SEP SEIT DGIT

INSTITUTO TECNOLÓGICO DE NUEVO LAREDO

INGENIERÍA EN SISTEMAS COMPUTACIONALES

"Antología de Graficación en Lenguaje C++"

Por:
Ing. Bruno López Takeyas
Docente de Ing. en Sistemas Computacionales

CONTENIDO

1. CONCEPTOS BÁSICOS DE GRAFICACIÓN

Tal como un artista selecciona diversos medios para representar sus pinturas, los programadores, escogen un modo y formato especial para habilitar el monitor para graficar. Cada modo proporciona ciertas características como la resolución, número posible de colores, modo texto o modo gráfico y otros elementos donde cada modo requiere de cierto equipo (hardware).

1.1. Resolución

Las imágenes gráficas mostradas en un monitor de computadora se componen de pequeños puntos llamados píxeles, los cuales están distribuidos en la pantalla en filas; existe una cantidad específica de filas y cada fila tiene una cantidad específica de píxeles. La cantidad de píxeles usada en la pantalla se conoce como resolución. Cada modo gráfico tiene una resolución particular.

1.2. Inicializar el monitor en modo gráfico

Para habilitar el monitor en modo gráfico y utilizar sus píxeles, es necesario incluir el encabezado **#include <graphics.h>** que contiene las declaraciones y funciones relacionadas con graficación e inicializar el monitor en modo gráfico y utilizar sus píxeles con la función ***initgraph()***. Dicha función requiere las declaraciones mostradas en la Fig. 1.

```
int monitor=DETECT;  // Variable para detectar el tipo de monitor
int modo;            // Modo de operación del monitor
```

Fig. 1. Declaración de variables para detectar el tipo de monitor y habilitarlo en modo gráfico

También se puede declarar e inicializar con un tipo de monitor específico como lo presenta la Fig. 2.

```
int monitor=VGA;  // Variable para usar el monitor tipo VGA
int modo=VGAHI;  // Usar el monitor VGA a su maxima resolución
```

Fig. 2. Declaración de variables para habilitar un monitor VGA en modo gráfico

1.2.1. La función initgraph()

Una vez declaradas las variables monitor y modo que controlarán la resolución identificando el tipo de pantalla o monitor y su modo de operación respectivamente, se utiliza la función `initgraph()` para habilitar el monitor seleccionado en modo gráfico. La función `initgraph()` tiene 3 parámetros o argumentos:

1) La variable que identifica el monitor.
2) El modo de operación gráfico.
3) Subdirectorio que contiene los controladores de los monitores (archivos con extensión BGI) y los archivos con los tipos de letra (extensión CHR) como lo muestra la Fig.3.

```
int monitor=DETECT, modo;

initgraph(&monitor,&modo,"\\tc\\bgi");
```

Fig. 3. La función initgraph().

Si se desea usar el directorio actual por defecto, se utiliza la función `initgraph()` como lo indica la Fig. 4.

```
int monitor=DETECT, modo;

initgraph(&monitor,&modo,"");
```

Fig.4. La función initgraph() usando el subdirectorio actual por defecto

1.2.2. La función closegraph()

Para terminar de usar el monitor en modo gráfico y devolverlo a su modo de texto normal se usa la función ***closegraph()***.

1.3. Uso de coordenadas

Una vez que se inicializa el monitor en modo gráfico, las coordenadas tienen al píxel como unidad de medida. La función ***getmaxx()*** calcula la cantidad de píxeles por renglón y la función ***getmaxy()*** calcula la cantidad de renglones de la pantalla.

Las funciones de gráficos tienen como estándar el orden de manejo de coordenadas como columna, renglón; es decir, primero se anota la columna y después el renglón para posicionarse en dicha coordenada.

2. FUNCIONES DE GRAFICACIÓN

Sería muy difícil considerar todas las opciones posibles de todas las funciones de graficación; sin embargo, en esta antología se tratan los temas fundamentales para implementar este tipo de funciones. Básicamente mostraremos que antes de utilizar un color, un tipo de línea, de relleno, etc. es necesario definirlo.

2.1 Píxeles, líneas, figuras geométricas, colores y rellenos

En esta sección se presentan las funciones básicas de graficación, utilizadas para colocar pixeles, líneas, figuras geométricas, etc. en la pantalla.

2.1.1 La función putpixel()

Tal como se mencionó en la sección anterior, al habilitar el monitor en modo gráfico, la pantalla se divide en pequeños puntos llamados pixeles. La función `putpixel()` coloca un punto o píxel de un color específico en determinada coordenada de la pantalla. Esta función requiere 3 argumentos (Fig. 5):

1) Columna del punto (coordenada x).
2) Renglón del punto (coordenada y).
3) Color del píxel.

