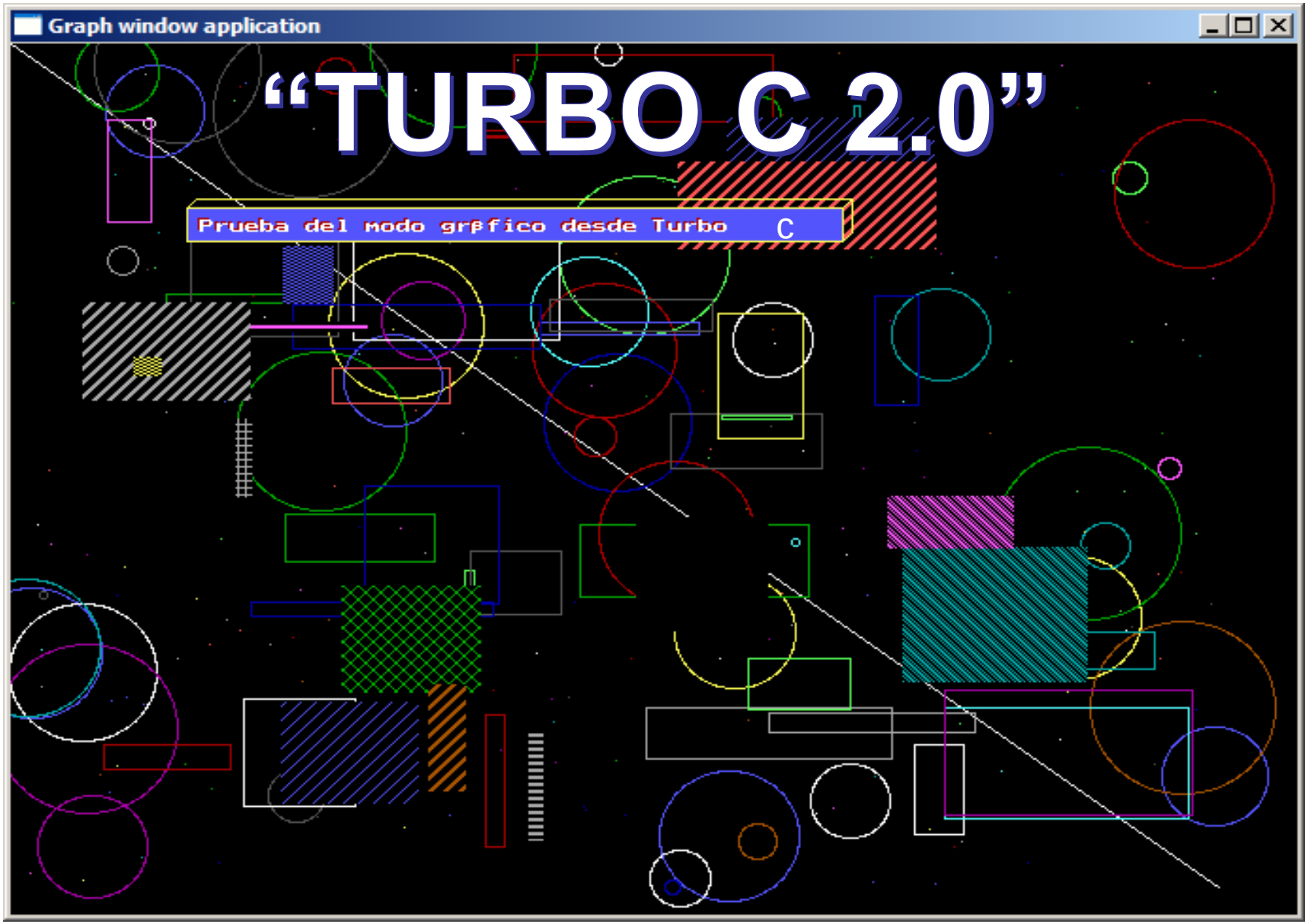
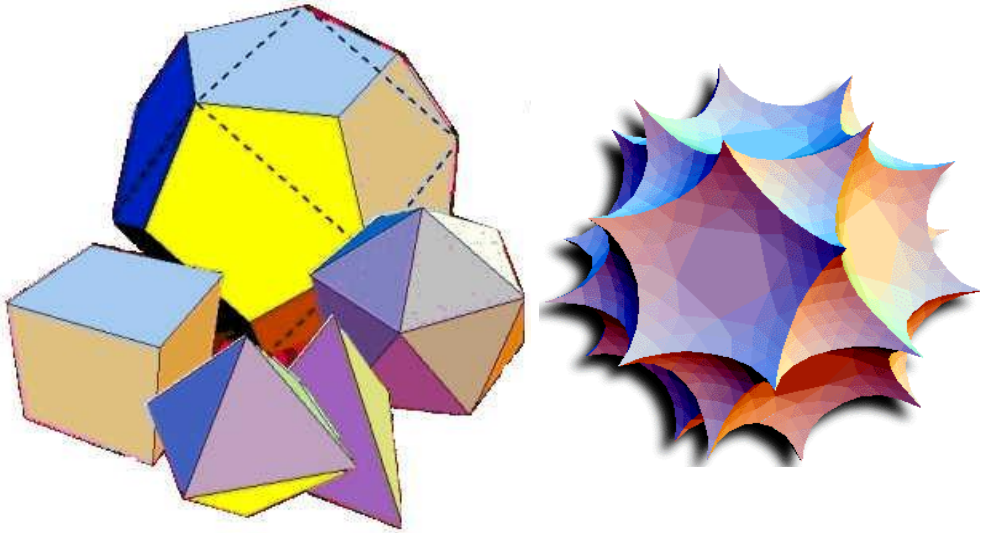
INTRODUCCIÓN A LA PROGRAMACIÓN GRÁFICA EN



ÍNDICE



¿Qué son los primitivos gráficos?

Definición:

Un primitivo grafico es un elemento fundamental de dibujo en un paquete grafico tal como un **punto**, **línea**, o **circulo**; pueden ser un carácter, o pueden se una **operación** tal como **relleno**, **coloreado** o **transferido de la imagen**. C cuenta con cinco grupos de primitivos gráficos.

1- Figuras Geométricas:
Dibujan las figuras de geometría clásica: **líneas**, **círculos**, **arcos**, **rectángulos**, **polígonos**, etc.

2- Relleno: Tiene dos formas de realizarse. El primero es con **polígonos**, donde se definen los **vértices** del **polígono** a ser rellenados la segunda es una **operación grafica** que busca algorítmicamente las **fronteras** de la función de **relleno**.

3- Rasterop: Es una **operación grafica** que **copia** el **área** de una **imagen** para luego **dibujarla** en cualquier **región** de la **pantalla**.

4- Graficas Matemáticas: Dibuja los primitivos **barras** y **sectores** para conseguir dibujar las herramientas del sector.

5- Texto Grafico: Sirve para escribir **texto** en **modo grafico**, utilizando **diferentes fuentes**.

