Desarrollar una aplicación con gráficas en DevC++ usando WinBGIm

Introducción

La librería WinBGIm ha sido creada originalmente por Konstantin Knizhnik's en el proyecto winbgi shareware y posteriormente modificada por Mark Richardson y Michael Main; posterior a esto, ya que la librería no tuvo más soporte desde el año 2004, Oswaldo Romero de la Universidad Distrital en Colombia hizo modificaciones menores y añadió unas funciones para la manipulación de imágenes BMP. Esta librería es una versión que emula la creada por la compañía Borland BGI (Borland Graphics Interface) la cual tiene la capacidad para crear gráficos en pantalla en aplicaciones de 16 bis sobre MS-DOS® y Windows®.

Esta librería facilita funciones que permiten escribir en modo gráfico en la pantalla (en realidad en una ventana de Windows) punto a punto, figuras geométricas, copiar y modificar el contenido de un trozo de la pantalla y manejar el ratón.

La ventaja de esta librería es que nos permite crear aplicaciones gráficas con compiladores GNU tales como el ambiente de desarrollo Dev C++ de la misma manera como se crean en el compilador comercial Borland C++, siendo muy utilizada en su tiempo por la facilidad de su uso.

¿Cómo se crea una aplicación?

En Dev-C++, ejecutamos la aplicación y creamos un nuevo proyecto de Consola, como se muestra en la figura 1.

Como nombre de proyecto, escoja una ubicación en donde se creará una carpeta con el mismo nombre; en esta carpeta se guardarán los archivos de código y por supuesto los archivos correspondientes a la librería.

El siguiente paso es copiar los archivos de la librería para poderlos colocar en el proyecto. Estos archivos son (ver figura 2):

- graph.h : Que es el encabezado (header) donde se definen los prototipos de las funciones de la librería.
- graph.cpp : Corresponde a la implementación de las funciones.

Luego debemos entrar a las opciones del proyecto (ver figura 3). Se puede hacer de 2 formas:

- presionando la combinación de teclas CTRL+H.
- ir con el puntero del ratón a la sección del Proyecto (Tabulador), y con click derecho seleccionar en el menú que aparece “Opciones del proyecto”.

