proporcion, y_proporcion;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   getaspectratio( &x_proporcion, &y_proporcion );

   circle( 300, 150, 50 );
   getch();    /* Pausa */

   setaspectratio( 2*x_proporcion, y_proporcion );
   circle( 300, 150, 50 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función setbkcolor Borland® C

Librería: graphics

```
void far setbkcolor(int color);
```

Esta función es usada para asignar el color de fondo al valor del color de fondo especificado por el argumento **color**.

Existen varios valores para ciertos colores de fondo.

Valor de retorno:

La función *setbkcolor* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setbkcolor( 4 );
   circle( 300, 150, 50 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función _setcursortype Borland® C

Librería: conio

```
void _setcursortype(int tipo_cursor);
```

Selecciona la apriencia del cursor entre tres tipos. El argumento **tipo_cursor** indica el tipo de cursor a seleccionar según éstos:

_NOCURSOR	Desactiva el cursor
_NORMALCURSOR	Cursor normal: el carácter de subrayado
_SOLIDCURSOR	Cursor es un cuadrado relleno

Valor de retorno:

La función *_setcursortype* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>

int main() {
   char nombre[15], si_no=' ';

   _setcursortype( _SOLIDCURSOR );
   clrscr();
   cprintf( "Ejemplo de \"_setcursortype\"\r\n\r\n" );
   cprintf( "Cambiamos el cursor a cuadrado.\r\n\r\n" );
   cprintf( "Escribe tu nombre: " );
   cscanf( "%s", &nombre );
   cprintf( "\r\n(Ahora desactivaremos el cursor)\r\n\r\n" );
   _setcursortype( _NOCURSOR );
   cprintf( "Escribiste \"%s\", ¿es esto correcto? (s/n) ", nombre );
   while( si_no != 's' && si_no != 'n' )
      si_no = getche();
   cprintf( "\r\nOpción: %s\r\n", 's'==si_no ? "SI" : "NO" );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();
   _setcursortype( _NORMALCURSOR );

   return 0;
}
```

Función setfillpattern Borland® C

Librería: graphics

```
void far setfillpattern(char far *trama, int color);
```

Esta función es usada para seleccionar una trama de relleno definido por el usuario. El argumento ***trama** apunta a una serie de ocho bytes que representa una trama de relleno de bits de 8 x 8. Cada byte representa una fila de ocho bits, donde cada bit está encendido o no (1 ó 0). Un bit de 0 indica que el píxel correspondiente será asignado el color de relleno actual. Un bit de 0 indica que el píxel correspondiente no será alterado. El argumento **color** especifica el color de relleno que será usado para la trama.

Valor de retorno:

La función *setfillpattern* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char trama1[8] = { 0x33, 0xEE, 0x33, 0xEE, 0x33, 0xEE, 0x33, 0xEE };
   char trama2[8] = { 0x0A, 0xF0, 0xF0, 0x0A, 0x0A, 0xF0, 0xF0, 0x0A };

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   bar( 50, 50, 150, 150 );

   setfillpattern( trama1, 9 );
   bar( 160, 50, 260, 150 );

   setfillpattern( trama2, 4 );
   bar( 105, 160, 205, 260 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función setfillstyle Borland® C

Librería: graphics

```
void far setfillstyle(int trama, int color);
```

Esta función es usada para seleccionar una trama predefinida y un color de relleno. El argumento **trama** especifica la trama predefinida, mientras que el argumento **color** especifica el color de relleno.

Existen trece valores ya definidos para tramas. Sin embargo, la trama USER_FILL (valor 12) no debería usarse para asignar unla trama definida por el usuario. En su lugar, se debería usar la función setfillpattern.

Valor de retorno:

La función *setfillstyle* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setfillstyle( LTSLASH_FILL, 6 );
   bar( 50, 50, 350, 300 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función setgraphbufsize Borland® C

Librería: graphics

```
unsigned far setgraphbufsize(unsigned bufer_tam);
```

Esta función es usada para cambiar el tamaño del búfer gráfico interno como es asignado por la función initgraph cuando el sistema gráfico es inicializado. El búfer gráfico es usado por varias funciones gráficos; por ello, se debería tener un mayor cuidado cuando se altera este búfer del tamaño por defecto de 4096. La función *setgraphbufsize* se debería llamar antes de llamar a la función initgraph.

Valor de retorno:

La función *setgraphbufsize* retorna el tamaño anterior del búfer gráfico interno.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int buf_inicial, buf_nuevo=10000;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   buf_inicial = setgraphbufsize( buf_nuevo );

   closegraph();

   printf( "Búfer inicial: %d\tBúfer nuevo: %d\n", buf_inicial, buf_nuevo );

   return 0;
}
```

Función setgraphmode Borland® C

Librería: graphics

```
void far setgraphmode(int modo);
```

Esta función es usada para seleccionar el modo gráfico actual pero solamente cuando el sistema gráfico haya sido inicializado con la función initgraph. El argumento **modo** define el modo a usar según el dispositivo actual. Además de seleccionar un nuevo modo, la función *setgraphmode* despeja la pantalla y reinicia todas las opciones gráficas a sus valores por defecto. Esta función suele usarse conjuntamente con restorecrtmode para cambiar entre modos gráficos y de texto.

Valor de retorno:

La función *setgraphmode* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos y fuente para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   outtext( "Esto es una prueba para cambiar entre modo gráfico..." );
   getch();

   restorecrtmode();
   printf( "...y en modo texto.\nPulsa una tecla para volver\n" );
   getch();

   setgraphmode( gmodo );

   rectangle( 200, 100, 400, 250 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función setlinestyle Borland® C

Librería: graphics

```
void far setlinestyle(int estilo,
   unsigned trama, int grosor);
```

Esta función es usada para definir las características de líneas para líneas rectas.

El argumento **estilo** especifica la trama de línea predefinida para su uso. El argumento **trama** es una trama de 16 bits que describe el estilo de línea cuando el argumento **estilo** es USERBIT_LINE, ó 4. Un bit 1 en esta trama indica que el píxel correspondiente será asignado el color actual. Un bit 0 indica que el píxel correspondiente no será alterado. El argumento **grosor** define el grosor de la línea.

Existen varios valores para los diferentes estilos y grosores de líneas rectas.

Valor de retorno:

La función *setlinestyle* no retorna ningún valor; sin embargo, si un argumento es inválido, entonces la función graphresult retorna grError (11).

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos y fuente para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setlinestyle( DOTTED_LINE, 0, THICK_WIDTH );
   line( 200, 300, 400, 50 );

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función setpalette Borland® C

Librería: graphics

```
void far setpalette(int num_paleta, int color);
```

Esta función es usada para modificar una sola entrada en la paleta actual. El argumento **num_paleta** especifica el miembro de la paleta a cambiar. El argumento **color** especifica el nuevo valor de color para el miembro de la paleta.

Existen varios valores para los colores dependiendo del dispositivo.