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

Fig.5.La función putpixel()

El código fuente del programa de la Fig. 6 muestra un ejemplo que inicializa el monitor en modo gráfico y coloca , en forma aleatoria, múltiples pixeles de colores en la pantalla hasta que se oprima cualquier tecla.

```
/*
Programa para inicializar el monitor en modo grafico

MiniTaller: Tecnicas avanzadas de programacion en Lenguaje C++
Instructor: M.C. Bruno Lopez Takeyas
*/

#include <graphics.h>  // Encabezado con declaraciones de
graficos
#include <conio.h>     // Para el uso de kbhit()
#include <stdlib.h>    // Para el uso de random()
#include <dos.h>       // Para el uso de delay

void main(void)
{
int monitor=VGA, modo=VGAHI; // Declaracion de tipo de monitor
y modo

initgraph(&monitor,&modo,"\\tc\\bgi");
// Inicializa el modo grafico indicando el monitor y modo
utilizado
// El subdirectorio \\tc\\bgi indica la ruta de localizacion de
los
// archivos *.BGI (monitores) y *.CHR (tipos de letras)

while(!kbhit())  // Mientras no se oprima cualquier tecla
{
putpixel(random(getmaxx()),random(getmaxy()),random(getmaxcolor
()));
// Coloca un pixel en la pantalla, seleccionando al azar la
columna
// (1..getmaxx), el renglon (1..getmaxy) y el color
(1..getmaxcolor)
delay(5);
}

closegraph(); // Termina el modo grafico (vuelve a su modo
normal)
}
```

Fig. 6.- Inicializar el monitor en modo gráfico y colocación de pixeles en la pantalla

2.1.2 La función line()

Esta función se utiliza para dibujar una línea entre 2 puntos. Para ello, la función requiere 4 parámetros que representan las coordenadas (en pixeles) de los dos puntos que se desea unir mediante una línea recta. La Fig. 7 muestra un ejemplo que une los puntos 50,100 y 300,200 (columna, renglón respectivamente).

```
line(50,100,300,200);
```

Fig.7. La función line().

2.1.3 La función setlinestyle()

Esta función se utiliza para determinar el tipo de línea o trazo que se desea. Se pueden utilizar trazos con línea continua, línea punteada, línea interrumpida, o un patrón de línea definido por el usuario. Esta función requiere 3 argumentos:

1) Tipo de línea: Puede ser SOLID_LINE, DOTTED_LINE, CENTER_LINE, DASHED_LINE o USERBIT_LINE.
2) Patrón: Este argumento regularmente es ignorado (excepto cuando se trata de un tipo de línea definido por el usuario).
3) Ancho de línea: Define la amplitud del trazo.

La Fig. 8 muestra un ejemplo que une los puntos 50,100 y 300,200 con una línea punteada.

```
setlinestyle(DOTTED_LINE,0, NORM_WIDTH);
line(50,100,300,200);
```

Fig. 8. La función setlinestyle().

2.1.4 La función circle()

Esta función dibuja un círculo y requiere 3 argumentos:

1) Coordenada de la columna del centro (en pixeles).
2) Coordenada del renglón del centro (en pixeles).
3) Radio del círculo (en pixeles).

La Fig. 9 dibuja un círculo cuyo centro se encuentra en el punto 300,150 y su radio es de 27 pixeles.

```
circle(300,150,27);
```

Fig. 9. La función circle().

Se pueden combinar funciones de graficación para obtener trazos específicos o determinados colores, sólo es cuestión de definir previamente el color o tipo de trazo deseado para después invocar la función de graficación deseada. El ejemplo de la Fig. 10 muestra el código necesario para dibujar un círculo con un trazo interrumpido. En este caso primeramente se define el tipo de línea (trazo) deseado y posteriormente se invoca la función `circle()` con los parámetros adecuados.

```
setlinestyle(DASHED_LINE,0, NORM_WIDTH);
circle(300,150,27);
```

Fig. 10. Ejemplo de un círculo con trazo interrumpido.

2.1.5 La función rectangle()

Esta función dibuja un rectángulo indicando las coordenadas de las esquinas superior izquierda e inferior derecha respectivamente. La Fig. 11 muestra un ejemplo de una función que dibuja un rectángulo desde el punto 50,100 hasta el punto 400,250.

```
rectangle(50,100,400,250);
```

Fig. 11. La función rectangle().

2.1.6 La función ellipse()

Se usa esta función para dibujar un arco elíptico o una elipse completa. Esta función requiere 6 argumentos:

1) Columna del centro de la elipse (coordenada x).
2) Renglón del centro de la elipse (coordenada y).
3) Ángulo inicial del arco.
4) Ángulo donde termina el arco.
5) Radio horizontal de la elipse.
6) Radio vertical de la elipse.

Se puede dibujar una elipse completa indicando el ángulo inicial como 0° (cero) y el ángulo final como 359° . La Fig. 12 dibuja una elipse completa con centro en 100,150, con radio horizontal de 25 y radio vertical de 40 pixeles.

```
ellipse(100,150,0,359,25,40);
```

Fig. 12. La función ellipse().