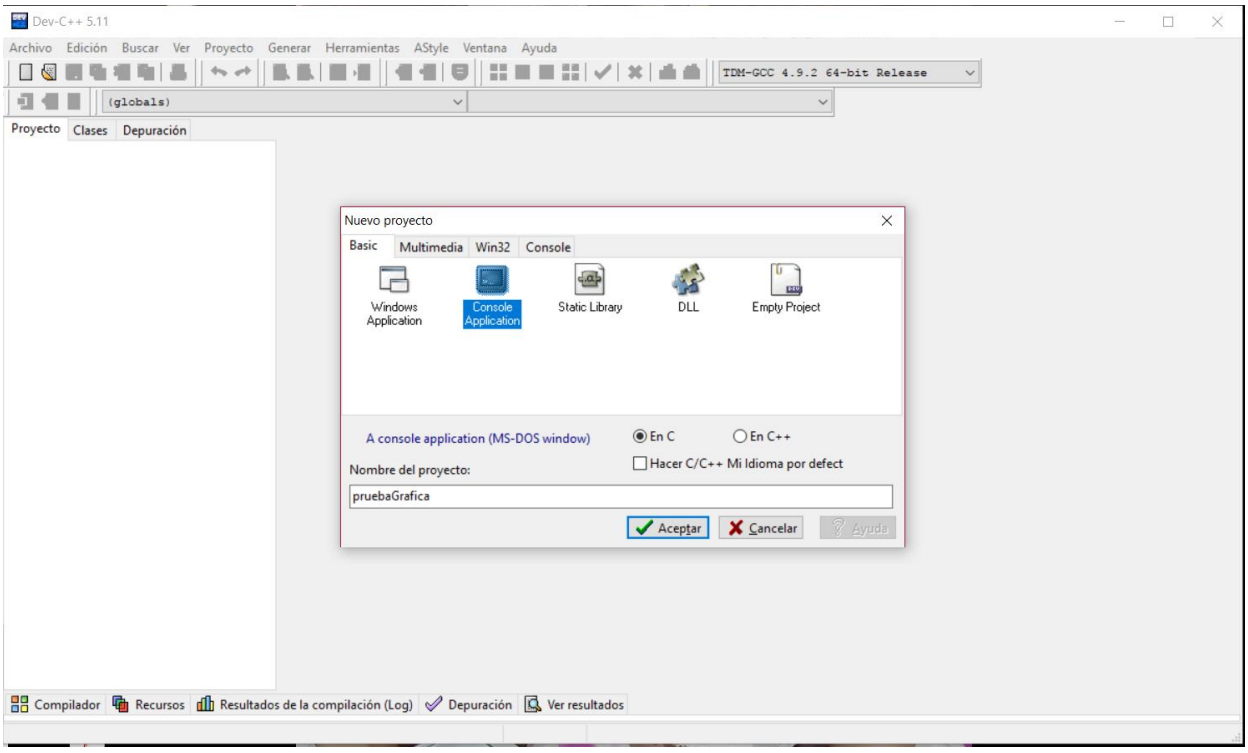


Figura 1. Nuevo proyecto.

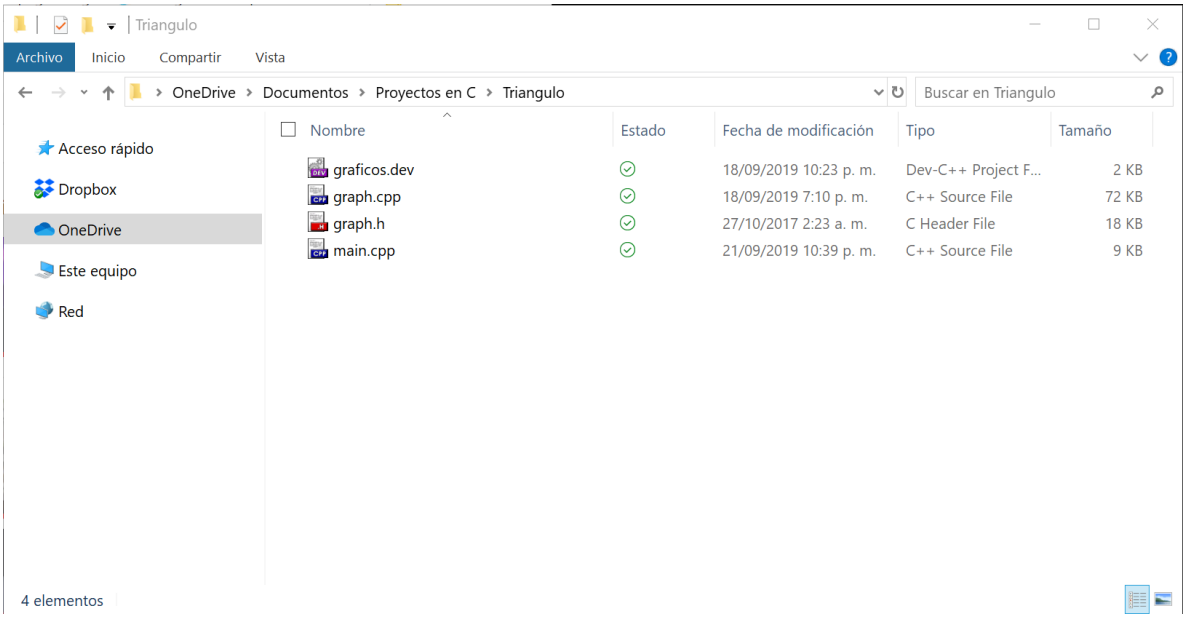


Figura 2. Ejemplo de ubicación de los archivos del proyecto.