Nota: Recuerda que todos los cambios hechos a la paleta tiene un efecto visual inmediato y que la función setpalette *no debería usarse con el dispositivo IBM-8514.*

Valor de retorno:

La función *setpalette* no retorna ningún valor; sin embargo, si los valores pasados son inválidos, entonces la función graphresult retorna grError (-11) y la paleta no es alterada.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct palettetype palette;
   int size, temp, i, y=0;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   getpalette( &palette );
   size = palette.size;
   for( i=0; i<size; i++ ) {
      y += 30;
      setcolor( palette.colors[i] );
      line( 20, y, 520, y );
   }

   getch();    /* Pausa */
   for( i=0; i<size/2; i++ ) {
      temp = palette.colors[i];
```

```
      setpalette( i, palette.colors[size-1-i] );
      setpalette( size-1-i, temp );
   }

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función setrgbpalette Borland® C

Librería: graphics

```
void far setrgbpalette(int num_paleta,
   int rojo, int verde, int azul);
```

Esta función es para usarse con los dispositivos de IBM 8514 y VGA. El argumento **num_paleta** especifica el miembro de la paleta a ser modificado. Para la IBM 8514 (y para el modo de 256K de la VGA), el intervalo de la paleta es de 0 á 255. Para los modos de VGA, el intervalo es de 0 á 15. Los argumentos **rojo**, **verde**, y **azul** especifican la intensidad del color para el miembro de la paleta. De cada byte (de cada argumento) sólo los seis bits más signifcativos son cargados en la paleta.

Por razones de compatibilidad con otros adaptadores gráficos de IBM, el dispositivo BGI define las primeras dieciséis entradas a la paleta de la IBM 8514 a los colores por defecto de la EGA/VGA.

Nota: Recuerda que todos los cambios hechos a la paleta tiene un efecto visual inmediato y que la función setrgbpalette *no debería usarse con el dispositivo IBM-8514.*

Valor de retorno:

La función *setrgbpalette* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct palettetype palette;
   int size, i, y=0;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   getpalette( &palette );
   size = palette.size;
   for( i=0; i<size; i++ ) {
      y += 30;
      setcolor( palette.colors[i] );
      line( 20, y, 520, y );
   }

   getch();    /* Pausa */
```

```
   for( i=0; i<size; i++ )
      setrgbpalette( i, 2*i+33, 42, 63-4*i );     /* Tonos de naranja y azul */

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Función settextjustify Borland® C

Librería: graphics

```
void far settextjustify(int horizontal, int vertical);
```

Esta función es usada para especificar el método en el cual el texto es colocado en la pantalla con relación a la posición del cursor. El argumento **horizontal** define la justificación horizontal, mientras que el argumento **vertical** indica la justificación vertical.

Existen varios valores y constantes para las justificaciones.

Valor de retorno:

La función *settextjustify* no retorna ningún valor; sin embargo, si los valores pasados son inválidos, entonces la función graphresult retorna grError (-11) y la paleta no es alterada.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   settextjustify( RIGHT_TEXT, BOTTOM_TEXT );
   moveto(300, 200);
   outtext( "(RIGHT_TEXT, BOTTOM_TEXT)" );

   settextjustify( RIGHT_TEXT, TOP_TEXT );
   moveto(300, 200);
   outtext( "(RIGHT_TEXT, TOP_TEXT)" );

   settextjustify( LEFT_TEXT, BOTTOM_TEXT );
   moveto(300, 200);
   outtext( "(LEFT_TEXT, BOTTOM_TEXT)" );

   settextjustify( LEFT_TEXT, TOP_TEXT );
   moveto(300, 200);
   outtext( "(LEFT_TEXT, TOP_TEXT)" );
```

```
    setcolor( 1 );
    line( 300, 200, 300, 100 );

    setcolor( 2 );
    line( 300, 200, 300, 300 );

    setcolor( 3 );
    line( 300, 200, 100, 200 );

    setcolor( 4 );
    line( 300, 200, 500, 200 );

    getch();
    closegraph();

    return 0;
}
```

Función settextstyle Borland® C

Librería: graphics

```
void far settextstyle(int fuente,
   int orientacion, int tam_caracter);
```

Esta función es usada para especificar las características para la salida de texto con fuente. El argumento **fuente** especifica la fuente registrada a usar. La fuente ha de estar registrada para resultados predecibles; es decir, usa registerbgifont antes de usar esta función. El argumento **orientacion** especifica la orientación en que el texto ha de ser mostrado. La orientación por defecto es HORIZ_DIR. El argumento **tam_caracter** define el factor por el cual la fuente actual será multiplicada. Un valor distinto a 0 para el argumento **tam_caracter** puede ser usado con fuentes escalables o de bitmap. Sin embargo, un valor distinto a 0 para el argumento **tam_caracter**, el cual selecciona el tamaño del carácter definido por el usuario usando la función setusercharsize, solamente funciona con fuentes escalables. El argumento **tam_caracter** puede agrandar el tamaño de la fuente hasta 10 veces su tamaño normal.

Existen varios valores y constantes para las justificaciones.

Valor de retorno:

La función *settextstyle* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char mensaje[40];
   char nombre[25];

   printf( "Escribe tu nombre: " );
   scanf( "%s", nombre );
   sprintf( mensaje, "Hola %s!", nombre );

/* Esta fuente ha de ser enlazada antes de poder registrarla
   registerbgifont( sansserif_font );
*/
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );
```

```
   settextstyle( DEFAULT_FONT, 0, 2 );
   outtextxy( 100, 50, mensaje );

   settextstyle( DEFAULT_FONT, 1, 2 );
   outtextxy( 200, 125, mensaje );

   settextstyle( SANS_SERIF_FONT, 1, 3 );
   outtextxy( 400, 150, mensaje );

   getch();
   closegraph();

   return 0;
}
```

Función setusercharsize Borland® C

Librería: graphics

```
void far setusercharsize(int x_dividendo, int x_divisor,
   int y_dividendo, int y_divisor);
```

Esta función extablece las características de fuentes escalables. Para que esta función afecte el tamaño del carácter, el argumento **tam_caracter** de la función settextstyle debe ser 0. La anchura del carácter se establece con los argumentos **x_dividendo** y **x_divisor** que representan la proporción. Similarmente, los argumentos **y_dividendo** e **y_divisor** especifican la altura del carácter.

Valor de retorno:

La función *setusercharsize* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   char mensaje[40];
   char nombre[25];

   printf( "Escribe tu nombre: " );
   scanf( "%s", nombre );
   sprintf( mensaje, "Hola %s!", nombre );

/* Esta fuente ha de ser enlazada antes de poder registrarla */
   registerbgifont( sansserif_font );

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   settextstyle( SANS_SERIF_FONT, 0, 0 );
   setusercharsize( 1, 4, 1, 2 );    /* 25% de ancho; 50% de alto */
   outtextxy( 100, 50, mensaje );

   settextstyle( SANS_SERIF_FONT, 0, 1 );
   outtextxy( 100, 125, mensaje );

   settextstyle( SANS_SERIF_FONT, 1, 0 );
```

```
    setusercharsize( 1, 2, 3, 4 );    /* 50% de ancho; 75% de alto */
    outtextxy( 400, 150, mensaje );

    getch();
    closegraph();

    return 0;
}
```

Función setviewport Borland® C

Librería: graphics

```
void far setviewport(int izquierda, int superior,
   int derecha, int inferior, int recorte_banderin);
```

Esta función es usada para definir el área gráfico. La esquina superior izquierda del área gráfica está definida por los argumentos **izquierda** y **superior**. Estos argumentos corresponden a los valores *x* e *y* de la esquina superior izquierda. Similarmente, los argumentos **derecha** e **inferior** definen la esquina inferior derecha del área gráfica. El argumento **recorte_banderin** define si los datos para la salida gráfica serán recortados por el borde del área gráfico. Un valor de 0 para **recorte_banderin** indica que los datos de salida no serán recortados, mientras que un valor distinto a 0 indica que los datos serán recortados. Cuando el área gráfica es incializada, la posición del cursor será mudado a la posición (0,0) (la esquina superior izquierda). Todos los datos de salida después de que el área gráfica haya sido inicializada serán con relación a este punto. El área gráfica por defecto cubre la pantalla entera.

Valor de retorno:

La función *setviewport* no retorna ningún valor; sin embargo, si los valores pasados son inválidos, entonces la función graphresult retorna grError (-11) y el área gráfica no será alterada.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int color;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   lineto( 100, 100 );
   outtextxy( 15, 5, "Inicial" );
   getch();

   setviewport( 250, 200, 450, 300, 0 );
   setcolor( 9 );
   lineto( 100, 100 );
   outtextxy( 15, 5, "Nueva" );
   moveto( 0, 0 );
   lineto( -50, -20 ); /* Fuera del área */
```

```
    getch();

    setviewport( 250, 200, 450, 300, 1 );
    setcolor( 4 );
    moveto( 120, 40 );
    lineto( 150, -20 );  /* Fuera del área */
    outtextxy( 25, 15, "Con recorte" );

    getch();    /* Pausa */
    closegraph();

    return 0;
}
```