2.1.7 La función arc()

Se usa esta función para dibujar un arco circular. Esta función requiere 5 argumentos:

1) Columna del centro del arco (coordenada x).
2) Renglón del centro del arco (coordenada y).
3) Ángulo inicial del arco.
4) Ángulo donde termina el arco.

5) Radio.

Los puntos de inicio y final del arco se especifican por medio de sus ángulos medidos en el sentido de las manecillas del reloj; es decir, con 0° en las 3:00, 90° en las 12:00 y así sucesivamente. La Fig. 13 dibuja un arco de 30° con centro en 100,150 y radio 40 pixeles.

```
arc(100,150,0,29,40);
```

Fig. 13. La función arc().

2.1.8 La función setcolor()

Se utiliza esta función para definir el color de los trazos siguientes; es decir, antes de dibujar un trazo de un color específico, éste debe definirse. Esta función sólo tiene un argumento que representa el código del color deseado. P. ejem. BLACK, RED, BLUE, GREEN, YELLOW, etc. o bien su número entero correspondiente. La Fig. 14 muestra la tabla de colores y sus respectivos valores.

Constante	Valor
BLACK	0
BLUE	1
GREEN	2
CYAN	3
RED	4
MAGENTA	5
BROWN	6
LIGHTGRAY	7
DARKGRAY	8
LIGHTBLUE	9
LIGHTGREEN	10
LIGHTCYAN	11
LIGHTRED	12
LIGHTMAGENTA	13
YELLOW	14
WHITE	15

Fig. 14. Tabla de colores y sus valores.

La Fig. 15 muestra un ejemplo del uso de la función `setcolor()` donde se dibuja un circulo de color rojo y después un rectangulo de color azul.

```
setcolor(RED);
circle(300,150,27);

setcolor(BLUE);
rectangle(50,100,400,250);
```

Fig. 15. Uso de la función setcolor().

2.1.9 Definiendo el tipo de relleno de una figura con la función setfillstyle()

Si se desea rellenar una figura, es necesario definir previamente el patrón y color del relleno. La Fig. 16 muestra los patrones de relleno disponibles.

PATRÓN	VALOR	DESCRIPCIÓN
EMPTY_FILL	0	Color del fondo
SOLID_FILL	1	Relleno sólido con el color determinado
LINE_FILL	2	Relleno con línea (---)
LTSLASH_FILL	3	Relleno con /// líneas de ancho normal
SLASH_FILL	4	Relleno con **///** líneas
BKSLASH_FILL	5	Relleno con \\\ líneas
LTBKSLASH_FILL	6	Relleno con \\\ líneas de ancho normal
HATCH_FILL	7	Relleno de líneas cruzadas ligeras
XHATCH_FILL	8	Relleno de líneas cruzadas gruesas
INTERLEAVE_FILL	9	Relleno de líneas
WIDE_DOT_FILL	10	Relleno de puntos espaciados
CLOSE_DOT_FILL	11	Relleno de puntos cercanos
USER_FILL	12	Relleno definido por el usuario

Fig. 16. Patrones de relleno de la función setfillstyle()

Por ejemplo, si se desea definir el patrón de relleno de puntos cercanos de color rojo, se usa la función `setfillstyle()` como lo muestra la Fig. 17.

```
setfillstyle(CLOSE_DOT_FILL, RED);
```

Fig. 17. Selección del patrón de relleno CLOSE_DOT_FILL de color rojo (RED)

2.1.10 La función floodfill()

Una vez seleccionado el patrón de relleno mediante la función `setfillstyle()`, se procede a rellenar una figura usando la función `floodfill()`. Es muy importante resaltar que la figura que se desea rellenar esté completamente cerrada, ya que esta función contiene un algoritmo que busca el contorno de la figura y, en caso de encontrar una apertura, la función extralimitará la figura y también rellenará la parte externa de la misma. La función `floodfill()` requiere identificar un punto que se encuentre dentro del contorno de la figura y necesita 3 argumentos:

1) Coordenada de la columna del punto interno de la figura.
2) Coordenada del renglón del punto interno de la figura.
3) Color del borde de la figura.

El ejemplo mostrado en la Fig. 18 dibuja un circulo de color ROJO y lo rellena de color AZUL sólido.

```
setcolor(RED);
circle(300,150,27);

setfillstyle(SOLID_FILL,BLUE);
floodfill(300,150,RED);
```

Fig. 18. Uso de la función floodfill().

Nótese que la función `floodfill()` requiere como argumento el color del contorno de la figura que se desea rellenar.

3. APLICACIONES CON GRÁFICOS

A continuación se presentan dos ejemplos típicos con gráficos en los que se aplican las funciones previamente descritas.