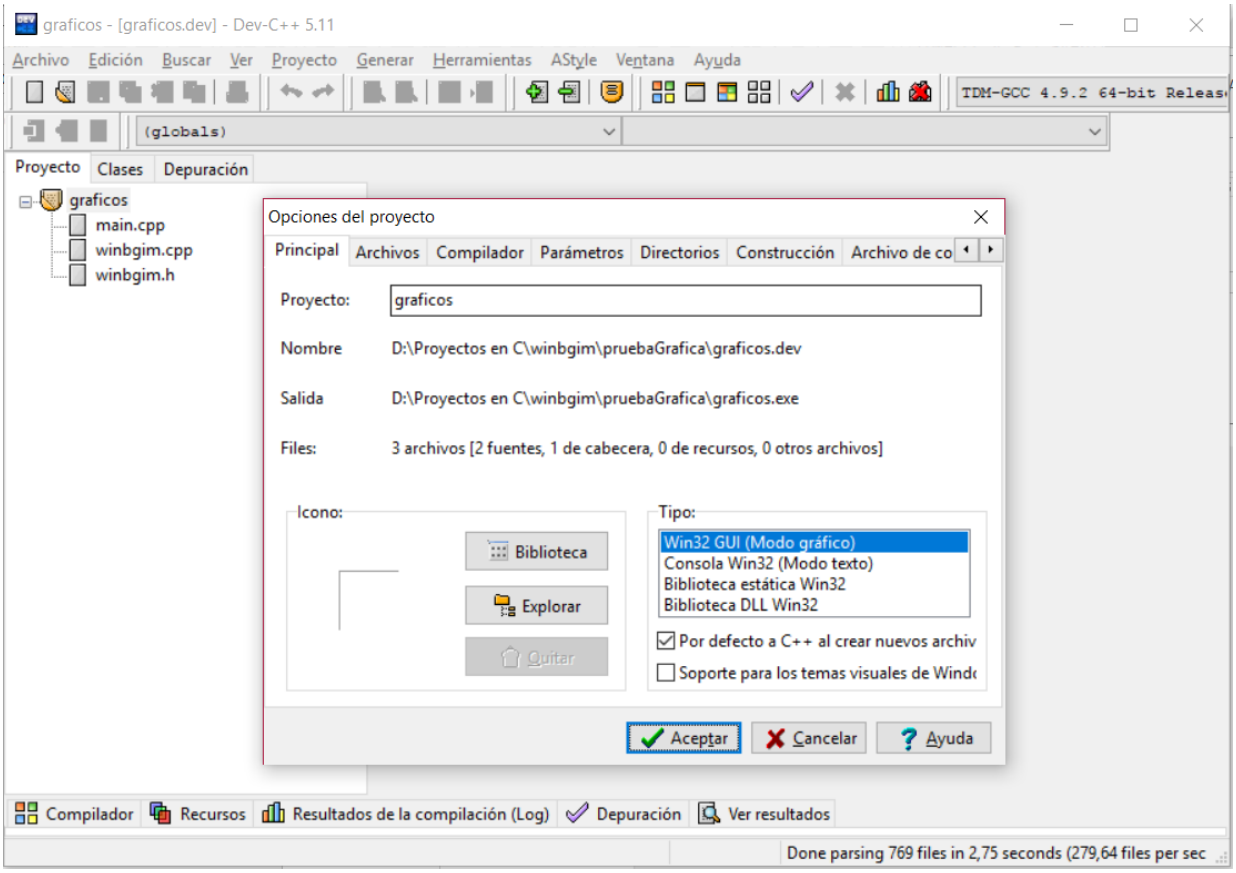
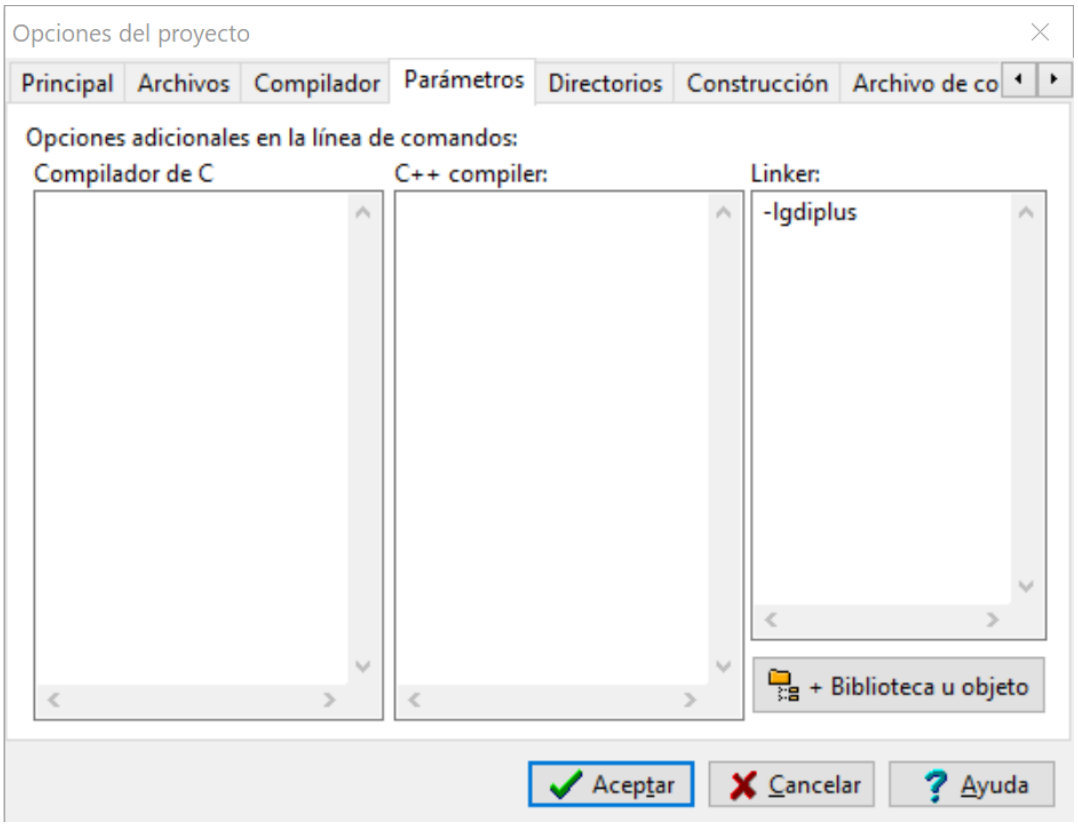