Función setvisualpage Borland® C

Librería: graphics

```
void far setvisualpage(int pagina);
```

Esta función es usada para establecer la página visual como es especificado por el argumento **pagina**. Una página es una sección de memoria donde se guarda la información del vídeo. Cuando se usa con un sistema (EGA o VGA) con suficiente memoria de vídeo para soportar múltiples páginas de gráficos, la función *setvisualpage* (junto con la función setactivepage) perimte al programador crear gráficos en páginas escondidas y pasar de página entre las que se han definido con información gráfica. Esto es la base para crear animación.

Valor de retorno:

La función *setvisualpage* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int visual=1;

   printf( "Instrucciones:\nPulsa el espacio para cambiar de página, cualquier otra
tecla para salir\n" );
   printf( "(Pulsa cualquier tecla para entrar en modo gráfico)\n" );
   getch();
/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setactivepage( 0 );
   setfillstyle( SOLID_FILL, 6 );
   sector( 300, 150, 45, 315, 150, 50 );

   setactivepage( 1 );
   setfillstyle( SOLID_FILL, 6 );
   sector( 300, 150, 90, 270, 150, 50 );

   while( getch() == ' ' ) {
      setvisualpage( visual );

      visual = 0==visual ? 1 : 0;
   }

   closegraph();
```

```
   return 0;
}
```

Función setwritemode Borland® C

Librería: graphics

```
void far setwritemode(int modo);
```

Esta función es usada para establecer el modo lógico de escritura para líneas rectas. El argumento **modo** especifica el modo de escritura, el cual determina la interacción entre valores de píxels existentes y los valores de píxels en la línea.

Existen dos valores para los modos de escritura.

Valor de retorno:

La función *setwritemode* no retorna ningún valor.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setfillstyle( SOLID_FILL, 1 );
   bar( 50, 50, 500, 300 );

   setwritemode( COPY_PUT );
   setcolor( 10 );
   line( 20, 60, 220, 100 );

   setwritemode( XOR_PUT );
   line( 20, 80, 220, 120 );

   getch();
   closegraph();

   return 0;
}
```

Función textattr Borland® C

Librería: conio

```
void textattr(int atributo);
```

Esta función asigna ambos colores de primer plano y de fondo en una sola llamada. (Normalmente, se asignan estos atributos mediante las funciones a textcolor y textbackground). La función *textattr* no afecta cualesquiera de los caracteres actualmente en pantalla, pero sí afecta aquéllas mostradas por funciones que usan el vídeo directamente para la salida en modo texto después de llamar a la función *textattr*.

La información de los colores está codificado en el argumento **atributo** según este diagrama:

Bits	7	6	5	4	3	2	1	0
Valores	P	f	f	f	p	p	p	p

En el argumento **atributo** de 8 bits:

- pppp es el color de primer plano de 4 bits (0-15).
- fff es el color de fondo de 3 bits (0-7).
- P es el bit de encendido de parpadeo.

Si el bit del parpadeo está activado, entonces los caracteres parpadean. Esto se puede lograr añadiendo la constante BLINK al atributo.

Si se usan las constantes simbólicas definidas en conio.h para crear los atributos de texto usando *textattr*, ten en cuenta las siguientes limitaciones para el color de fondo seleccionado:

- Sólo se pueden elegir uno de los primeros ocho colores para el fondo.
- Deberás mudar el color de fondo seleccionado 4 bits a la izquierda para que estén colocados en las posiciones correctas de los bits.

Existen varias constantes simbólicas de colores para usar.

Valor de retorno:

La función *textattr* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>

int main() {
   /* Azul de fondo y rojo claro de texto */
```

```
    int atributo=BLUE << 4 | LIGHTRED;

    cprintf( "Ejemplo de \"textattr\"\r\n\r\n" );
    textattr( atributo );
    cprintf( "Este mensaje tiene otro color de fondo y de texto.\r\n" );
    textattr( atributo + BLINK );
    cprintf( "Este mensaje está parpadeando.\r\n" );
    normvideo();
    cprintf( "Pulsa una tecla para continuar...\r\n" );
    getch();

    return 0;
}
```

Función textbackground Borland® C

Librería: conio

```
void textbackground(int color);
```

Esta función selecciona el color de fondo especificado por el argumento **color**. Esta función solamente funciona con aquellas funciones que envían datos de salida en modo texto directamente a la pantalla. El argumento **color** es un número entero entre 0 y 7; también se pueden usar constantes simbólicas definidas en conio.h en lugar de enteros. La función *textattr* no afecta cualesquiera de los caracteres actualmente en pantalla, pero sí afecta aquéllas mostradas por funciones que usan el vídeo directamente para la salida en modo texto después de llamar a la función *textattr*.

Existen varias constantes simbólicas de colores para usar.

Valor de retorno:

La función *textbackground* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>

int main() {
   cprintf( "Ejemplo de \"textbackground\" y \"textcolor\"\r\n\r\n" );
   textbackground( BLUE );
   textcolor( LIGHTRED );
   cprintf( "Este mensaje tiene otro color de fondo y de texto.\r\n" );
   textbackground( WHITE );
   cprintf( "Este mensaje tiene un color de fondo distinto.\r\n" );
   normvideo();
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Función textcolor Borland® C

Librería: conio

```
void textcolor(int color);
```

Esta función selecciona el color de texto especificado por el argumento **color**. Esta función solamente funciona con aquellas funciones que envían datos de salida en modo texto directamente a la pantalla. El argumento **color** es un número entero entre 0 y 15 y el número 128, para activar el parpadeo; también se pueden usar constantes simbólicas definidas en conio.h en lugar de enteros. La función *textcolor* no afecta cualesquiera de los caracteres actualmente en pantalla, pero sí afecta aquéllas mostradas por funciones que usan el vídeo directamente para la salida en modo texto después de llamar a la función *textcolor*.

Existen varias constantes simbólicas de colores para usar.

Valor de retorno:

La función *textcolor* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>

int main() {
   cprintf( "Ejemplo de \"textbackground\" y \"textcolor\"\r\n\r\n" );
   textbackground( BLUE );
   textcolor( LIGHTRED );
   cprintf( "Este mensaje tiene otro color de fondo y de texto.\r\n" );
   textbackground( WHITE );
   cprintf( "Este mensaje tiene un color de fondo distinto.\r\n" );
   normvideo();
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Función textheight Borland® C

Librería: graphics

```
int far textheight(char far *texto);
```

Esta función es usada para determinar la altura, en píxels, de la cadena de texto especificada por el argumento ***texto**. La altura del texto se determina usando la fuente actual y el tamaño del carácter.

Valor de retorno:

La función *textheight* retorna la altura, en píxels, del texto especificado por el argumento.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int anchura, altura;
   char mensaje[5] = "Hola";

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   anchura = textwidth( mensaje );
   altura = textheight( mensaje );

   closegraph();

   printf( "El mensaje: \"%s\" tiene de anchura: %d y altura: %d\n", mensaje,
anchura, altura );
   printf( "Pulsa una tecla para continuar...\n" );
   getch();

   return 0;
}
```

Función textwidth Borland® C

Librería: graphics

```
int far textwidth(char far *texto);
```

Esta función es usada para determinar la anchura, en píxels, de la cadena de texto especificada por el argumento ***texto**. La anchura del texto se determina usando la fuente actual y el tamaño del carácter.