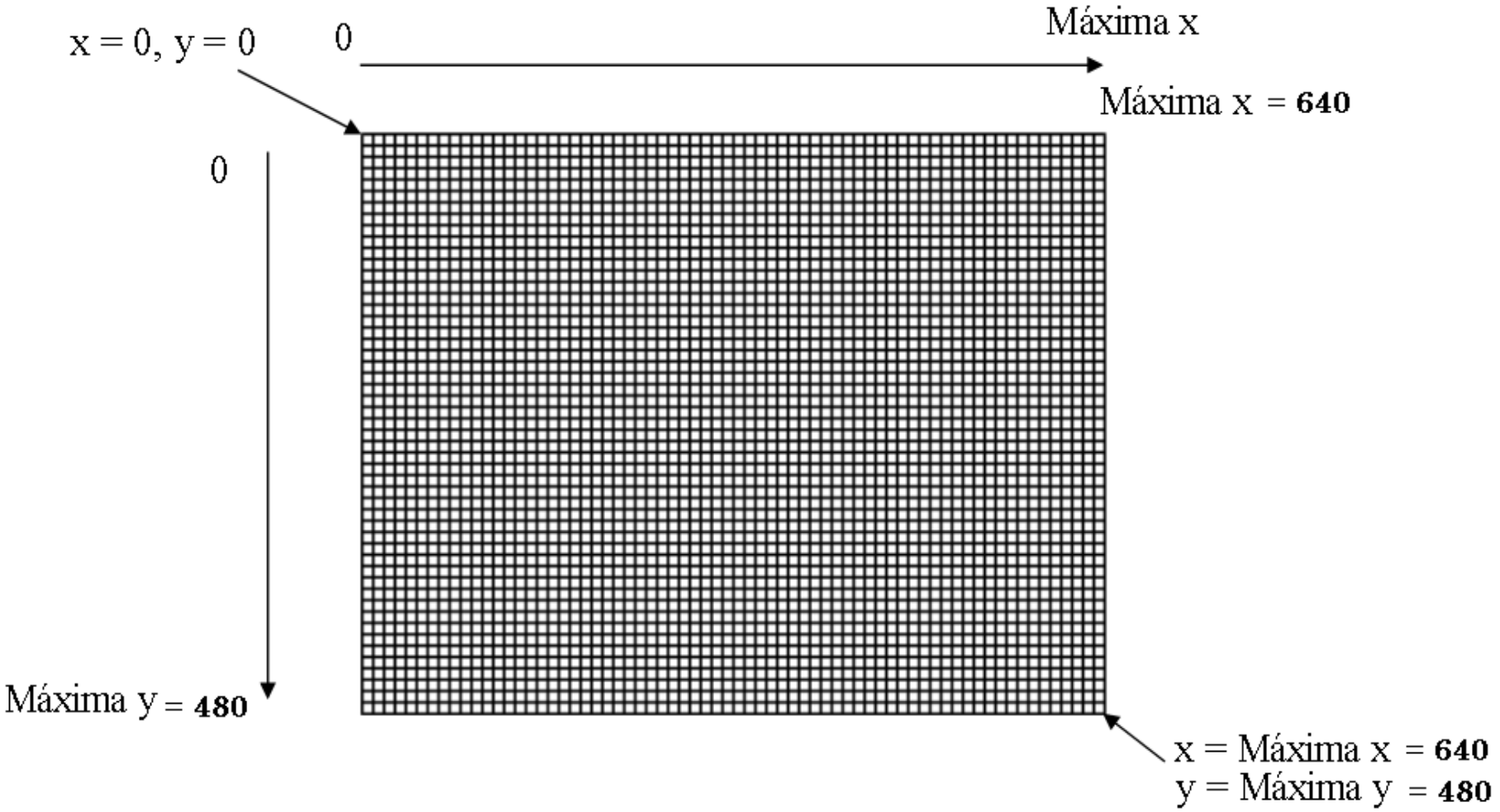
Modo Gráfico

Para describir un modo grafico debemos conocer primero algunos conceptos básicos:

- **Pixel:** es un punto en la pantalla.
- **Resolución:** cantidad de pixeles que caben en una pantalla.

Bueno el modo que estudiaremos tiene una resolución de **640** x **480**.

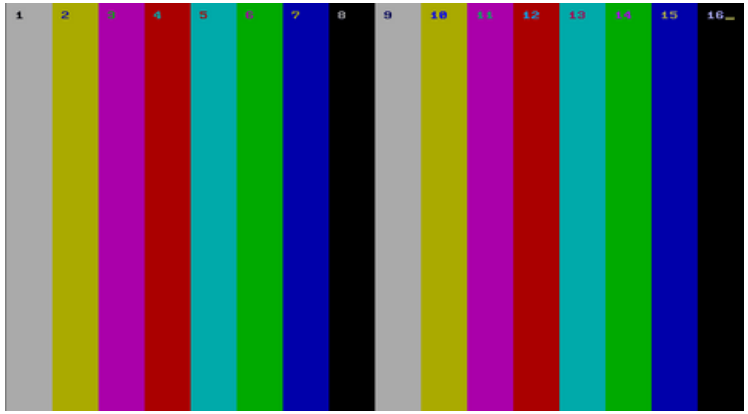
La pantalla gráfica



Modo Gráfico

Pero ¿Que significa esto?, fácil, en una pantalla caben 640 pixeles a lo largo y 480 pixeles a lo alto. Además nuestros pixeles pueden ser elegidos entre 16 colores distintos.

Negro	BLACK	0
Azul	BLUE	1
Verde	GREEN	2
Cyan	CYAN	3
Rojo	RED	4
Magenta	MAGENTA	5
Marrón	BROWN	6
Gris	LIGHTGRAY	7
Gris oscuro	DARKGRAY	8
Azul intenso	LIGHTBLUE	9
Verde intenso	LIGHTGREEN	10
Cyan intenso	LIGHTCYAN	11
Rojo intenso	LIGHTRED	12
Magenta intenso	LIGHTMAGENTA	13
Amarillo	YELLOW	14
Blanco	WHITE	15

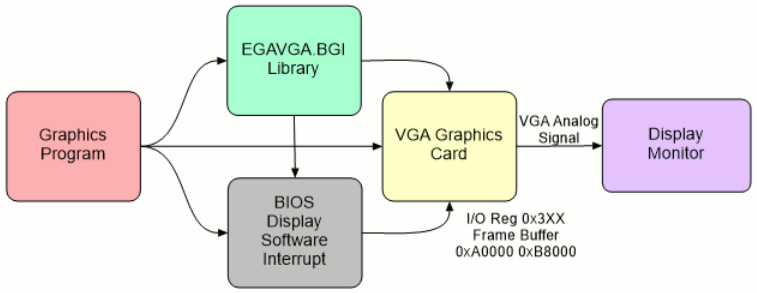


Entrando al Modo Gráfico

Para cargar un modo grafico el Turbo C nos entrega una serie de archivos que nos permite instalar el modo de video, estos archivos tienen extensión BGI y representan el modo de video con el que se va a trabajar (CGA, HERCULES, EGA). Lamentablemente como estos programas son un poco antiguos no incluyen archivos BGI para los modos VGA ni SUPER VGA.