3.1 Graficar la ecuación y = seno(x)

El código fuente del programa de la Fig. 19 muestra el comportamiento gráfico de la ecuación y = seno(x). Este programa muestra un ciclo de la curva senoidal con una amplitud de 150 pixeles. En él se incluye el encabezado `#include <math.h>` para aplicar la función matemática `sin(x)`, la cual calcula el seno de un ángulo expresado en **radianes** (todas las funciones trigonométricas en lenguaje C++ así lo requieren).

```
/* Programa para dibujar una curva senoidal
 Autor: Ing. Bruno Lopez Takeyas*/

#include <graphics.h> /*Para utilizar las funciones de
graficacion*/
#include <conio.h>
#include <stdlib.h>
#include <math.h> /*Para el calculo de la funcion Seno*/

void main(void)
{
  int monitor=DETECT,modo; /*Variables para el monitor*/
  const PI=3.1415927; /*Declaracion de la constante PI*/
  int angulo; /*Variable que controla el angulo (en grados) de la
curva*/
  int amplitud=150; /*Amplitud de la curva*/
  int columna=100; /*Coordenada de la columna donde inicia la
curva*/
  float y;
  float radianes; /*Variable para convertir los angulos a
radianes*/
  char *cadena; /*Variable auxiliar para desplegar mensajes
graficos*/

  initgraph(&monitor, &modo, ""); /*Inicializa el monitor en modo
grafico*/

  line(columna,getmaxy()/2,getmaxx()-10,getmaxy()/2);
  line(columna,50,columna,getmaxy()-50); /*Dibuja los ejes*/
```

```
  setcolor(CYAN); /*Establece el color celeste como
predeterminado*/
  settextstyle(SANS_SERIF_FONT,HORIZ_DIR,4);
  /*Establece el tipo de letra Sanserif*/
  outtextxy(120,10,"Graficacion de una Senoidal");
  /*Despliega este mensaje grafico*/

  setcolor(BLUE); /*Establece el color azul como predeterminado*/
  settextstyle(SMALL_FONT,HORIZ_DIR,4);
  /*Establece el tipo de letra Small*/

  outtextxy(getmaxx()-65,getmaxy()/2,"x (Grados)");
  outtextxy(40,50,"y=seno(x)"); /*Despliega estos mensajes*/

  for(angulo=0;angulo<=375;angulo++) /*Ciclo para los angulos de
la curva*/
  {
    radianes=float(angulo*PI/180.0); /*Convierte los angulos a
radianes*/

    y=amplitud*sin(radianes); /*La funcion sin(x) requiere los
angulos en
    radianes para calcular la amplitud de la curva*/

    putpixel(columna+angulo,getmaxy()/2-y,YELLOW);
    /*Coloca un punto en la pantalla*/

    if((angulo%30)==0) /*Coloca el valor del angulo cada 30
grados*/
    {
      itoa(angulo, cadena, 10);
      outtextxy(columna+angulo,getmaxy()/2,cadena);
    }
  }
  getch();
  closegraph(); /*Cierra el modo grafico del monitor*/
  return;
}
```

Fig. 19. Código fuente para graficar la ecuación y = seno(x).

3.2 Programa fuente que dibuja los escudos del Tec y de ISC

El código fuente del programa de la Fig. 20 muestra la forma de dibujar el escudo del Instituto Tecnológico de Nuevo Laredo y el escudo de la carrera de Ingeniería en Sistemas Computacionales (ISC), apoyándose en el encabezado de la Fig. 21.

```
/*
Programa para graficar los escudos del Tec y de ISC
 MiniTaller: Tecnicas avanzadas de programacion en Lenguaje C++
 Instructor: M.C. Bruno Lopez Takeyas
*/

#include <graphics.h>  // Encabezado con declaraciones de
graficos
#include <conio.h>
#include <stdio.h>
#include <tec.h> // Encabezado desarrollado por el programador
con el
           // codigo fuente de los escudos del Tec y de ISC

void main(void)
{
  int monitor=DETECT, modo; // Declaracion de tipo de monitor y
modo
                   // Automaticamente detecta el tipo de monitor

  initgraph(&monitor,&modo,"\\tc\\bgi");
// Inicializa el modo grafico indicando el monitor y modo
utilizado
// El subdirectorio \\tc\\bgi indica la ruta de localizacion de
los
// archivos *.BGI (monitores) y *.CHR (tipos de letras)

  TEC(100,200); //Escudo del Tec en columna=100 y renglon=200
  ISC(getmaxx()-100,200); //Escudo de ISC en columna=540 y
renglon=200

  getch();
  closegraph(); // Termina el modo grafico (vuelve a su modo
normal)
  return;
}
```