Valor de retorno:

La función *textwidth* retorna la anchura, en píxels, del texto especificado por el argumento.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int anchura, altura;
   char mensaje[5] = "Hola";

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   anchura = textwidth( mensaje );
   altura = textheight( mensaje );

   closegraph();

   printf( "El mensaje: \"%s\" tiene de anchura: %d y altura: %d\n", mensaje,
anchura, altura );
   printf( "Pulsa una tecla para continuar...\n" );
   getch();

   return 0;
}
```

Función ungetch Borland® C

Librería: conio

```
int ungetch(int c);
```

Empuja el carácter especificado por el argumento **c** de vuelta a la consola, forzando el carácter empujado, **c**, a ser el siguiente carácter leído. La función *ungetch* no funciona si es llamada más de una vez antes de la siguiente lectura.

Valor de retorno:

La función *ungetch* retorna el carácter empujado, si tiene éxito; si no, entonces retorna EOF.

Ejemplo:

```
#include <conio.h>
#include <ctype.h>

int main() {
   char c;

   clrscr();
   cprintf( "Ejemplo de \"ungetch\"\r\n\r\n" );
   cprintf( "Escribe una letra: " );
   if( (c = getche()) != EOF )
      ungetch( toupper( c ) );
   cprintf( "Se ha leído: \'%c\'\r\n", getch() );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();

   return 0;
}
```

Función wherex Borland® C

Librería: conio

```
int wherex(void);
```

Obtiene la coordenada *x* de la posición del cursor actual (dentro de la ventana de texto en uso).

Valor de retorno:

La función *wherex* retorna un número entero entre 1 y el número de columnas en el modo de texto en uso.

Ejemplo:

```
#include <conio.h>

int main() {
   clrscr();
   cprintf( "Ejemplo de \"wherex\" y \"wherey\"\r\n\r\n" );
   cprintf( "La posición del cursor es: (%d,%d)\r\n", wherex(), wherey() );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();

   return 0;
}
```

Función wherey Borland® C

Librería: conio

```
int wherey(void);
```

Obtiene la coordenada *y* de la posición del cursor actual (dentro de la ventana de texto en uso).

Valor de retorno:

La función *wherey* retorna un número entero entre 1 y el número de filas en el modo de texto en uso.

Ejemplo:

```
#include <conio.h>

int main() {
   clrscr();
   cprintf( "Ejemplo de \"wherex\" y \"wherey\"\r\n\r\n" );
   cprintf( "La posición del cursor es: (%d,%d)\r\n", wherex(), wherey() );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();

   return 0;
}
```

Función window Borland® C

Librería: conio

```
void window(int izquierda, int superior, int derecha, int inferior);
```

Define una ventana de texto en pantalla especificado por los argumentos **izquierda** y **superior**, que describen la esquina superior izquierda y por los argumentos **derecha** e **inferior**, que describen la esquina inferior derecha. El tamaño mínimo de la ventana de texto es una columna por una fila. La ventana por defecto es la pantalla completa con la esquina superior izquierda siendo (1,1) y la inferior derecha siendo (C,F); donde C es el número de columnas y F el número de filas según el modo de texto en uso. La llamada a la función *window* será ignorada si alguno de los argumentos no son válidos.

Valor de retorno:

La función *window* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>

int main() {
   clrscr();
   cprintf( "Ejemplo de \"window\"\r\n\r\n" );
   cprintf( "La ventana de texto será de (10,10) á (50,20).\r\n" );
   windows( 10, 10, 50, 20 );
   cprintf( "Ahora estamos dentro de la ventana de texto.\r\n" );
   cprintf( "Pulsa una tecla para continuar..." );
   getch();

   return 0;
}
```

Indice de macros

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

-C-

Macro	Librería	Fichero de cabecera C
colores	conio	conio.h
colores	graphics	graphics.h

-D-

Macro	Librería	Fichero de cabecera C
drivers	graphics	graphics.h

-E-

Macro	Librería	Fichero de cabecera C
enlazar	graphics	graphics.h
errores	graphics	graphics.h

-F-

Macro	Librería	Fichero de cabecera C
fuentes	graphics	graphics.h

-I-

Macro	Librería	Fichero de cabecera C
inp	conio	conio.h
inportb	conio	conio.h
inpw	conio	conio.h

-L-

Macro	Librería	Fichero de cabecera C
linea	graphics	graphics.h

-M-

Macro	Librería	Fichero de cabecera C
modos	conio	conio.h
modos	graphics	graphics.h

-O-

Macro	Librería	Fichero de cabecera C
outp	conio	conio.h
outportb	conio	conio.h
outpw	conio	conio.h

-P-

Macro	Librería	Fichero de cabecera C
put_op	graphics	graphics.h

-T-

Macro	Librería	Fichero de cabecera C
trama	graphics	graphics.h

Tabla de Colores Borland® C

Librería: conio

Colores de Fondo y de Texto

Constante	Valor	Significado	De Fondo o de Texto
BLACK	0	Negro	Ambos
BLUE	1	Azul	Ambos
GREEN	2	Verde	Ambos
CYAN	3	Cían	Ambos
RED	4	Rojo	Ambos
MAGENTA	5	Magenta	Ambos
BROWN	6	Marrón	Ambos
LIGHTGRAY	7	Gris Claro	Ambos
DARKGRAY	8	Gris Oscuro	Sólo para texto
LIGHTBLUE	9	Azul Claro	Sólo para texto
LIGHTGREEN	10	Verde Claro	Sólo para texto
LIGHTCYAN	11	Cían Claro	Sólo para texto
LIGHTRED	12	Rojo Claro	Sólo para texto
LIGHTMAGENTA	13	Magenta Claro	Sólo para texto
YELLOW	14	Amarillo	Sólo para texto
WHITE	15	Blanco	Sólo para texto
BLINK	128	Parpadeo	Sólo para texto

Nota: Algunos monitores no reconocen la señal de intensidad usada para crear los ocho colores "claros" (8-15). En tales monitores, los colores claros son mostrados como sus equivalentes "oscuros" (0-7). Además, ciertos sistemas que no muestran colores pueden interpretar estos números como tonos de un color, tramas especiales, o atributos especiales (como puede ser subrayado, en negrita, itálico, etc.). Exactamente lo que se muestre depende del sistema gráfico que se tiene.