DRIVERS BGI:

DRIVER	MODOS	VALOR GMODE	RESOLUCION	PALETA
CGA gdriver=1	CGAC0 CGAC1 CGAC2 CGAC3 CGAHI	0 1 2 3 4	320x200 320x200 320x200 320x200 640x200	C0 C1 C2 C3 2 colores
MCGA gdriver=2	MCGAC0 MCGAC1 MCGAC2 MCGAC3 MCGAMED MCGAHI	0 1 2 3 4 5	320x200 320x200 320x200 320x200 640x200 640x480	C0 C1 C2 C3 2 colores 2 colores
EGA gdriver=3	EGALO EGAHI	0 1	640x200 640x350	16 colores 16 colores
VGA gdriver=9	VGALO VGAMED VGAHI	0 1 2	640x200 640x350 640x480	16 colores 16 colores 16 colores
OTROS VALORES: Existen otros drivers BGI (Hercules, ATT400, IBM8514) que ya han caido en desuso y no resultan útiles para el programador. SVGA256.BGI: TWK256.BGI: Borland C dispone ya de drivers SVGA en formato BGI, además de drivers para modos tweaked o unchained como el modo 13X, X, Y, Q, etc...				



Modo Gráfico

El procedimiento que nos permite entrar al modo grafico se llama INITGRAPH y necesita la librería GRAPHICS.H (En la primera línea de nuestro programa).

Veamos como funciona:

Sintaxis: **initgraph**(&numdriver, &modvideo, path);

Como ven el procedimiento necesita 3 parámetros, los 2 primeros son de tipo integer y el segundo es un string. Veamos que significan:

Modo Gráfico

- numdriver: es una variable que contiene el modo de video que vamos a cargar. Nosotros vamos a ocupar el 3 (EGAVGA.BGI).

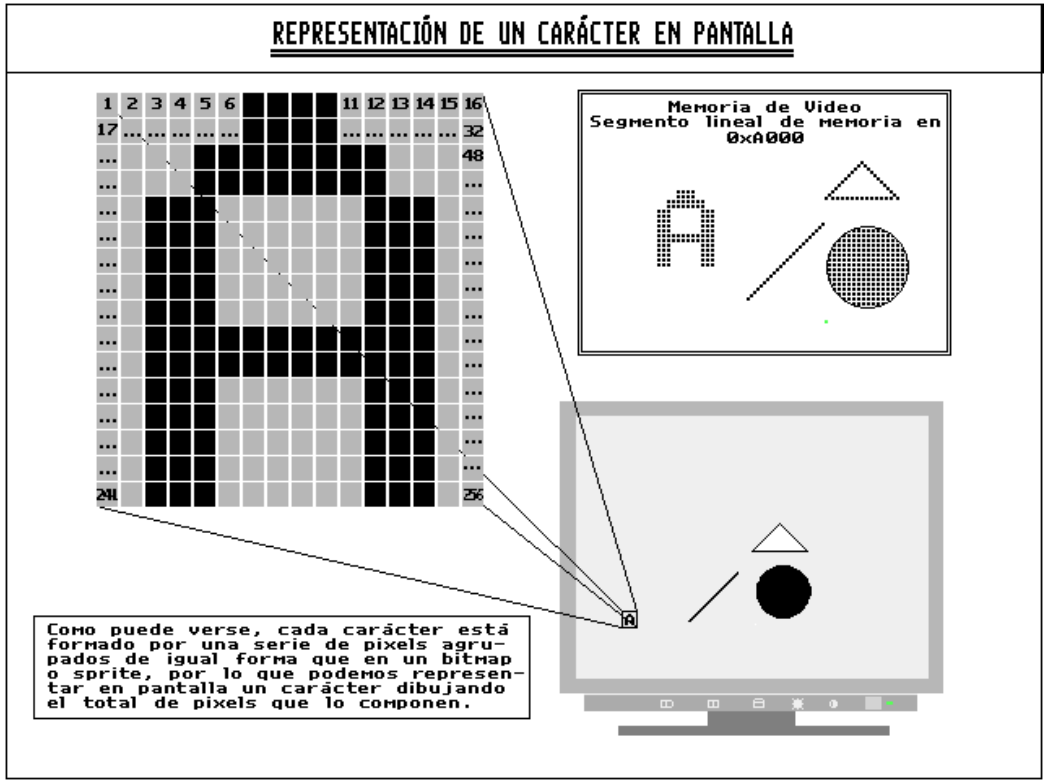
- modvideo: 1 Modo texto y 8 Modo Gráfico.

- path: es para especificar el directorio donde se encuentra el manejador grafico o el archivo con extensión BGI en la carpeta BIN.

Modo Gráfico

Estructura de un programa gráfico:

1. incluir la librería gráfica: **#include<graphics.h>**
2. inicializar el modo gráfico.
3. **Dibujar algo.**
4. Cerrar el modo grafico.



Modo Gráfico

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

void main(void)
{

  int driver = DETECT,modo,codigo;
  initgraph(&driver,&modo,"c:\\tc20\\bin");
  cleardevice();  /*system("cls")*/  /*Borrar Pantalla*/

    /*Dibujar*/

  closegraph (); /*restorecrtmode();*/ /*Restaurar el modo gráfico*/
  getch();

} /*Fin del programa principal*/
```

FUNCIONES BASICAS DE GRAPHICS.H.

setbkcolor(**color**);

Esta función es usada para asignar el color de fondo especificado por el argumento color. Existen varios valores para ciertos colores de fondo.

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.

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

FUNCIONES BASICAS DE GRAPHICS.H.

```
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
void main (void)
    {
        int adap=DETECT, modo, color;
          initgraph(&adap,&modo,"C:\\tc20\\bin ");
        cleardevice();
              setbkcolor(6);