Fig. 20.- Programa fuente con los escudos del Tec y de ISC.

```
/* LIBRERIA ..............................................
TEC.H
   Libreria de los logos del ITNL y de ISC
*/

void ISC(int x,int y)
{
     setfillstyle(SOLID_FILL,WHITE);
     ellipse(x+10,y-40,7,182,35,14);        /* cresta */
     ellipse(x+45,y-4,270,90,10,37);        /* nuca   */
     line(x+32+12,y+34,x+38+12,y+42);
     ellipse(x+50,y+56,90,170,55,15);       /* cuello */
     ellipse(x-8,y+47,280,95,3,7);          /* garganta */
     arc(x-18,y+35,60,330,9);               /* barbilla */
     line(x-18,y+35-9,x-18,y+35+9);
     line(x-14,y+35-7,x-14,y+35+7);
     line(x-5,y+42,x+7,y+37);
     line(x+7,y+37,x+10,y+40);
     line(x+10,y+40,x+11,y+45);             /* tarjeta  */
     circle(x+21,y-23,12);          floodfill(x+21,y-23,WHITE);
     line(x-18,y+42,x+38,y-48);
     arc(x+40,y-9,40,275,3);
     line(x+42,y-12,x+55,y-2);
     line(x+40,y-6,x+54,y+5);
     line(x+38,y-10,x-17,y-46);
     arc(x-25,y+25-2,90,270,3);               /* labio inferior
*/
     arc(x-25,y+19-3,90,270,3);               /* labio superior
*/
     ellipse(x-25,y-32,90,270,2,8);
     line(x-25,y+25-13,x-32,y+20-13);
     line(x-32,y+20-13,x-25,y+20-25);
     ellipse(x-25,y-15,90,270,2,8);
     line(x+33-4,y+46-4,x+33,y+32);
     line(x+38,y+46-6,x+38,y+30);
     arc(x+34,y+28,265,5,4);
     arc(x-21,y-21,45,280,3);
     arc(x-15,y-16,280,45,3);
     line(x-18,y-23,x-13,y-18);
     line(x-20,y-17,x-14,y-12);
     floodfill(x-17,y-20,WHITE);
     line(x-17,y-27,x-13,y-23);
     line(x-13,y-33,x-9,y-29);
     line(x-13,y-29,x+5,y-54);
     line(x-17,y-27,x-13,y-33);
     line(x-13,y-23,x-9,y-29);
     floodfill(x-14,y-25,WHITE);
     line(x-16-5-2,y-17+3-2,x-12-3,y-12+3);
     line(x-16-5-2,y-17+3-2,x-26,y-12);
     line(x-19,y-12,x-25,y-8);
```

```
     floodfill(x-23,y-11,WHITE);
     arc(x+30,y+22,235,45,8);
     line(x+30-7+4,y+22+7+1-2,x+30-2+7,y+22-7+2);
     line(x+30,y+22,x-26,y-25);
     line(x-24,y-24,x+54,y-24);
     line(x+30,y+22,x+30,y-50);
     rectangle(x-2,y+28,x+17,y+34);
     rectangle(x+1,y+19,x+14,y+28);
     rectangle(x+3,y+21,x+12,y+26);
     rectangle(x-20,y-14+4,x-14,y-6+4);
     rectangle(x-14,y-14+8+2,x-11,y-6-4+2);
     line(x-20,y-14+4,x-20,y-6+4);
     rectangle(x-19,y-6+4,x-15,y-6+4+2);
     ellipse(x-21,y-14+8,90,270,1,4);
     settextstyle(SMALL_FONT,HORIZ_DIR,5);
     outtextxy(x-14,y+56,"SISTEMAS");
     outtextxy(x-42,y+69,"COMPUTACIONALES");
     outtextxy(x-35,y-56,"I");
     outtextxy(x-25,y-61,"N");
     outtextxy(x-15,y-64,"G");
     outtextxy(x-5,y-67,"E");
     outtextxy(x+5,y-70,"N");
     outtextxy(x+17,y-70,"I");
     outtextxy(x+25,y-67,"E");
     outtextxy(x+35,y-64,"R");
     outtextxy(x+45,y-61,"I");
     outtextxy(x+53,y-56,"A");
     return;
}

void TEC(int x,int y)
{
     setcolor(WHITE);
     circle(x,y,60);
     setcolor(GREEN);
     circle(x,y,64);
     circle(x,y,63);
     setcolor(WHITE);
/* libro  */
     line(x-54,y+17,x-17,y+17);
     line(x-54,y+17,x-43,y-30);
     line(x-51,y+14,x-16,y+14);
     line(x-51,y+14,x-40,y-32);
     line(x+1,y-28,x+1,y-18);
/* fabrica  */
     line(x+24,y-40,x+24,y-49);
     line(x+24,y-49,x+13,y-49);
     line(x+13,y-49,x+13,y-55);
     line(x+13,y-55,x+2,y-49);
     line(x+2,y-49,x+2,y-55);
     line(x+2,y-55,x-10,y-49);
```

```
     line(x-10,y-49,x-10,y-55);
     line(x-10,y-55,x-20,y-48);
     line(x-20,y-48,x-20,y-32);