Tabla de Colores Borland® C

Librería: graphics

Colores de Fondo

Constante	Valor	Significado
BLACK	0	Negro
BLUE	1	Azul
GREEN	2	Verde
CYAN	3	Cían
RED	4	Rojo
MAGENTA	5	Magenta
BROWN	6	Marrón
LIGHTGRAY	7	Gris Claro
DARKGRAY	8	Gris Oscuro
LIGHTBLUE	9	Azul Claro
LIGHTGREEN	10	Verde Claro
LIGHTCYAN	11	Cían Claro
LIGHTRED	12	Rojo Claro
LIGHTMAGENTA	13	Magenta Claro
YELLOW	14	Amarillo
WHITE	15	Blanco

Colores para Modos de 16 Colores

Constante	Valor	Significado
BLACK	0	Negro
BLUE	1	Azul
GREEN	2	Verde
CYAN	3	Cían
RED	4	Rojo
MAGENTA	5	Magenta
BROWN	6	Marrón
LIGHTGRAY	7	Gris Claro
DARKGRAY	8	Gris Oscuro
LIGHTBLUE	9	Azul Claro
LIGHTGREEN	10	Verde Claro
LIGHTCYAN	11	Cían Claro
LIGHTRED	12	Rojo Claro
LIGHTMAGENTA	13	Magenta Claro
YELLOW	14	Amarillo
WHITE	15	Blanco

Colores para modos de CGA

Número de Paleta	Color 1	Significado	Color 2	Significado	Color 3	Significado

0	CGA_LIGHTGREEN	Verde Claro	CGA_LIGHTRED	Rojo Claro	CGA_YELLOW	Amarillo
1	CGA_LIGHTCYAN	Cían Claro	CGA_LIGHTMAGENTA	Magenta Claro	CGA_WHITE	Blanco
2	CGA_GREEN	Verde	CGA_RED	Rojo	CGA_BROWN	Marrón
3	CGA_CYAN	Cían	CGA_MAGENTA	Magenta	CGA_LIGHTGRAY	Gris Claro
Valor asignado:	1		2		3	

Nota: Color 0 se reserva para el color de fondo y se asigna con lo función setbkcolor*, pero los demás colores son fijos. Estas constantes se usan con* setcolor*.*

Colores para la paleta

Constante (CGA)	Valor	Constante (EGA/VGA)	Valor
BLACK	0	EGA_BLACK	0
BLUE	1	EGA_BLUE	1
GREEN	2	EGA_GREEEN	2
CYAN	3	EGA_CYAN	3
RED	4	EGA_RED	4
MAGENTA	5	EGA_MAGENTA	5
BROWN	6	EGA_LIGHTGRAY	7
LIGHTGRAY	7	EGA_BROWN	20
DARKGRAY	8	EGA_DARKGRAY	56
LIGHTBLUE	9	EGA_LIGHTBLUE	57
LIGHTGREEN	10	EGA_LIGHTGREEN	58
LIGHTCYAN	11	EGA_LIGHTCYAN	59
LIGHTRED	12	EGA_LIGHTRED	60
LIGHTMAGENTA	13	EGA_LIGHTMAGENTA	61
YELLOW	14	EGA_YELLOW	62
WHITE	15	EGA_WHITE	63

Nota: Estas constantes se usan con las funciones setpalette *y* setallpalette*.*

Tabla de Dispositivos Borland® C

Librería: graphics

Dispositivos Gráficos

Dispositivo/Constante	Valor
DETECT	0
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
IBM8514	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

Enlazar Dispositivos y Fuentes Borland®

Librería: graphics

Para enlazar los ficheros de dispositivos gráficos y de fuentes escalables a programas que usan el BGI, se ha de convertir estos ficheros a ficheros objetos (.OBJ). Los ficheros objetos pueden ser enlazados (o ligados) al fichero ejecutable. Este proceso es ventajosa ya que permite que el programa acceda a los dispositivos gráficos y/o fuentes escalables directamente, sin tener que cargarlos desde el disco mientras el programa se esté ejecutando. La desventaja de este proceso es el tamaño del fichero ejecutable que lógicamente incrementará debido a que incorpora los ficheros enlazados.

Esta conversión se hace mediante el programa BGIOBJ.EXE. La síntaxis simplificada de este programa es la siguiente:

```
BGIOBJ <nombre>
```

donde <nombre> es el nombre del fichero: fuente escalable o dispositivo gráfico. El programa producirá otro fichero del mismo nombre con la extensión .OBJ.

Por ejemplo:

```
BGIOBJ EGAVGA.BGI
```

Esto convertirá el fichero EGAVGA.BGI a EGAVGA.OBJ.

Ahora hay que enlazar (o ligar) este ficehro objeto con la librería gráfica. El modo de hacer esto es usando el programa TLIB.EXE. La síntaxis simplificada es la siguiente:

```
TLIB <nombre_librería> +<nombre_objeto1> [+<nombre_objeto2>...]
```

donde <nombre_librería> es la librería al cual queremos añadir los ficheros objetos cuyos nombres son <nombre_objeto1>, <nombre_objeto2>, etc.. Como sólo se puede añadir ficheros objetos a la librería, por eso no hace falta incluir la extensión .OBJ.

Por ejemplo:

```
TLIB GRAPHICS +EGAVGA +GOTH +BOLD +EURO
```

Esto añadirá los ficheros objetos EGAVGA.OBJ, GOTH.OBJ, BOLD.OBJ, y EURO.OBJ a la librería GRAPHICS.LIB.

Para poder seleccionar estos dispositivos y/o fuentes, se ha de registrarlos para que sean enlazados en el programa. La forma de hacer esto es a través de las funciones registerbgidriver y registerbgifont.en el programa - antes de llamar a initgraph. Esto informa al sistema gráfico de que tales ficheros están presentes y asegura que están enlazados cuando el enlazador (o linker) cree el fichero ejecutable. Las rutinas de registro aceptan cada uno un parámetro; un nombre simbólico definido en graphics.h. Cada rutina de registro retorna un valor no negativo si el dispositivo o fuente se registra con éxito.

He aquí la lista de los dispositivos y fuentes que se usan con la funciones registerbgidriver y registerbgifont:

Dispositivos/Fuentes y sus Nombres Simbólicos

Dispositivo (.BGI)	**Nombre simbólico (registerbgidriver)**	**Fuente (.CHR)**	**Nombre simbólico (registerbgifont)**
CGA	CGA_driver	TRIP	triplex_font
EGAVGA	EGAVGA_driver	LITT	small_font
HERC	Herc_driver	SANS	sansserif_font
ATT	ATT_driver	GOTH	gothic_font
PC3270	PC3270_driver		
IBM8514	IBM8514_driver		

Tabla de Errores Borland® C

Librería: graphics

Códigos de Errores

Constante	Código	Significado
grOk	0	Ningún error
grNoInitGraph	-1	Gráficos no iniciados
grNotDetected	-2	Ningún adaptador gráfico detectado
grFileNotFound	-3	Fichero de dispositivo no encontrado
grInvalidDriver	-4	Fichero de dispositivo no válido
grNoLoadMem	-5	No hay memoria para cargar dispositivo
grNoScanMem	-6	No hay memoria para rellenar
grNoFloodMem	-7	No hay memoria para usar floodfill
grFontNotFound	-8	Fichero de fuente no encontrado
grNoFontMem	-9	No hay memoria para cargar la fuente
grInvalidMode	-10	Modo gráfico no válido
grError	-11	Error gráfico
grIOerror	-12	Error gráfico de Entrada/Salida
grInvalidFont	-13	Fichero de fuente no válido
grInvalidFontNum	-14	Número de fuente no válido
grInvalidDeviceNum	-15	Número de dispositivo no válido
grInvalidVersion	-18	Número de versión no válido

Tabla de Fuentes Borland® C

Librería: graphics

Fuentes para Texto

Constante	Valor	Significado
DEFAULT_FONT	0	Fuente bitmap de 8x8
TRIPLEX_FONT	1	Fuente escalable de tipo triple
SMALL_FONT	2	Fuente escalable pequeña
SANS_SERIF_FONT	3	Fuente escalable de tipo sans serif
GOTHIC_FONT	4	Fuente escalable de tipo gótico
SCRIPT_FONT	5	Fuente escalable de tipo manuscrito
SIMPLEX_FONT	6	Fuente escalable de tipo manuscrito simple
TRIPLEX_SCR_FONT	7	Fuente escalable de tipo manuscrito triple
COMPLEX_FONT	8	Fuente escalable de tipo complejo
EUROPEAN_FONT	9	Fuente escalable de tipo europeo
BOLD_FONT	10	Fuente escalable en negrita