Figura 3. Opciones del proyecto

En este cuadro debemos seleccionar en la sección TIPO, la opción “Win32 GUI (Modo gráfico)”. Luego aceptamos.

Por último, ya que esta librería está basada en Windows, incluimos la sentencia **–lgdiplus** en las opciones del proyecto en la pestaña *parámetros,* porque está basado en este sistema, como muestra la figura siguiente:



¡Listo! Ya podemos empezar con nuestro código.

En el código

Lo primero que tenemos que hacer es vincular los archivos de la librería en el proyecto. Para esto con el puntero del ratón vamos a la sección del Proyecto (Tabulador), y con click derecho seleccionar en el menú la opción “Agregar al proyecto” como indica la figura 4. Aparecerá un cuadro de diálogo en donde seleccionamos los archivos y aceptamos (ver figura 5).

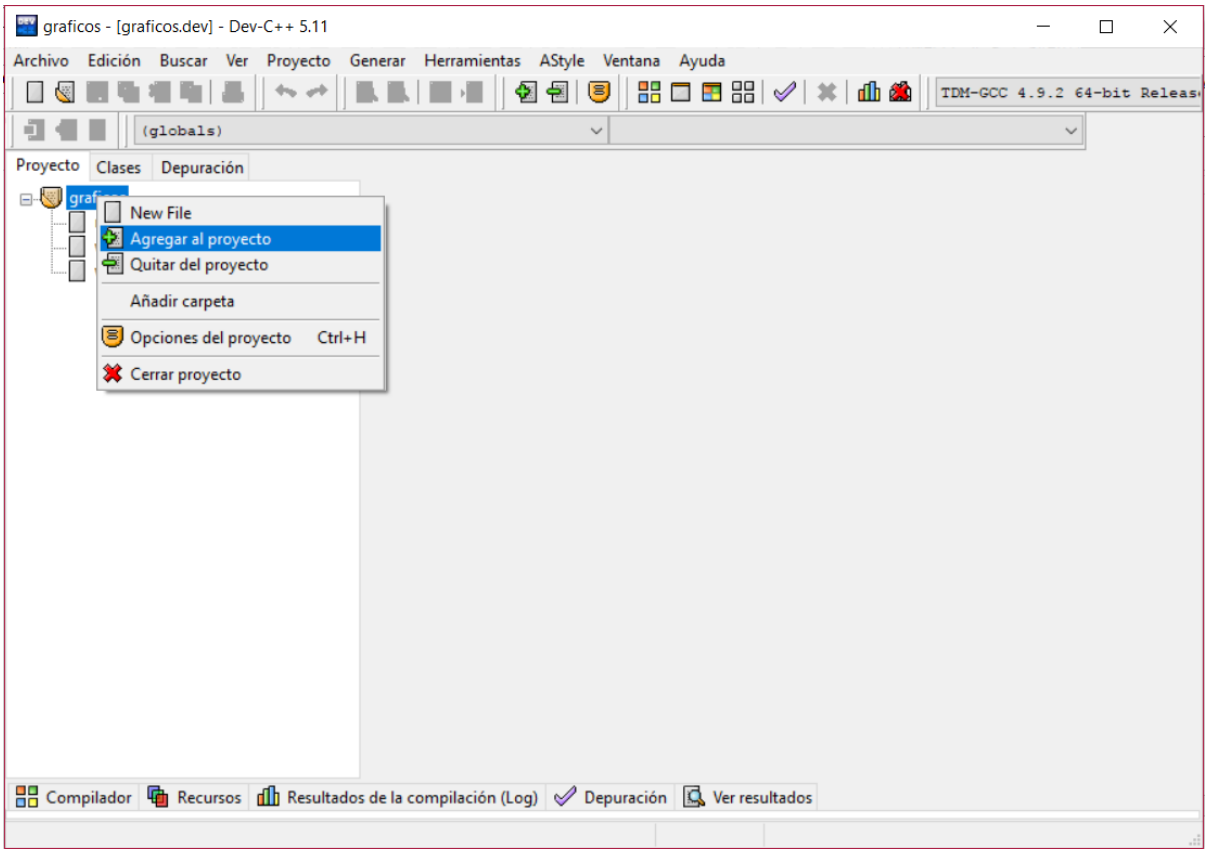


Figura 4. Opción para agregar archivos al proyecto

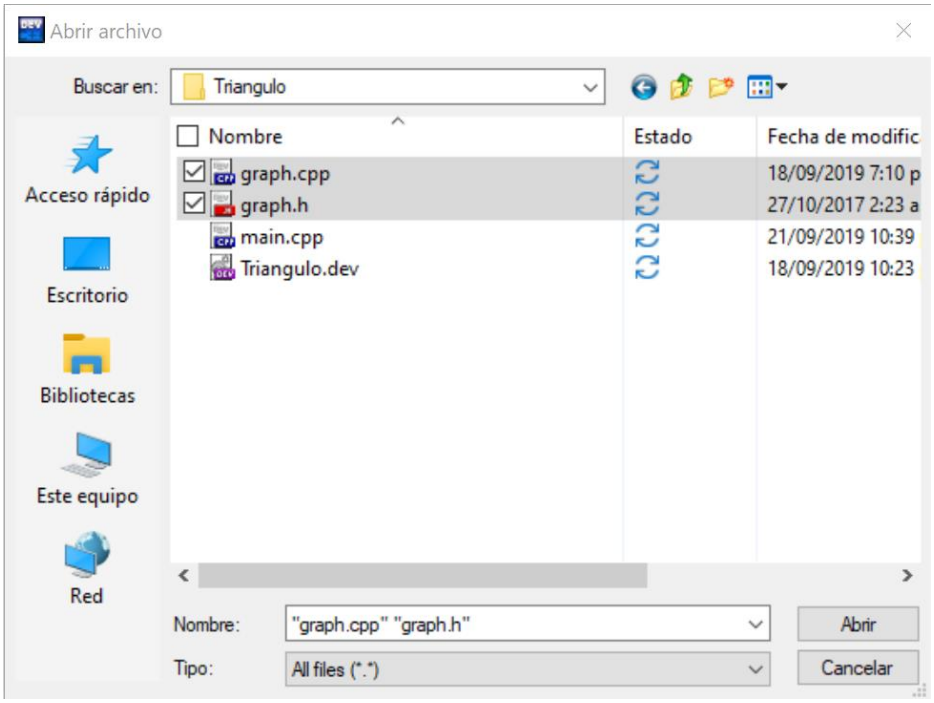


Figura 5. Cuadro de diálogo para seleccionar archivos a vincular.