     line(x-20,y-38,x+24,y-38);
     line(x-15,y-40,x+24,y-40);
     line(x-15,y-42,x+24,y-42);
     line(x-15,y-45,x+24,y-45);
     line(x-10,y-45,x-10,y-40);
     line(x-5,y-45,x-5,y-40);
     line(x,y-45,x,y-40);
     line(x+5,y-45,x+5,y-40);
     line(x+10,y-45,x+10,y-40);
     line(x+15,y-45,x+15,y-40);
     line(x+20,y-45,x+20,y-40);
     line(x+8,y-52,x+8,y-58);
     line(x+6,y-51,x+6,y-58);
     line(x+6,y-58,x+8,y-58);
     line(x-4,y-52,x-4,y-58);
     line(x-6,y-51,x-6,y-58);
     line(x-4,y-58,x-6,y-58);
     line(x-15,y-52,x-15,y-58);
     line(x-17,y-51,x-17,y-58);
     line(x-15,y-58,x-17,y-58);
/* rayo    */
     setcolor(RED);
     setfillstyle(SOLID_FILL,RED);
     line(x+25,y-40,x+10,y-16);
     line(x+25,y-40,x+11,y-23);
     line(x+10,y-16,x+9,y-19);
     line(x+11,y-23,x+9,y-28);
     line(x+9,y-19,x-13,y+21);
     line(x+9,y-28,x-12,y+12);
     line(x-13,y+21,x-14,y+18);
     line(x-12,y+12,x-14,y+9);
     line(x-14,y+18,x-25,y+40);
     line(x-14,y+9,x-25,y+40);
     floodfill(x-4,y,RED);
/*  engrane orilla externa  */
     setcolor(WHITE);
     line(x-25,y+40,x-20,y+43);
     line(x-20,y+43,x-15,y+41);
     line(x-15,y+41,x-6,y+43);
     line(x-6,y+43,x-4,y+47);
     line(x-4,y+47,x+4,y+48);
     line(x+4,y+48,x+7,y+44);
     line(x+7,y+44,x+13,y+43);
     line(x+13,y+43,x+18,y+45);
     line(x+18,y+45,x+25,y+42);
     line(x+25,y+42,x+25,y+40);
     line(x+25,y+40,x+27,y+34);
     line(x+26,y+36,x+34,y+32);
```

```
     line(x+34,y+32,x+39,y+32);
     line(x+39,y+32,x+45,y+23);
     line(x+45,y+23,x+42,y+13);
     line(x+42,y+14,x+44,y+9);
     line(x+44,y+9,x+48,y+6);
     line(x+48,y+6,x+48,y-3);
     line(x+48,y-3,x+44,y-7);
     line(x+44,y-7,x+44,y-13);
     line(x+44,y-13,x+45,y-17);
     line(x+45,y-17,x+40,y-27);
     line(x+40,y-27,x+33,y-26);
     line(x+33,y-26,x+29,y-32);
     line(x+31,y-30,x+31,y-36);
     line(x+31,y-36,x+25,y-40);
/* engrane orilla interna    */
     line(x-25,y+38,x-20,y+40);
     line(x-20,y+40,x-15,y+38);
     line(x-15,y+38,x-5,y+41);
     line(x-5,y+41,x-3,y+45);
     line(x-3,y+45,x+5,y+45);
     line(x+5,y+45,x+8,y+42);
     line(x+8,y+42,x+13,y+41);
     line(x+13,y+41,x+18,y+43);
     line(x+18,y+43,x+25,y+40);
     line(x+25,y+40,x+25,y+43);
     line(x+25,y+40,x+27,y+34);
     line(x+27,y+34,x+33,y+29);
     line(x+33,y+29,x+38,y+29);
     line(x+38,y+29,x+44,y+19);
     line(x+42,y+14,x+44,y+8);
     line(x+44,y+8,x+48,y+2);
     line(x+43,y-7,x+43,y-13);
     line(x+43,y-13,x+44,y-17);
/* semicirculos centrales   */
     arc(x,y,243,63,5);
     arc(x,y,240,57,20);
     ellipse(x,y+18,230,10,12,4);
     arc(x,y,275,25,10);
     arc(x,y,275,25,15);
/*  atomo   */
     ellipse(x-25,y-9,0,360,7,15);
     ellipse(x-25,y-9,0,360,15,7);
     arc(x-34,y-3,120,355,5);
     arc(x-16,y-18,320,120,5);
     line(x-37,y-7,x-19,y-23);
     line(x-30,y,x-12,y-16);
     arc(x-31,y-19,65,230,5);
     arc(x-15,y-4,230,80,5);
     line(x-20,y,x-35,y-17);
     line(x-13,y-9,x-28,y-22);
/*  hoja del libro     */
```

```
     ellipse(x-33,y+15,0,180,17,3);
     ellipse(x-12,y-30,0,180,12,4);
     ellipse(x-12-18,y-34,180,0,10,4);
     ellipse(x+18,y-30,90,180,17,4);
/* orificios del engrane */
     arc(x-7,y+26,90,295,4);
     arc(x+10,y+22,285,52,6);      arc(x+10,y+22,285,52,5);
     line(x-5,y+31,x+10,y+29);
     ellipse(x+25,y-7,220,90,7,17);
     arc(x+24,y-15,90,180,9);
     arc(x+24,y-15,90,180,8);
     return;
}
```