Orientaciones para Texto

Constante	Valor	Significado
HORIZ_DIR	0	Texto horizontal
VERT_DIR	1	Texto vertical

Justificación de Texto en la Horizontal

Constante	Valor	Significado
LEFT_TEXT	0	Justificar a la izquierda
CENTER_TEXT	1	Centrar el texto
RIGHT_TEXT	2	Justificar a la derecha

Justificación de Texto en la Vertical

Constante	Valor	Significado
BOTTOM_TEXT	0	Justificar debajo
CENTER_TEXT	1	Centrar el texto
TOP_TEXT	2	Justificar arriba

Macro inp Borland® C

Librería: conio

```
int inp(unsigned id_puerto);
```

Lee 1 byte desde el puerto indicado por el argumento **id_puerto**. Si la macro *inp* es llamada cuando conio.h haya sido incluida, será tratada como una macro que expande el código "en línea" (inline). Si no se incluye conio.h, o si se incluye conio.h pero desactivas la definición de la macro *inp*, entonces se instituye la función *inp*.

Valor de retorno:

La macro *inp* retorna el valor leído desde el puerto apuntado por el argumento **id_puerto**.

Ejemplo:

```
#include <conio.h>

int main() {
   int valor;
   unsigned id_puerto=0;    /* Puerto de serie 0 */

   valor = inp( id_puerto );
   clrscr();
   cprintf( "Ejemplo de \"inp\"\r\n\r\n" );
   cprintf( "Leemos 1 byte desde el puerto %d: %d.\r\n", id_puerto, valor );
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Macro inportb Borland® C

Librería: conio

```
unsigned char inportb(int id_puerto);
```

Lee 1 byte desde el puerto indicado por el argumento **id_puerto**. Si la macro *inportb* es llamada cuando conio.h haya sido incluida, será tratada como una macro que expande el código "en línea" (inline). Si no se incluye conio.h, o si se incluye conio.h pero desactivas la definición de la macro *inportb*, entonces se instituye la función *inportb*.

Valor de retorno:

La macro *inportb* retorna el valor leído desde el puerto apuntado por el argumento **id_puerto**.

Ejemplo:

```
#include <conio.h>

int main() {
   unsigned valor;
   int id_puerto=0;    /* Puerto de serie 0 */

   valor = inportb( id_puerto );
   clrscr();
   cprintf( "Ejemplo de \"inportb\"\r\n\r\n" );
   cprintf( "Leemos 1 byte desde el puerto %d: 0x%X.\r\n", id_puerto, valor );
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Macro inpw Borland® C

Librería: conio

```
int inpw(unsigned id_puerto);
```

Lee 1 byte de la parte baja de 1 palabra (word) de 16 bits desde el puerto de entrada indicado por el argumento **id_puerto**; lee el byte alto desde **id_puerto**+1. Si la macro *inpw* es llamada cuando conio.h haya sido incluida, será tratada como una macro que expande el código "en línea" (inline). Si no se incluye conio.h, o si se incluye conio.h pero desactivas la definición de la macro *inpw*, entonces se instituye la función *inpw*.

Valor de retorno:

La macro *inpw* retorna el valor leído de una palabra (word) de tamaño desde el puerto apuntado por el argumento **id_puerto** e **id_puerto**+1.

Ejemplo:

```
#include <conio.h>

int main() {
   int valor;
   unsigned id_puerto=0;    /* Puerto de serie 0 */

   valor = inpw( id_puerto );
   clrscr();
   cprintf( "Ejemplo de \"inpw\"\r\n\r\n" );
   cprintf( "Leemos 1 word desde el puerto %d: 0x%X.\r\n", id_puerto, valor );
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Tabla de Líneas Borland® C

Librería: graphics

Estilos de Líneas

Constante	Valor	Significado
SOLID_LINE	0	Línea continua
DOTTED_LINE	1	Línea hecha con puntos
CENTER_LINE	2	Línea centrada
DASHED_LINE	3	Línea discontinua
USERBIT_LINE	4	Línea definida por el usuario

Grosores para Líneas

NORM_THICK	1	Grosor es de 1 píxel
THICK_WIDTH	3	Grosor es de 3 píxels

Modos de Escritura

Constantes	Valor	Significado
COPY_PUT	0	Píxels de la línea sobrescriben los píxels existentes
XOR_PUT	1	Píxels de la pantalla son el resultado de la operación OR de los píxels existentes y los de la línea

Tabla de Modos Borland® C

Librería: conio

Modos de Texto (Compatibles con MS-DOS®)

Constante	Valor	Significado
LASTMODE	-1	Selecciona el modo anterior
BW40	0	Blanco y negro, con 50 columnas
C40	1	Color, con 40 columnas
BW80	2	Blanco y negro, con 80 columnas
C80	3	Color, con 80 columnas
MONO	7	Monocromo, con 80 columnas
C4350	64	En EGA: 43 líneas y en VGA: 50 líneas

Los anteriores modos de texto son compatibles con el entorno de MS-DOS®, independientemente del modo gráfico.

Los siguientes modos de texto pueden o no ser válidas, según el sistema gráfico.

Modos de Texto Nuevos

Constante	Valor	Significado
C40X14	8	Color, con 40 columnas y 14 líneas
C40X21	9	Color, con 40 columnas y 21 líneas
C40X28	10	Color, con 40 columnas y 28 líneas
C40X43	11	Color, con 40 columnas y 43 líneas
C40X50	12	Color, con 40 columnas y 50 líneas
C40X60	13	Color, con 40 columnas y 60 líneas
C80X14	14	Color, con 80 columnas y 14 líneas
C80X21	15	Color, con 80 columnas y 21 líneas
C80X28	16	Color, con 80 columnas y 28 líneas
C80X43	17	Color, con 80 columnas y 43 líneas
C80X50	18	Color, con 80 columnas y 50 líneas
C80X60	19	Color, con 80 columnas y 60 líneas
BW40X14	20	Blanco y negro, con 40 columnas y 14 líneas
BW40X21	21	Blanco y negro, con 40 columnas y 21 líneas

BW40X28	22	Blanco y negro, con 40 columnas y 28 líneas
BW40X43	23	Blanco y negro, con 40 columnas y 43 líneas
BW40X50	24	Blanco y negro, con 40 columnas y 50 líneas
BW40X60	25	Blanco y negro, con 40 columnas y 60 líneas
BW80X14	26	Blanco y negro, con 80 columnas y 14 líneas
BW80X21	27	Blanco y negro, con 80 columnas y 21 líneas
BW80X28	28	Blanco y negro, con 80 columnas y 28 líneas
BW80X43	29	Blanco y negro, con 80 columnas y 43 líneas
BW80X50	30	Blanco y negro, con 80 columnas y 50 líneas
BW80X60	31	Blanco y negro, con 80 columnas y 60 líneas
MONO14	32	Monocromo, con 14 líneas
MONO21	33	Monocromo, con 21 líneas
MONO28	34	Monocromo, con 28 líneas
MONO43	35	Monocromo, con 43 líneas
MONO50	36	Monocromo, con 50 líneas
MONO60	37	Monocromo, con 60 líneas
_ORIGMODE	65	Modo Original al comienzo del programa

Nota: Los modos nuevos monocromos no son válidos para el modo gráfico VGA.

Tabla de Modos Borland® C

Librería: graphics

Modos Gráficos

Dispositivo	Modo/Constante	Código	Resolución	Paleta	Páginas
CGA	CGAC0	0	320x200	4 colores	1
	CGAC1	1	320x200	4 colores	1
	CGAC2	2	320x200	4 colores	1
	CGAC3	3	320x200	4 colores	1
	CGAHI	4	640x200	2 colores	1
MCGA	MCGAC0	0	320x200	4 colores	1
	MCGAC1	1	320x200	4 colores	1
	MCGAC2	2	320x200	4 colores	1
	MCGAC3	3	320x200	4 colores	1
	MCGAMED	4	640x200	2 colores	1
	MCGAHI	5	640x480	2 colores	1
EGA	EGALO	0	640x200	16 colores	4
	EGAHI	1	640x350	16 colores	2
EGA64	EGA64LO	0	640x200	16 colores	1
	EGA64HI	1	640x350	4 colores	1
EGAMONO	EGAMONOHI	3	640x200	2 colores	1* / 2**
VGA	VGALO	0	640x200	16 colores	2
	VGAMED	1	640x350	16 colores	2
	VGAHI	2	640x480	16 colores	1
ATT400	ATT400C0	0	320x200	4 colores	1
	ATT400C1	1	320x200	4 colores	1
	ATT400C2	2	320x200	4 colores	1
	ATT400C3	3	320x200	4 colores	1
	ATT400MED	4	640x200	2 colores	1
	ATT400HI	5	640x400	2 colores	1
HERC	HERCMONOHI	0	720x348	2 colores	2
PC3270	PC3270HI	0	720x350	2 colores	1
IBM8514	IBM8514LO	0	640x480	256 colores	
	IBM8514HI	1	1024x768	256 colores	