Luego de vincular nuestros archivos de la librería, seleccionamos el archivo en donde encontraremos la función main(), o en donde vayamos a usar las funciones gráficas, e incluímos el archivo de encabezado winbgim.h con la directiva #include “winbgim.h”, como se ve en la figura 6.

Luego de esto, es usar las funciones que provee la librería de acuerdo a nuestra imaginación.

☺

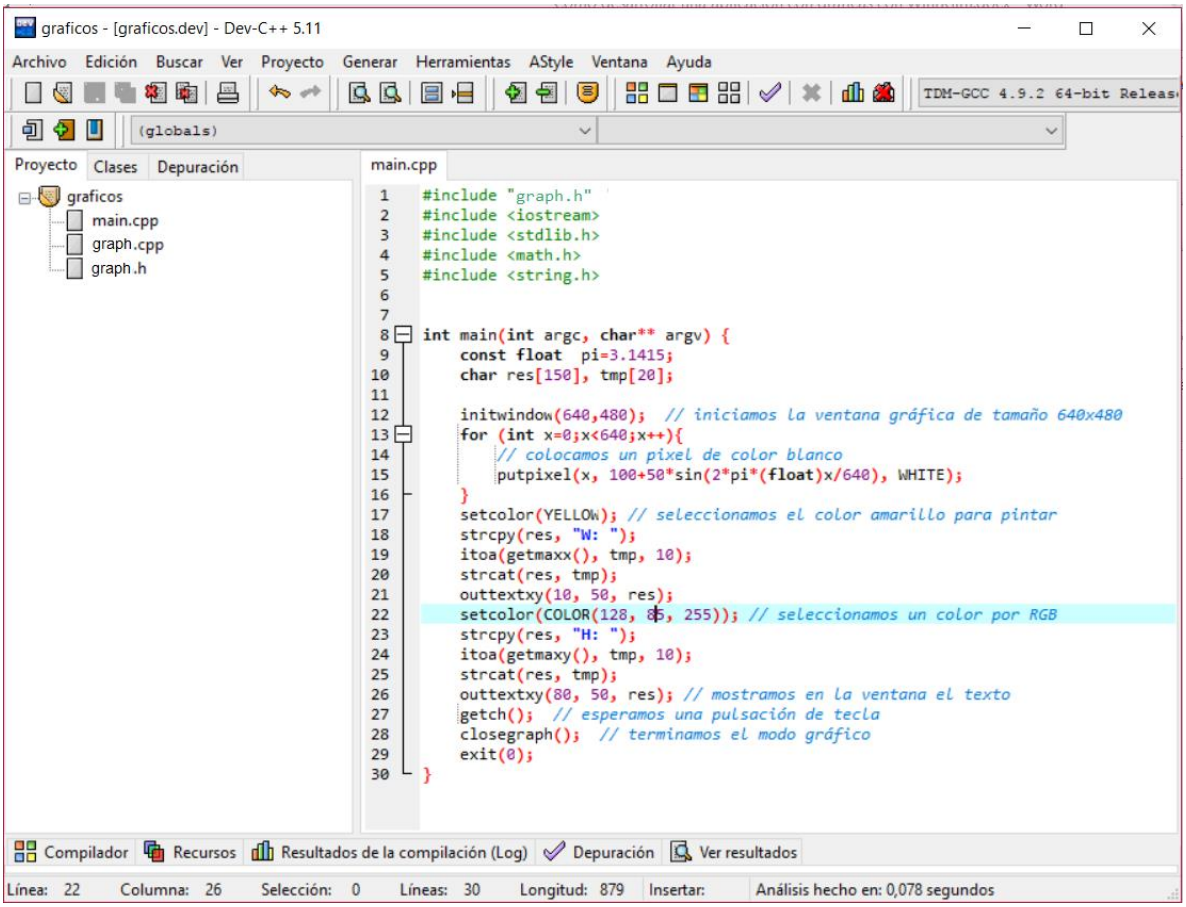


Figura 6. En la primera línea incluimos la librería para usar las funciones en el código.