              color=getbkcolor();
       getch();
       closegraph();
     }
```

FUNCIONES DE CONTROL DE ATRIBUTOS.

setcolor (color);

Esta función coloca el atributo color, es decir escoge un color entre **0** y **15** su equivalente en ingles, todo lo que dibuje después de esta instrucción tendrá el color establecido por la función **setcolor**.

setlinestyle (estilo, patron, grosor);

Fija los atributos de estilo y grosor de las líneas dibujadas por algunas funciones de dibujo.

El parámetro de **estilo** escoge uno de varios estilos predefinidos tales como punteado, quebrado, etc.

El parámetro **grosor** selecciona un grosor de **1** ó de **3 bits**.

La siguiente figura demuestra los estilos predefinidos de línea, con los dos grosores, el tipo de línea por defecto es continuo, con grosor normal.

FUNCIONES DE CONTROL DE ATRIBUTOS.

Controla el estilo y grosor de las líneas dibujadas con las funciones **line**, **rectangle**, **drawpoly**, **fillpoly** y **bar3d**. En **pieslice**, fija el patrón y grosor de los radios. En **pieslice**, **circle**, **arc** y **ellipse**, fija el grosor de la circunferencia, pero esta se dibuja como una curva continua a pesar del valor de **patrón**.

	Grosor	
Estilo	**NORM_WIDTH**	**THICK_WIDTH**
SOLID_LINE	———	▬▬▬
DOTTED_LINE	.....	·····
CENTER_LINE	.....	▬ ▬ ▬ ▬ ▬ ▬ ▪
DASHED_LINE	– . – . – . – . – . – .	▬ · ▬ · ▬ · ▬ ·
USERBIT_LINE	.....	▪▪▪▪▪

getpixel (x, y);

Devuelve el valor del píxel que está en la posición (**x**, **y**).

getmaxcolor ();

Regresa el índice máximo de color para el modo de operación actual del adaptador. El índice mínimo es siempre **0**. Entonces, se regresa el número de colores disponibles menos **1**.

FUNCIONES DE CONTROL DE ATRIBUTOS.

getmaxx();

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 = 639.

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

getmaxy();

Esta función es usada para obtener la coordenada máxima de la pantalla en la dirección vertical. Este valor suele ser la resolución vertical máxima menos 1 = 479.

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

FUNCIONES BASICAS DE GRAPHICS.H.

```
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
void main (void)
    {
      int adap=DETECT, modo, x_max,y_max;
        initgraph(&adap,&modo,"C:\\tc20\\bin ");
      cleardevice();
          x_max=getmaxx();
          y_max=getmaxy();
      closegraph();
        printf("x maxima:%d\ty maxima:%d\n",x_max,y_max);
      getch();
    }
```

LAS FUNCIONES DE TEXTO GRÁFICO:

outtextxy (**x**, **y**, ***cadena**);

outtext (***cadena**) ;

settextstyle (**Tipo_Fuente**, **direccion**, **tamano**) ;

Fija la fuente, la dirección de escritura y el tamaño del texto a los valores de los parámetros **nuevo fuente**, **dirección** y **tamaño**. Hay una fuente de caracteres de puntos y cuatro fuentes de plumados. La dirección puede ser horizontal o vertical. El tamaño puede ser un factor integral de agrandamiento en la horizontal y la vertical.

LAS FUNCIONES DE TEXTO GRÁFICO:

Fuente	Clase de Caracteres
DEFAULT_FONT	8 x 8 matriz de puntos
TRIPLEX_FONT	plumado triplex
SMALL_FONT	plumado pequeño
SANS_SERIF_FONT	plumado sans serif
GOTHIC_FONT	plumado gótico

Dirección	descripción
HORIZ_DIR	horizontal (de la izquierda a la derecha)
VERT_DIR	vertical (de abajo hacia arriba)

Tamaño	descripción
USER_CHAR_SIZE	Usa el tamaño fijado en la fn. **setusercharsize**.
1-10	Escala (1=más pequeña, 10=más grande)

Por defecto se usa la fuente **DEFAULT_FONT** con tamaño 1 y dirección horizontal.

LAS FUNCIONES DE TEXTO GRÁFICO:

settextjustify (jus_thoriz, just_vert);

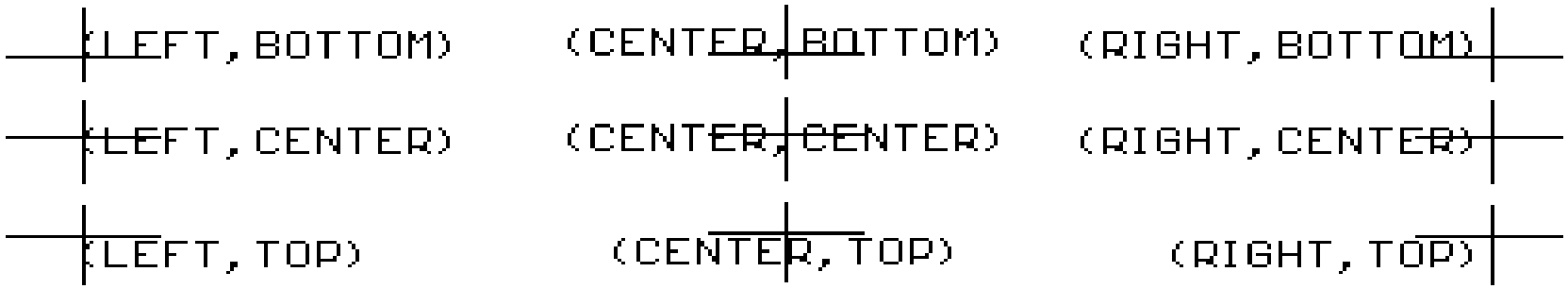
Fija el tipo de justificación del texto en las direcciones horizontal y vertical. Se justifica con respecto a un punto de referencia (**x**,**y**) en el caso de **outtextxy**, o con respecto al Puntero Gráfico en el caso de **outtext**. A continuación se dan los valores predefinidos para los parámetros.

Just_horiz	**Ubicación Horiz. con Respecto al Punto de Ref.**
LEFT_TEXT	la izquierda del texto
CENTER_TEXT	centro horizontal del texto
RIGHT_TEXT	la derecha del texto

Just_vert	**Ubicación Vertical con Respecto al Punto de Ref.**
TOP_TEXT	la parte superior del texto
CENTER_TEXT	el centro vertical del texto
BOTTOM_TEXT	la parte inferior del texto

LAS FUNCIONES DE TEXTO GRÁFICO:

settextjustify (jus_thoriz, just_vert);



ATRIBUTOS DE RELLENOS:

Turbo C utiliza dos métodos para definir la región de relleno. El primer relleno de **polígonos, usa la lista de vértices del polígono para calcular la geometría del interior**, el segundo relleno es por el **método de inundación, busca desde un punto inicial llamada la semilla en todas las direcciones para encontrar una frontera que se encierre la regional frontera se reconoce como el valor del píxel que tiene**.

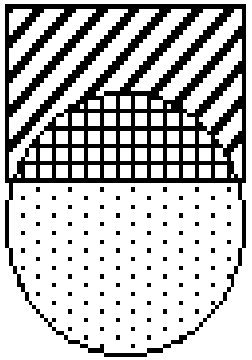
La función **setfillstyle** que sea de gran importancia a la hora de realizar los dos tipos de rellenado y **bar que es una función similar a rectangle**.

RELLENO POR INUNDACIÓN:

floodfill (x, y, col_frontera);

Rellena por inundación la región que encierra al punto (**x**,**y**) con una frontera de color **col_frontera**, usando el patrón y el color de relleno seleccionados.

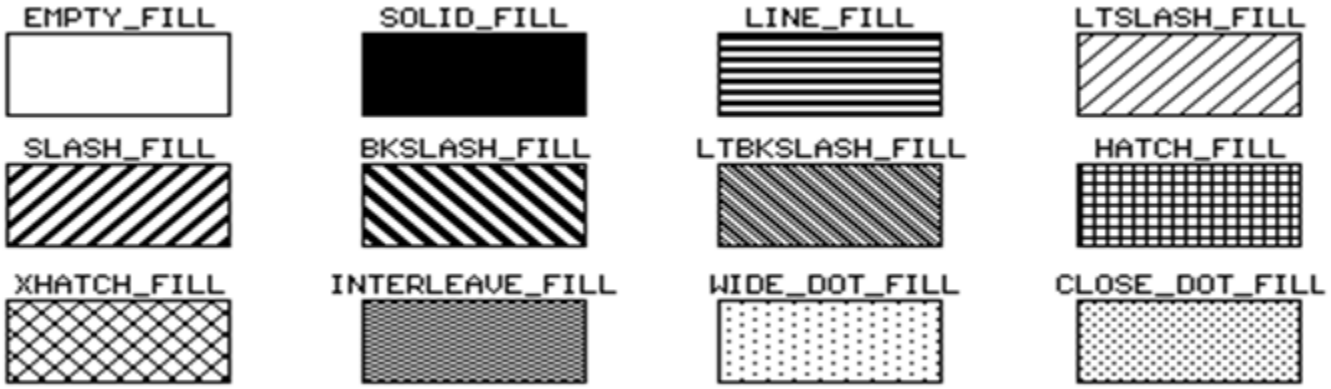
```
#include <conio.h>
#include <graphics.h>
void main (void)
    {
      int adap=DETECT, maxcol=3;
      initgraph(&adap,&modo,"C:\\tc20\\bin ");
      cleardevice();
          setcolor (maxcol);
              circle (100, 100, 50);
           floodfill (100, 100, maxcol);
     getch();
     closegraph();
    }
```



FUNCION SETFILLSTYLE:

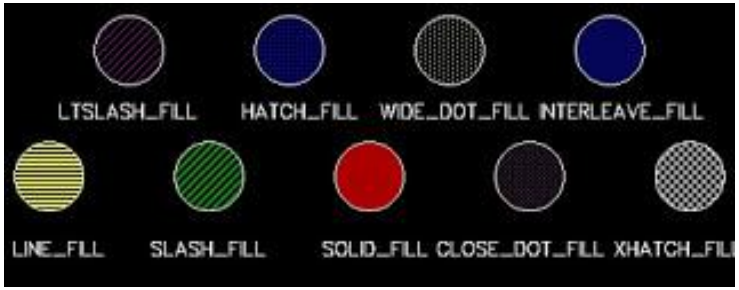
setfillstlyle (trama_relleno, color_relleno);

Fija el patrón actual de relleno a "**trama_relleno**" y el color actual de relleno a "**color_relleno**". Nótese que el color actual de relleno es distinto del color actual de dibujo. Hay 12 tipos de relleno predefinidos que se dan en la siguiente figura. Incluido un patrón vacío (**EMPTY_FILL**) que no rellena, y un patrón sólido (**SOLID_FILL**). Además, se permite definir un patrón de relleno propio con la función **setfillpattern**.



FUNCION SETFILLSTYLE:

```
#include <graphics.h>
#include <conio.h>
void main (void) {
                    int driver = DETECT,modo;
                    initgraph( &driver, &modo, "c:\\tc20\\bin" );
                    cleardevice();
                      setfillstyle( LTSLASH_FILL, 6 );
                        bar( 50, 50, 350, 300 );
                    getch();
                    closegraph();
                   }
```



FUNCIONES GRÁFICAS PARA EL DIBUJO DE FIGURAS GEOMÉTRICAS:

- **putpixel** (int **x**, int **y**, int **col**);

Dibuja un pixel (punto) en la posición (**x**,**y**) con el color col. El píxel es el área más pequeña de la pantalla que se pueda controlar.

- **line** (int **x1**, int **y1**, int **x2**, int **y2**);

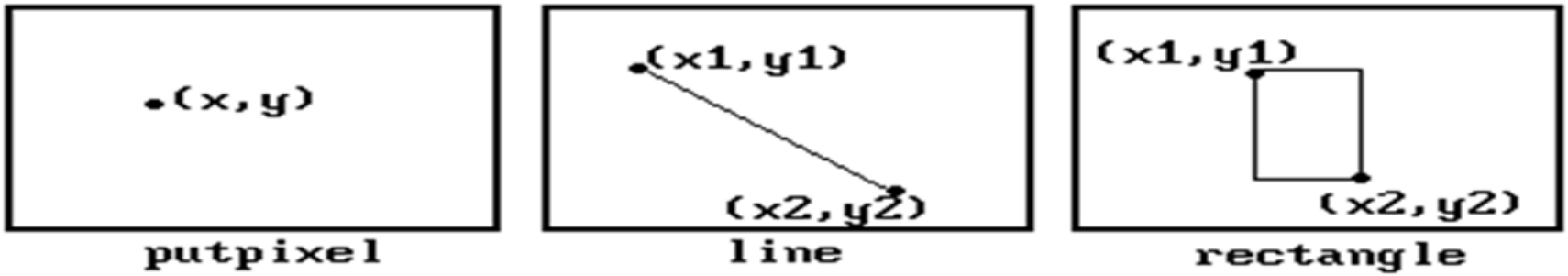
Dibuja una línea desde (**x1**, **y1**) hasta (**x2**, **y2**) usando el color fijado por **setcolor** y el patrón y grosor de línea fijados por **setlinestyle**.

- **rectangle** (int **x1**, int **y1**, int **x2**, int **y2**);

Dibuja un rectangle con izquierda superior (**x1**, **y1**) y derecha inferior en (**x2**, **y2**) usando el color fijado por **setcolor** y el patrón y grosor de línea fijados por **setlinestyle**.

FUNCIONES GRÁFICAS PARA EL DIBUJO DE FIGURAS GEOMÉTRICAS:

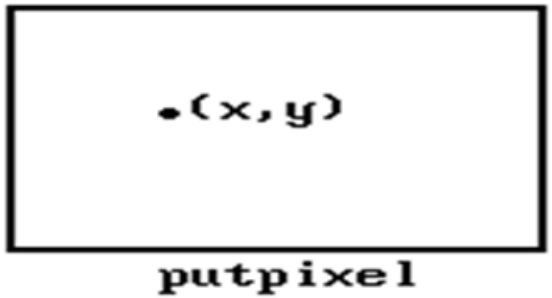
- **putpixel** (**x**, **y**, **col**);
- **line** (**x1**, **y1**, **x2**, **y2**);
- **rectangle** (**x1**, **y1**, **x2**, **y2**);



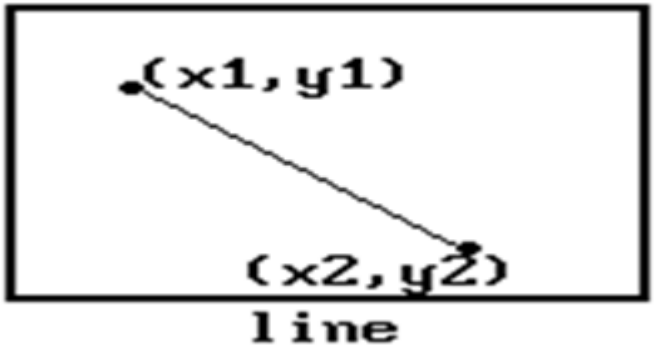
Salida de las funciones **putpixel**, **line** y **rectangle**.

FUNCIONES GRÁFICAS PARA EL DIBUJO DE FIGURAS GEOMÉTRICAS:

```
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
void main (void)
    {
        int adap=DETECT,modo,t;
          initgraph(&adap,&modo,"C:\\tc20\\bin ");
        cleardevice();
          for (t=0;t<200;t++)
                { putpixel(100+t,50+t,RED); }
                 getch();
          closegraph();
    }
          /*****************Pixel*****************/
```



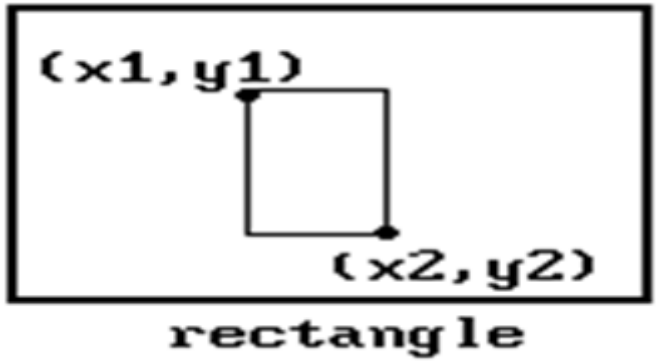
FUNCIONES GRÁFICAS PARA EL DIBUJO DE FIGURAS GEOMÉTRICAS:




```
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
void main (void)
    {
        int adap=DETECT,modo;
          initgraph(&adap,&modo,"C:\\tc20\\bin ");
        cleardevice();
         line(0,0,getmaxx()/2,getmaxy()/2);
       getch();
       closegraph();
    }              /*****************Línea****************/
```

FUNCIONES GRÁFICAS PARA EL DIBUJO DE FIGURAS GEOMÉTRICAS:

```
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
void main (void)
     {
          int adap=DETECT,modo;
            initgraph(&adap,&modo,"C:\\tc20\\bin ");
          cleardevice();
           rectangle(40,40,400,300);
         getch();
         closegraph();
      }
/*****************Rectángulo*****************/
```



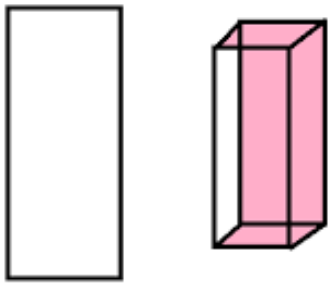
FUNCION BAR:

bar (izq, sup, der, inf);

bar3d(200,100,300,150,25,1);

Esta función dibujara una barra rectangular y rellenada de dos dimensiones. La **Esquina superior izquierda** de la barra rectangular esta definida por los argumentos izquierda y superiores 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 **setfillstyle**.



FUNCIONES GRÁFICAS PARA EL DIBUJO DE FIGURAS GEOMÉTRICAS:

- **circle** (int **x**, int **y**, int **rad**);

Dibuja un circulo con centro (**x**, **y**) y **radio rad** usando el color fijado por **setcolor** y grosor de línea fijados por **setlinestyle**.

- **void arc** (int **x**, int **y**, int **ang1**, int **ang2**, int **rad**);

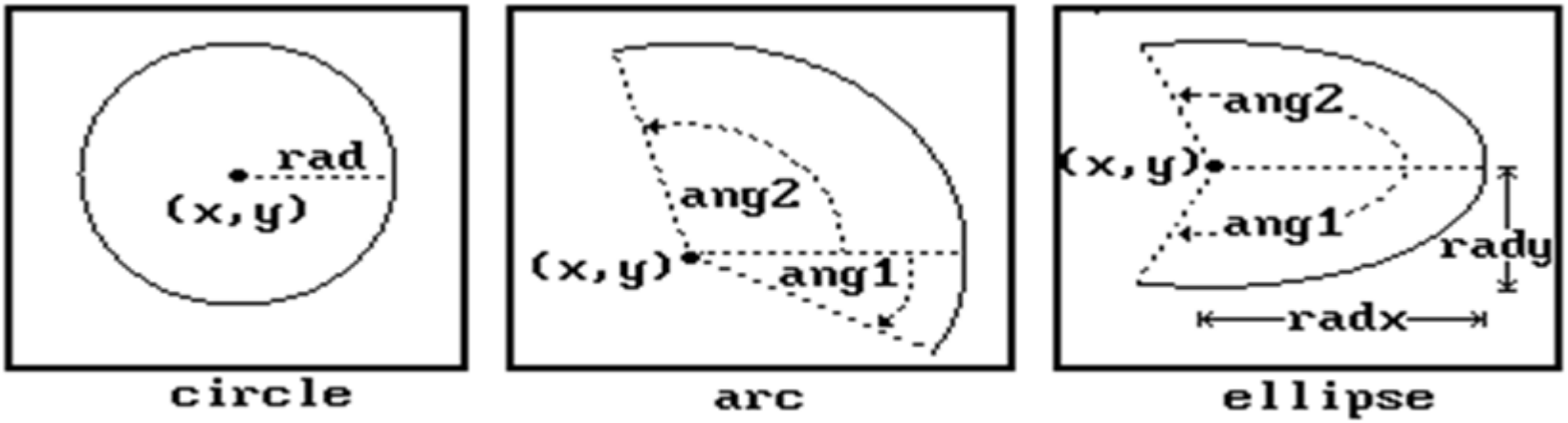
Dibuja un **arco** con centro (**x**,**y**), ángulo inicial (en grados) **ang1**, ángulo terminal (en grados) **ang2** y **radio rad**, usando el color actual y el grosor de línea fijado por **setlinestyle**.

- **void ellipse** (int **x**, int **y**, int **ang1**, int **ang2**, int **radx**, int **rady**);

Dibuja un arco de elipse con el centro en (**x**,**y**), ángulo inicial (en grados) **ang1**, ángulo terminal (en grados) **ang2**, radio horizontal **radx** y **radio** vertical **rady**, usando el color actual fijado por **setcolor** y el grosor de línea fijado por **setlinestyle**.

FUNCIONES GRÁFICAS PARA EL DIBUJO DE FIGURAS GEOMÉTRICAS:

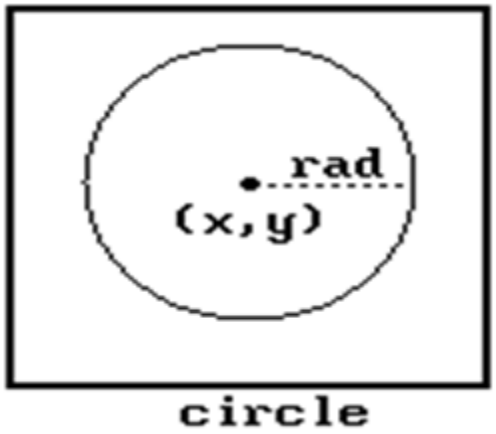
- **circle** (**x**, **y**, **rad**);
- **arc** (**x**, **y**, **ang1**, **ang2**, **rad**);
- **ellipse** (**x**, **y**, **ang1**, **ang2**, **radx**, **rady**);



Salida de las funciones **circle**, arc y **ellipse**.

FUNCIONES GRÁFICAS PARA EL DIBUJO DE FIGURAS GEOMÉTRICAS:

```
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
void main (void)
    {
        int adap=DETECT,modo;
          initgraph(&adap,&modo,"C:\\tc20\\bin ");
        cleardevice();
         circle(300,200,80);
       getch();
       closegraph();
     }
        /*****************Circulo*****************/
```



ARCOS

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

Esta función creara 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.

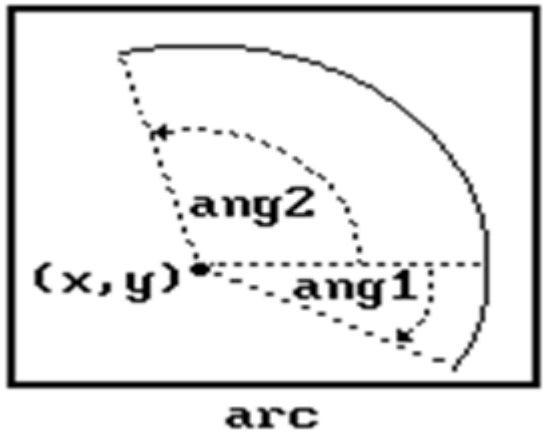
El arco no esta rellenado, pero es dibujado usando el color actual .

El arco comienza al ángulo especificado por el argumento comienzo_angulo, se dibuja en la dirección contraria a las agujas del reloj hasta llegar al ángulo especificado por el argumento ángulo final.

La función arc (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 fusión **arc** sin embargo, ignorara el argumento trama de la función setlinestyle.

FUNCIONES GRÁFICAS PARA EL DIBUJO DE FIGURAS GEOMÉTRICAS:

```
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
void main (void)
    {
        int adap=DETECT, modo, radio;
          initgraph(&adap,&modo,"C:\\tc20\\bin ");
        cleardevice();
         for (radio=200;radio<254;radio +=200)
                { arc(320,240,45,150,radio); }
       getch();
       closegraph();
     }
        /*****************Arcos*****************/
```



arc

ELIPSES

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

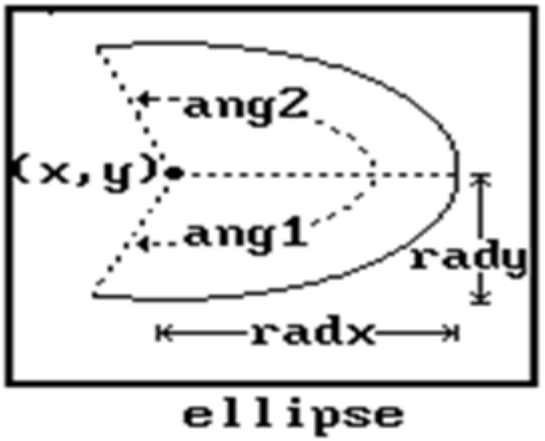
Esta función es usada para dibujar un arco elíptico en el color actual.

El arco elíptico esta centrado en el punto especificado por los argumentos **x** e **y**. Ya que el arco es elíptico 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 a este el eje horizontal a la derecha del centro de la elipse ser 0 grados .

El arco elíptico es dibujado con el grosor de línea actual con esta establecida por la función **setlinestyle**. Sin embargo, el estilo de línea es ignorado por la función elipse.

FUNCIONES GRÁFICAS PARA EL DIBUJO DE FIGURAS GEOMÉTRICAS:




```
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include <stdio.h>
void main (void)
    {
        int adap=DETECT, modo, radio;
          initgraph(&adap,&modo,"C:\\tc20\\bin ");
        cleardevice();
       ellipse(getmaxx()/2, getmaxy()/2,360,180,50,25);
       getch();
       closegraph();
     }
        /****************Ellipse****************/
```

RELLENO DE POLÍGONOS:

El método de "relleno de polígonos" usa la lista de vértices de un polígono para calcular precisamente donde rellenar.

drawpoly (**n_puntos**, ***vertice**);

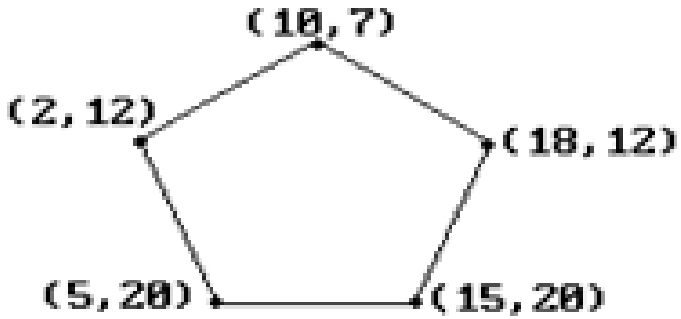
Dibuja una polilínea (una figura de segmentos de líneas unidas) de **n_puntos** vértices usando el color actual fijado por **setcolor** y el patrón y grosor de línea fijados por **setlinestyle**.

Las coordenadas de los vértices están en el arreglo vértice que tiene (**2 * n_puntos**) elementos ordenados: **x1**, **y1**, **x2**, **y2**, . . . **xn**, **yn**.

Dibuja de (**x**, **y**) a (**x2**, **y2**) de (**x2**, **y2**) a (**x3**, **y3**) . . . de (x_{n-1}, y_{n-1}) a (x_n, y_n). el primer punto tiene que repetirse al final de la lista para cerrar la figura.

RELLENO DE POLÍGONOS:

```
#include <graphics.h>
#include <conio.h>
void main (void) {
                    int driver = DETECT,modo;
                    int penta[ ] = {5,20,15,20,18,12,10,7,2,12,5,20};
                    initgraph( &driver, &modo, "c:\\tc20\\bin" );
                    cleardevice();
                    setcolor (10);
                      setlinestyle (DASHED_LINE, 0, THICK_WIDTH);
                       setfillstyle (XHATCH_FILL, getmaxcolor());
                          drawpoly(6, penta) ;
                    getch();
                    closegraph();
                   }
```

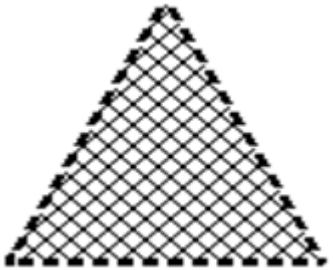


RELLENO DE POLÍGONOS:

fillpoly (**n_pts**, **p[ ]**);

Dibuja el perímetro del polígono definido por los **n_pts** puntos en el arreglo **p[ ]**, usando el color actual de dibujo y el estilo actual de líneas (patrón y grosor). Rellena el interior del polígono usando el estilo y color actual de relleno.