```
* Si la tarjeta es de 64K
```

```
** Si la tarjeta es de 256K
```

Macro outp Borland® C

Librería: conio

```
int outp(unsigned id_puerto, int valor);
```

Escribe el último byte del argumento **valor** al puerto indicado por el argumento **id_puerto**. Si la macro *outp* es llamada cuando conio.h haya sido incluida, será tratada como una macro que expande el código "en línea" (inline). Si no se incluye conio.h, o si se incluye conio.h pero desactivas la definición de la macro *outp*, entonces se instituye la función *outp*.

Valor de retorno:

La macro *outp* retorna el valor escrito al puerto apuntado por el argumento **id_puerto**.

Ejemplo:

```
#include <conio.h>

int main() {
   int valor=0xFF;
   unsigned id_puerto=0;    /* Puerto de serie 0 */

   outp( id_puerto, valor );
   clrscr();
   cprintf( "Ejemplo de \"outp\"\r\n\r\n" );
   cprintf( "Escribimos 1 byte al puerto %d: %d.\r\n", id_puerto, valor );
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Macro outportb Borland® C

Librería: conio

```
void outportb(int id_puerto, unsigned char valor);
```

Escribe 1 byte al puerto de salida indicado por el argumento **id_puerto**. Si la macro *outportb* es llamada cuando conio.h haya sido incluida, será tratada como una macro que expande el código "en línea" (inline). Si no se incluye conio.h, o si se incluye conio.h pero desactivas la definición de la macro *outportb*, entonces se instituye la función *outportb*.

Valor de retorno:

La función *outportb* no retorna ningún valor.

Ejemplo:

```
#include <conio.h>

int main() {
   int valor=0xFA, id_puerto=0;    /* Puerto de serie 0 */

   outportb( id_puerto, valor );
   clrscr();
   cprintf( "Ejemplo de \"outport\"\r\n\r\n" );
   cprintf( "Escribimos 1 word (2 bytes) al puerto %d: 0x%X.\r\n", id_puerto, valor
);
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Macro outpw Borland® C

Librería: conio

```
void outpw(unsigned id_puerto, int valor);
```

Escribe el último byte de 1 palabra (word) de 16 bits al puerto de entrada indicado por el argumento **id_puerto**; escribe el primer byte desde **id_puerto**+1, usando una sola instrucción de 16 bits *OUT*. Si la macro *outpw* es llamada cuando conio.h haya sido incluida, será tratada como una macro que expande el código "en línea" (inline). Si no se incluye conio.h, o si se incluye conio.h pero desactivas la definición de la macro *outpw*, entonces se instituye la función *outpw*.

Valor de retorno:

La macro *outpw* retorna el valor escrito de una palabra (word) de tamaño al puerto apuntado por el argumento **id_puerto** e **id_puerto**+1.

Ejemplo:

```
#include <conio.h>

int main() {
   int valor=0xFFAA;
   unsigned id_puerto=0;    /* Puerto de serie 0 */

   outpw( id_puerto, valor );
   clrscr();
   cprintf( "Ejemplo de \"outpw\"\r\n\r\n" );
   cprintf( "Escribimos 1 word al puerto %d: 0x%X.\r\n", id_puerto, valor );
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Tabla de Operaciones con Putimage Borland® C

Librería: graphics

Operaciones con putimage

Constante	Valor	Significado
COPY_PUT	0	Sobrescribir los píxels existentes
XOR_PUT	1	Operación OR Exclusivo con los píxels
OR_PUT	2	Operación OR Inclusivo con los píxels
AND_PUT	3	Operación AND con los píxels
NOT_PUT	4	Invertir la imagen

Nota: Estas operaciones se usan exclusivamente con la función putimage*.*

Tabla de Tramas Borland® C

Librería: graphics

Tramas predefinidas

Constante	Valor	Significado
EMPTY_FILL	0	Rellena con el color de fondo
SOLID_FILL	1	Rellena enteramente
LINE_FILL	2	Rellena con líneas horizontales: ---
LTSLASH_FILL	3	Rellena con rayas finas: ///
SLASH_FILL	4	Rellena con rayas gruesas: ///
BKSLASH_FILL	5	Rellena con rayas inversas y finas: \\\
LTBKSLASH_FILL	6	Rellena con rayas inversas y gruesas: \\\
HATCH_FILL	7	Rellena con líneas cruzadas cuadriculadamente: +++
XHATCH_FILL	8	Rellena con líneas cruzadas diagonalmente: XXXX
INTERLEAVE_FILL	9	Rellena con líneas entrelazadas
WIDE_DOT_FILL	10	Rellena con lunares bastante distanciados
CLOSE_DOT_FILL	11	Rellena con lunares poco distanciados
USER_FILL	12	Rellena con la trama definida por el usuario

Nota: Todos los tipos de tramas menos EMPTY_FILL usan el color de relleno seleccionado; EMPTY_FILL usa el color de fondo para rellenar.

Indice de tipos y estructuras

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

-A-

Estructura	Librería	Fichero de cabecera C
arccoordstype	graphics	graphics.h

-F-

Estructura	Librería	Fichero de cabecera C
fillsettingstype	graphics	graphics.h

-L-

Estructura	Librería	Fichero de cabecera C
linesettingstype	graphics	graphics.h

-P-

Estructura	Librería	Fichero de cabecera C
palettetype	graphics	graphics.h

-T-

Estructura	Librería	Fichero de cabecera C
textsettingstype	graphics	graphics.h
text_info	conio	conio.h

-V-

Estructura	Librería	Fichero de cabecera C
viewporttype	graphics	graphics.h

Estructura arccoordstype Borland® C

Librería: graphics

```
struct arccoordstype {
   int x, y;
   int xstart, ystart;
   int xend, yend;
};
```

Los miembros **x** e **y** definen el centro del arco. Los miembros **xstart** e **ystart** definen las coordenadas *x* e *y* del punto de comienzo del arco. Similarmente, los miembros **xend** e **yend** definen las coordenadas *x* e *y* del punto de final del arco.