Algunas rutinas o funciones

Rutinas de inicialización

void initwindow(int ancho, int alto);

Esta rutina hay que llamarla antes que cualquier otra de esta biblioteca, ya que inicializa el sistema gráfico y crea una ventana gráfica del tamaño especificado en los parámetros. El sistema de referencia empieza en las coordenadas 0,0, que representa la esquina superior izquierda, por lo que el eje y crece hacia la parte de abajo de la ventana.

void closegraph(void);

Al acabar de trabajar en el modo gráfico, hay que llamar a esta rutina, que libera toda la memoria reservada por el sistema gráfico y devuelve la pantalla al modo en el que estaba antes de inicializar el modo gráfico.

Rutinas de escritura

void cleardevice(void);

Esta rutina borra la pantalla gráfica rellenándola del color de fondo y mueve la posición de dibujo al origen de la pantalla, la posición (0,0).

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

Esta rutina dibuja una línea entre los puntos (x1, y1) y (x2, y2), con el color, el estilo de línea y el ancho actuales. No mueve la posición de escritura.

void lineto (int x, int y)

Traza una línea desde la posición actual de cursor hasta x, y.

void circle (int x, int y, int radius);

Esta rutina dibuja un círculo en el color de dibujo actual cuyo centro está en la posición (x, y) y de radio r.

Otras rutinas de dibujo

Estas otras rutinas dibujan diversas figuras geométricas:

void arc (int x, int y, int stangle, int endangle, int radius);

void bar(int izq, int arriba, int derecha, int abajo);

void bar3d (int left, int top, int right, int bottom, int depth, int topflag);

void ellipse (int x, int y, int stangle, int endangle, int xradius, int yradius);

void fillellipse (int x, int y, int xradius, int yradius);

*void 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, 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.

*void fillpoly (int numpoints, int *polypoints);*

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.

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

Esta función es usada para dibujar y rellenar 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.

void rectangle (int left, int top, int right, int bottom);

void 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.

void moveto (int x, int y);

Esta rutina mueve la posición de dibujo a la coordenada (x, y).

void outtext(char cad[]);

Esta rutina escribe una cadena de caracteres en pantalla, usando el tipo de letra, la dirección y el tamaño actuales.

void 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 *corresponde a*l valor del color del píxel.

void 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 rellenado.

Rutinas de lectura

int getbkcolor (void); int getcolor (void);
getcolor devuelve el color de dibujo actual. Ése es el color en el que escriben los puntos cuando se dibujan líneas y demás figuras. getbkcolor devuelve el color de fondo actual.

int getch (void);
Esta rutina lee un carácter de la pantalla gráfica sin esperar que se pulse la tecla de nueva línea. Para las teclas extendidas, primero se lee el valor 0 y, en una segunda lectura, se lee el valor correspondiente a cada tecla. Por ejemplo:

```
#define KEY_HOME    71 // tecla Inicio
#define KEY_END     79 // tecla Fin
#define KEY_UP      72 // cursor hacia arriba
#define KEY_LEFT    75 // cursor hacia la izquierda
#define KEY_RIGHT   77 // cursor hacia la derecha
#define KEY_DOWN    80 // cursor hacia abajo
#define KEY_F1      59 // tecla de función F1
#define KEY_F2      60 // tecla de función F2
```

int getmaxx (void); int getmaxy (void);
Estas rutinas devuelven el valor máximo de x y de y, respectivamente, para la pantalla gráfica.

unsigned getpixel (int x, int y);
Esta función devuelve el color del pixel de la coordenada (x,y).

int getx (void); int gety (void);
Estas rutinas devuelven, respectivamente, la coordenada x y la coordenada y de dibujo.

*int textheight(char far *texto); int textwidth(char far *texto);*
Esta función es usada para determinar la altura (textheight) o el ancho (textwidth) de la cadena de texto especificada por el argumento *texto. Estos valores son dados en píxels del texto especificado por el argumento. Los valores correspondientes del texto se determinan usando la fuente y el tamaño del carácter actual.

Rutinas de actualización de parámetros gráficos

void setcolor (int color); void setbkcolor (int color);
Estas rutinas establecen el nuevo color de dibujo y el nuevo color de fondo, respectivamente.

void setfillstyle(int trama, int color);

Esta función es usada para seleccionar una trama predefinida y un color de relleno. El argumento trama corresponde a 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 en la trama definida por el usuario. En su lugar, se debería usar la función setfillpattern. Las tramas son:

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

void 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.