```
#include <graphics.h>
#include <conio.h>
void main (void) {
        int driver = DETECT,modo, tri[ ] = {0,100,  100,100,  50,0};
        initgraph( &driver, &modo, "c:\\tc20\\bin" );
        cleardevice();
        setcolor (10);
         setlinestyle (DASHED_LINE, 0, THICK_WIDTH);
          setfillstyle (XHATCH_FILL, getmaxcolor());
             fillpoly (3, tri);
        getch();
        closegraph();
       }
```



FUNCIÓN LINETO:

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).

```
#include <graphics.h>
#include <conio.h>
void main() {
int adap, modo ;
initgraph( &adap, &modo, "C:\\tc20\\bin" );
moveto( 20, 20 );
lineto( 40, 60 );
lineto( 80, 90 );
getch();
closegraph();
}
```

FUNCIÓN MOVEREL:

void 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.

```
#include <graphics.h>
#include <conio.h>
void main() {
int driver = DETECT,modo;
initgraph( &adap, &modo, "C:\\tc20\\bin" );
moveto( 20, 20 );
linerel( 20, 40 );
moverel( 50, 50 );
linerel( 40, 30 );
getch();
closegraph();
}
```

Elaborado por: Ing. Grevin Alexander Silva Lizano

FUNCIÓN LINEREL:

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 específica el número relativo de píxel para atravesar en la dirección horizontal.

El argumento dy específica el número relativo de píxeles 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.

Elaborado por: Ing. Grevin
Alexander Silva Lizano

```
#include <graphics.h>
#include <conio.h>
void main() {
int driver = EGA,modo = EGAHI;
initgraph( &driver, &modo, "C:\\tc20\\bin" );
moveto( 20, 20 );
linerel( 20, 40 );
linerel( 40, 30 );
getch();
closegraph();
}
```

FUNCIÓN GETX:

int 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.

```
#include <graphics.h>
#include <stdio.h>
void main() {
int adap, modo, x, y;
initgraph( &adap,&modo, "C:\\tc20\\BIN" );
moveto( 300, 150 );
x = getx();
y = gety();
closegraph();
printf (“Cursor gráfico\n\nX: %d\tY: %d\n", x, y);
}
```

Elaborado por: Ing. Grevin
Alexander Silva Lizano

FUNCIÓN GETY:

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.

```
#include <graphics.h>
#include <stdio.h>
void main() {
int adap, modo, x, y;
initgraph( &adap, &modo, “C:\\TC20\\BIN” );
moveto( 300, 150 );
x = getx();
y = gety();
printf (“Cursor gráfico\n\nX: %d\tY: %d\n", x, y);
closegraph();
getch();
}
```

Utilización de bar3d

```
/* Declaracion de Librerias a Utilizar*/
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <graphics.h>

/*Declaracion de funciones*/
void InicioModoGrafico(void);
void  main ()
{
int color=10, relleno=6;
/* Modo grafico*/
InicioModoGrafico();
setfillstyle(relleno,color);
bar3d(200,100,300,150,25,1);
/*Borrar pantalla presentacion*/
/*cleardevice(); closegraph();*/
getch();
restorecrtmode();
}

void InicioModoGrafico(void)
            {
/*Inicializando el modo gráfico de Turbo C*/
int adap=DETECT,modo;
initgraph(&adap,&modo,"C:\\tc20\\bin ");
            }
```

Otras funciones de la librería graphics.h.

Función clearviewport

void 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.

```
#include <graphics.h>
#include <conio.h>
void main() {
int a = EGA,b = EGAHI,color;
initgraph( &a, &b, "C:\\tc20\\BIN" );
setviewport( 150, 150, 350, 350, 0 );
for( color = 0; color<16; color++ ) {
circle( 100, 100, 60 );
getch();
setbkcolor( color );
clearviewport();
}
getch(); /* Pausa */
closegraph();
}

/******* Ejemplo de clearviewport()*******/
```

Elaborado por: Ing. Grevin
Alexander Silva Lizano

Función getarccords

void 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 sintaxis 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 **xstar**t 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.

Ejemplo: /** getarccoords*****/**

```
#include <graphics.h>
#include <conio.h>
void main() {
int driver = EGA;
int modo = EGAHI,radio;
struct arccoordstype info_arco;
initgraph( &driver, &modo, "C:\\tc20\\BIN" );
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();
getch();
}
```

Función getaspectratio

void 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 píxeles más altos que anchos. La única excepción es en los modos de VGA que produce píxeles cuadrados; es decir, x_proporcion = y_proporcion.

Ejemplo: /** getaspectratio*****/**

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
void main() {
int driver = EGA, modo = EGAHI;
int x_proporcion, y_proporcion;
initgraph( &driver, &modo, "C:\\tc20\\BIN" );
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 );
getch();
}
```

Función getdefaultpalette

struct palettetype *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.

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

Ejemplo: /****getdefaultpalette******/**

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
void main() {
int driver = EGA;
int modo = EGAHI;
struct palettetype *palette = NULL;
int i;
initgraph( &driver, &modo, "C:\\tc20\\BIN" );
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" );
getch();
}
```

Función getfillpattern

void getfillpattern(char *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.

Ejemplo: /********getfillpattern**********/**

```
#include <graphics.h>
#include <conio.h>
void main() {
int driver = EGA;
int modo = EGAHI;
char trama1[8] = { 0x33, 0xEE, 0x33, 0xEE, 0x33, 0xEE, 0x33, 0xEE };
char trama2[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };
initgraph( &driver, &modo, "C:\\tc20\\bin" );
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();
getch();
}
```

Función getpixel

unsigned 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.

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

```
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
void main() {
int driver = EGA, modo = EGAHI, x, y, color;
initgraph( &driver, &modo, “C:\\TC20\\BIN” );
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 );
getch();
}
```

/********Función getpixel**********/**

Función graphfreemem

void far_graphfreemem(void *ptr, unsigned tamanyo);

Esta función es usada por la librería gráfica para desad judicar 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.

La función _graphfreemem no retorna ningún valor.

Ejemplo:/********** **Función graphfreemem********/**

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

void far_graphfreemem( void *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_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 );
}
```

```
void main() {
int driver = EGA,modo = EGAHI;
initgraph( &driver, &modo,"C:\\TC20\\BIN" );
circle( 200, 100, 50 );
getch();
closegraph();
getch();
}
```

TAREA #3

Objetivo: Aplicar el paradigmas de programación procedural C en el ambiente gráfico.

- Esta tarea es para ser desarrollada individual.

- La originalidad y creatividad será fundamental.

- Realizar en el lenguaje Turbo C 2.0 un avatar de su persona y crear un ambiente dónde simule un día de actividad de su vida.

- Se entregará el 25 de septiembre a las 3:59 de la tarde en la plataforma EVA.