Fig. 21.- Encabezado con los detalles de los escudos

3.3 Archivar y cargar una imagen

Igual que se puede almacenar cualquier variable con un tipo de dato en particular como enteros, reales, caracteres, cadenas, registros, o cualquier estructura de datos, también es posible almacenar en un archivo una imagen que se despliega en la pantalla. En esta sección se presentan las funciones necesarias para el manejo de imágenes con la finalidad de grabarlas en un archivo.

3.3.1 Declaración de variables necesarias

Cuando se dibuja una imagen en la pantalla, ésta puede ser almacenada en una variable de memoria para manipularse completamente. Esto se logra mediante un apuntador de tipo char far, el cual puede almacenar en memoria una imagen de hasta 1MB. Esto se logra con las declaraciones de variables mostradas en la Fig. 22.

Declaración	Uso
long tamano;	Variable de tipo entero largo para calcular el tamaño en bytes de la imagen que se desea archivar.
char far *imagen;	Apuntador para capturar en memoria la imagen (puede ser direccionada hasta 1 MB)

Fig. 22. Declaración de variables necesarias para cargar una imagen en memoria.

3.3.2 ¿Cómo cargar una imagen en memoria?

Para almacenar una imagen previamente dibujada en la pantalla, debe calcularse su tamaño en bytes, reservar espacio de memoria para posteriormente "cargarla" en una variable de memoria mediante las declaraciones de la Fig. 22. Esto se logra con las funciones que se presentan a continuación.

3.3.2.1 La función imagesize()

Se utiliza esta función para calcular el espacio (en bytes) ocupado por una imagen. Requiere de 4 argumentos que representan las coordenadas de las esquinas superior-izquierda e inferior-derecha respectivamente y devuelve un valor que representa su tamaño, el cual debe almacenarse en una variable de tipo entero largo (Fig. 23).

3.3.2.2 La función farmalloc()

Esta función reserva espacio en memoria para almacenar la imagen cuyo bloque posiblemente exceda 64k (según el tamaño calculado por la función `imagesize`) y requiere incluir el encabezado `#include <alloc.h>` (Fig. 23).

3.3.2.3 La función getimage()

Se usa esta función para almacenar en un apuntador de tipo `char far` (Fig. 22) la imagen comprendida entre los puntos indicados por las coordenadas de sus esquinas superior-izquierda e inferior-derecha (Fig. 23).

Función	***Uso***
tamano=(long int) imagesize(x1,y1,x2,y2)	Calcula el espacio necesario para capturar en memoria la imagen comprendida entre las esquinas (x1,y1) a (x2,y2)
imagen=(char far *) farmalloc(tamano)	Reserva el espacio de memoria indicado por ***tamano***
getimage(x1,y1,x2,y2,imagen)	Captura en el apuntador ***imagen*** el dibujo comprendido entre (x1,y1) y (x2,y2)

Fig. 23 .- Funciones necesarias para archivar una imagen

3.3.3. ¿Cómo archivar una imagen creada por el usuario?

Para almacenar una imagen previamente dibujada en la pantalla, es necesario capturar en memoria la imagen que se desea grabar usando un apuntador, apoyándose en las declaraciones de la Fig. 22 y las funciones mostradas en la Fig. 23. Para manipular el archivo se declara el alias correspondiente como se muestra en la Fig. 24. Una vez capturada en memoria la imagen, se abre un archivo binario con un nombre específico, para proceder a escribir el apuntador con la imagen capturada (Fig. 25).

Declaración	Uso
FILE *alias	Declaración del alias para el archivo.

Fig. 24 .- Declaración del alias del archivo

Función de manejo de archivos	Uso
Alias=fopen("IMAGEN.IMG","wb")	Crea el archivo binario "IMAGEN.IMG"
fwrite(imagen,sizeof(imagen),1,alias)	Graba la imagen capturada en el archivo
fclose(alias)	Cierra el archivo