Esta estructura se usa como parámetro en la función getarccoords, que se usa para recoger las coordenadas del centro, y los puntos del comienzo y final de la última llamada con éxito a la función arc.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   int radio;
   struct arccoordstype info_arco;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   for( radio=25; radio<=100; radio+=25 ) {
      arc( 300, 150, 45, 315, radio );
      getarccoords( &info_arco );
      moveto( info_arco.xstart, info_arco.ystart );
      lineto( info_arco.xend, info_arco.yend );
   }

   getch();    /* Pausa */
   closegraph();

   return 0;
}
```

Estructura fillsettingstype Borland® C

Librería: graphics

```
struct fillsettingstype {
   int pattern;
   int color;
};
```

Esta estructura se usa para obtener la información de tramas de relleno, mediante getfillsettings.

El campo **pattern** es la trama y el campo **color** es el color de relleno de la trama.

Existen trece valores ya definidos para tramas.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct fillsettingstype info;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   getfillsettings( &info );
   bar( 50, 50, 350, 300 );

   getch();    /* Pausa */
   closegraph();

   printf( "Trama de relleno: %d\tColor de relleno: %d\n",
      info.pattern, info.color );

   return 0;
}
```

Estructura linesettingstype Borland® C

Librería: graphics

```
struct linesettingstype {
   int linestyle;
   unsigned upattern;
   int thickness;
}
```

Esta estructura se usa para obtener la información actual para las líneas mediante la función getlinesettings.

El campo **linestyle** es el estilo de la línea recta. El campo **upattern** es la trama de la línea del usuario solamente cuando el campo **linestyle** es igual a **USERBIT_LINE**, ó 4. Cuando esto sea el caso, el miembro **upattern** contiene una trama de línea definido por el usuario de 16 bits. Un bit 1 en esta trama indica que el píxel correspondiente será asignado el color actual. Un bit 0 indica que el píxel correspondiente no será alterado. El campo **thickness** es el grosor de la línea.

Existen varios valores para los diferentes estilos y grosores de líneas rectas.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct linesettingstype info;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   setlinestyle( DOTTED_LINE, 0xFF33, THICK_WIDTH );
   circle( 350, 250, 50 );

   getlinesettings( &info );

   getch();    /* Pausa */
   closegraph();

   printf( "Líneas rectas.\nEstilo: %d\tTrama: %X\tGrosor: %d\n",
      info.linestyle, info.upattern, info.thickness  );

   return 0;
```

```
}
```

Estructura palettetype Borland® C

Librería: graphics

```
#define MAXCOLORS 15

struct palettetype {
   unsigned char size;
   signed char colors[MAXCOLORS+1];
}
```

Esta estructura se usa para obtener una los datos que definen la paleta según cada dispositivo.

El campo **size** indica el tamaño de la paleta. El campo **colors** contiene los valores numéricos que representan los colores que ofrece el dispositivo en su paleta de colores.

Ejemplo:

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h&t;

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct palettetype *palette = NULL;
   int i;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   palette = getpalettetype();
   circle( 300, 150, 50 );

   getch();    /* Pausa */
   closegraph();

   printf( "Paleta\n\nTamaño: %d\nColores: %d",
      palette->size, palette->colors[0] );
   for( i=1; i<palette->size; i++ )
      printf( ", %d", palette->colors[i] );
   printf( "\n" );

   return 0;
}
```

Estructura textsettingstype Borland® C

Librería: graphics

```
struct textsettingstype {
   int font;
   int direction;
   int charsize;
   int horiz;
   int vert;
};
```

Esta estructura se usa para obtener información acerca de la fuente gráfica actual mediante la función getextsettings.

Esta estructura contiene información de la fuente actual en uso, la orientación del texto, el tamaño del carácter, y la justificación horizontal y vertical.

Existen varios valores para describir el tipo, la orientación, y justificación de fuentes.

Ejemplo:

```
#include <graphics.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct textsettingstype info;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   gettextsettings( &info );

   closegraph();

   printf( "Texto\n\nFuente: %d\tSentido: %d\tTamaño: %d\n"
           "Justificación:\nHorizontal: %d, Vertical: %d\n",
           info.font, info.direction, info.charsize, info.horiz, info.vert);

   return 0;
}
```

Estructura text_info Borland® C

Librería: conio

```
struct text_info {
   unsigned char winleft;        /* Coordenada izquierda de la ventana */
   unsigned char wintop;         /* Coordenada superior de la ventana */
   unsigned char winright;       /* Coordenada derecha de la ventana */
   unsigned char winbottom;      /* Coordenada inferior de la ventana */
   unsigned char attribute;      /* Atributo de texto */
   unsigned char normattr;       /* Atributo normal*/
   unsigned char currmode;       /* Modo en Uso: BW40, BW80, C40, C80, ó C4350 */
   unsigned char screenheight;   /* Altura de la pantalla de texto */
   unsigned char screenwidth;    /* Anchura de la pantalla de texto */
   unsigned char curx;           /* Coordenada X de la ventana en uso */
   unsigned char cury;           /* Coordenada Y de la ventana en uso */
};
```

Esta estructuras se usa como parámetro en la función gettextinfo para obtener la información sobre la ventana de texto actual.

Ejemplo:

```
#include <conio.h>

int main() {
   struct text_info *ti;

   gettextinfo( ti );
   clrscr();
   cprintf( "Ejemplo de \"gettextinfo\"\r\n\r\n" );
   cprintf( "Dimensiones de la ventana: " );
   cprintf( "(%d,%d) á (%d,%d)\r\n", ti->winleft, ti->wintop,
      ti->winright, ti->winbottom );
   cprintf( "Atributo: %d  Normal: %d\r\n", ti->attribute, ti->normattr );
   cprintf( "Modo en uso: %d\r\n", ti->currmode );
   cprintf( "Dimensiones de la pantalla: %d x %d\r\n",
      ti->screenwidth, ti->screenheight );
   cprintf( "Coordenadas de la ventana: (%d,%d)\r\n", ti->curx, ti->cury );
   cprintf( "Pulsa una tecla para continuar...\r\n" );
   getch();

   return 0;
}
```

Estructura viewporttype Borland® C

Librería: graphics

```
struct viewporttype {
   int left, top;
   int right, bottom;
   int clip;
};
```

Esta estructura se usa para obtener información acerca del área gráfica actual mediante la función getviewsettings.

Esta estructura contiene información acerca de las esquinas superior izquierda e inferior derecha, también como el banderín de recorte del área gráfica.

Ejemplo:

```
#include <graphics.h>
#include <stdio.h>

int main() {
   int gdriver = EGA;
   int gmodo = EGAHI;
   struct viewporttype info;

/* Si has registrado los dispositivos para que formen parte de graphics.lib
** entonces usa estas sentencias:

   registerbgidriver( EGAVGA_driver );
   initgraph( &gdriver, &gmodo, "" );
*/

/* Si no, entonces has de "decir" dónde se encuentra el dispositivo gráfico */

   initgraph( &gdriver, &gmodo, "C:\\BC5\\BGI" );

   getviewsettings( &info );

   closegraph();

   printf( "Pantalla\n\nIzquierda: %d\tSuperior: %d\tDerecha: %d\t"
           "Inferior: %d\tBanderín: %d\n",
           info.left, info.top, info.right, info.bottom, info.clip );

   return 0;
}
```