Estilos de Líneas		
Constante	**Valor**	**Significado**
SOLID_LINE	0	continua
DOTTED_LINE	1	hecha con puntos
CENTER_LINE	2	centrada
DASHED_LINE	3	discontinua
USERBIT_LINE	4	definida por el usuario

Grosores para Líneas		
Constante	**Valor**	**Significado**
NORM_THICK	1	de 1 píxel
THICK_WIDTH	3	de 3 píxels

Modos de Escritura		
Constante	**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

void 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 a usar. El argumento orientacion corresponde a 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:

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

Rutinas varias

void delay (int miliseg);

Esta rutina hace que el programa se pare durante el número de milisegundos que se pasa como parámetro.

int kbhit (void);

Esta rutina devuelve 0 si no se ha pulsado ninguna tecla desde la última lectura y un valor distinto de 0 en caso contrario.

Rutinas del mouse

int mousex(void)

Retorna la coordenada x del Mouse relativa a la esquina superior izquierda

int mousey(void)

Retorna la coordenada y del Mouse relativa a la esquina superior izquierda

Los colores en Windows BGI

Hay dos maneras de referirse a los colores en WinBGI. La primera es usar la paleta tradicional de 16 colores, numerados del 0 al 15 y que tienen sus nombres propios que se pueden usar ara el fondo o el trazo:

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

La otra manera es formar un color indicando los tonos de rojo, azul y verde con valores entre 0 y 255. Esto se hace con la macro **COLOR(r, g, b)**, donde los tres parámetros indican el tono de cada color. Un ejemplo de estos colores se puede observar en el cuadro de diálogo del sistema de colores del Sistema Operativo (ver figura 7).

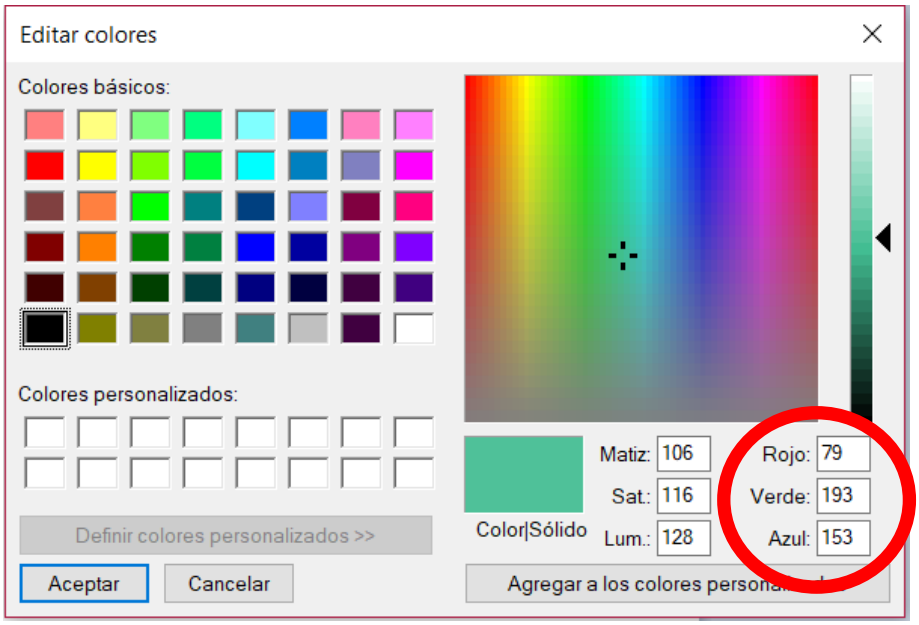


Figura 7. Cuadro de edición de colores el aplicativo Paint de Windows. En el círculo se muestra el esquema RGB.

Referencias

1. Borland BGI Graphics emulation for the MingW (GCC port) compiler, en: http://winbgim.codecutter.org/
2. Borland Graphics Interface (BGI) for Windows, en: http://www.cs.colorado.edu/~main/cs1300/doc/bgi/bgi.html
3. Biblioteca graphics Borland® C, en http://c.conclase.net/borland/?borlandlib=graphics#inicio