Fig. 25 .- Declaraciones necesarias para archivar una imagen

3.3.4 Ejemplo de aplicación

El programa de la Fig. 26 muestra el código completo de un ejemplo que dibuja en la pantalla el escudo del Instituto Tecnológico de Nuevo Laredo (incluido en el encabezado TEC.H) para posteriormente grabarlo en un archivo.

```
/*
Programa para graficar el escudo del Tec, grabarlo en un archivo
para
cargarlo y desplegarlo posteriormente

 MiniTaller: Tecnicas avanzadas de programacion en Lenguaje C++
 Instructor: M.C. Bruno Lopez Takeyas
*/

#include <graphics.h>  // Encabezado con declaraciones de
graficos
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <alloc.h> // Para usar la funcion farmalloc
#include <string.h>
#include <tec.h> // Encabezado desarrollado por el programador
con el
           // codigo fuente de los escudos del Tec y de ISC

void main(void)
{
  int monitor=DETECT, modo; // Declaracion de tipo de monitor y
modo
                    // Automaticamente detecta el tipo de monitor
  char *archivo; // Para capturar el nombre del archivo
  long tamano; // Variable para calcular el tama¤o en bytes de la
imagen que
             // se desea archivar
```

```
  char far *imagen; // Variable para capturar en memoria la
imagen que se desea
           // archivar
  FILE *alias_archivo;
  int columna,renglon;


  initgraph(&monitor,&modo,"\\tc\\bgi");
// Inicializa el modo grafico indicando el monitor y modo
utilizado
// El subdirectorio \\tc\\bgi indica la ruta de localizacion de
los
// archivos *.BGI (monitores) y *.CHR (tipos de letras)


  TEC(320,200); //Escudo del Tec en columna=320 y renglon=200

  setlinestyle(DOTTED_LINE,1,1);
  rectangle(250,130,390,270);

  gotoxy(1,19); cout << "Anote el nombre del archivo (incluyendo
la extension): ";
  gets(archivo);
  // Capturar el nombre del archivo

  tamano = (long int)imagesize(250,130,390,270); // Calcula el
tama¤o de la
  // imagen que se desea archivar

  imagen = (char far *) farmalloc(tamano); //Reserva la memoria
suficiente

  getimage(250,130,390,270,imagen); // Cargar en memoria la
imagen que
  // contiene el cuadro de las coordenadas indicadas

  alias_archivo=fopen(archivo,"wb"); // Crear el archivo con el
nombre capturado
  fwrite(imagen,1,tamano,alias_archivo); // Graba en el archivo
la imagen
  // cargada en memoria
  fcloseall(); // Cierra el archivo

  cout << "\n\ntama¤o=" << tamano << " bytes";
  cout << "\nOprima cualquier tecla para limpiar la pantalla y
cargar la imagen";
  getch();

  clearviewport(); // Limpia la pantalla en modo grafico
  alias_archivo=fopen(archivo,"rb"); // Abre el archivo en modo
de solo lectura
```

```
  fread(imagen,1,tamano,alias_archivo); // Lee desde el archivo
la imagen
  fcloseall(); // Cierra el archivo

  cout << "\nAnote las coordenadas donde desea desplegar la
imagen ...";
  cout << "\n\nColumna="; cin >> columna;
  cout << "\n\nRenglon="; cin >> renglon;

  putimage(columna,renglon,imagen,1); // Coloca en pantalla la
imagen
  // cargada del archivo en las coordenadas especificadas
  getch();
  closegraph(); // Termina el modo grafico (vuelve a su modo
normal)
  return;
}
```

Fig. 26.- Programa para archivar una imagen.

4. CONCLUSIONES

Aunque existen muchas otras funciones de graficación, esta antología presenta los conceptos y funciones básicas para iniciar la codificación de programas en C++ que permitan utilizar el monitor en modo gráfico. Aquí se muestran las operaciones fundamentales de graficación y se presentan ejemplos representativos, tal como el código fuente para graficar una ecuación senoidal y el código fuente para dibujar los escudos del Instituto Tecnológico de Nuevo Laredo y de la carrera de Ing. en Sistemas Computacionales. Además se explica la forma de cargar una imagen en memoria para posteriormente grabarla en un archivo.

Todos los programas que se muestran pueden obtenerse en el sitio http://www.itnuevolaredo.edu.mx/takeyas o bien solicitarse al autor escribiendo un correo electrónico a takeyas@itnuevolaredo.edu.mx.

5. BIBLIOGRAFÍA

- Barkakati Nabajyoti. **"The Waite Group´s. Turbo C Bible".** Howard W. Sams & Company. Estados Unidos. 1990.

- García Badell, J. Javier. **"Turbo C. Programación en manejo de archivos"**. Macrobit.

- Deitel y Deitel. **"C++ Cómo programar"**. Segunda edición. Pearson-Prentice Hall. Estados Unidos. 1999.

- Lafore, Robert. **"The Waite Group´s. Turbo C. Programming for the PC"**. Revised Edition. Howard W. Sams & Company. Estados Unidos. 1990.

- López Takeyas, Bruno. **"Minitaller: Técnicas avanzadas de programación en lenguaje C++".** Instituto Tecnológico de Nuevo Laredo, Tam. México. 2003.

- Schildt, Herbert. **"Turbo C. Programación avanzada"**. Segunda edición, McGraw Hill. Estados Unidos. 1990.

- Staugaard, Andrew. **"Técnicas estructuradas y orientadas a objetos. Una introducción utilizando C++"**. Segunda edición. Prentice Hall. Estados Unidos. 1998.