GRAFICACION EN LENGUAJE "C".

Palabras previas.

Este material está orientado a estudiantes de ciencias exactas, puesto que la mayoría de sus ejemplos versan sobre funciones matemáticas tales como circunferencias, exponenciales, parábolas, etc.

He tratado de seguir un orden coherente para el aprendizaje, comenzando con las operaciones básicas de impresión de puntos en pantalla y avanzando progresivamente a disposiciones complejas de elementos gráficos tales como: ejes desfasados, funciones definidas por tramos, síntesis de diversos tipos de ondas, funciones planas notables, animación, escalas angulares y resolución de sistemas complejos de ecuaciones utilizando metodología gráfica.

Recomiendo muy especialmente en toda resolución de problemas gráficos, realizar *SIEMPRE* un esquema previo en lápiz y papel con todas las acotaciones necesarias y manteniendo lo más estrechamente posible las proporciones de las figuras. Alguien dijo sabiamente que un problema bien planteado ya lleva en sí la solución.

He agregado una carpeta adicional con todos los ejecutables de los ejemplos de este material a fin de facilitar la visualización dinámica de los mismos.

Finalmente deseo a cada lector un buen aprovechamiento de todos los ejemplos y enseñanzas de esta humilde obra, y recordarle que sólo la práctica intensa hace al maestro, ¡así que a no desanimarse y a trabajar con verdadero entusiasmo!

¡ Buena suerte ¡

Ing. Juan Manuel Conti

Indice de contenidos.

CAPITULO 1

Modo gráfico **Pág. 4**
La resolución gráfica **Pág. 4**
La ocupación de memoria **Pág. 5**
Graficando en "C" **Pág. 6**
Inicializar el modo gráfico **Pág. 7**
Final de operaciones gráficas **Pág. 9**
Sistema de ejes de pantalla **Pág. 10**
Trazado de puntos luminosos **Pág. 11**
Casos un poco más complicados **Pág. 15**
Casquete circular **Pág. 18**
Un verdadero acertijo **Pág. 22**

CAPITULO 2

Graficación de funciones matemáticas **Pág. 24**
Calibración de ejes **Pág. 26**
Graficación de funciones senoidales **Pág. 29**
Conjunto de ondas senoidales **Pág. 32**
Ejes desfasados **Pág. 36**
Funciones exponenciales **Pág. 40**
Otro caso de ejes desplazados **Pág. 43**

CAPITULO 3

Funciones definidas por tramos **Pág. 47**

CAPITULO 4

Síntesis de onda cuadrada **Pág. 61**
Figuras de Lissajouss **Pág. 64**
Lemníscata **Pág. 69**
Folio de Descartes **Pág. 72**
Involuta de una circunferencia **Pág. 75**

CAPITULO 5 : Animación

Borrado de una figura **Pág. 79**
Generación de la Involuta **Pág. 86**
Figuras de Lissajouss móvil **Pág. 89**
Generación de una Cardioide **Pág. 90**

CAPITULO 6 : Método gráfico para resoluciones matem.

Sistema circunferencia – exponencial **Pág. 95**
Ráiz cuadrada **Pág. 100**

CAPITULO 7 : Escalas angulares.

Escalas graduadas **Pág. 104**
Voltímetro de aguja **Pág. 107**

CAPITULO 8 : Funciones periódicas cíclicas

Arcos de circunferencias **Pág. 113**
Onda Diente de Sierra **Pág. 116**
Onda Trapecial **Pág. 119**
Onda Parabólica **Pág. 122**

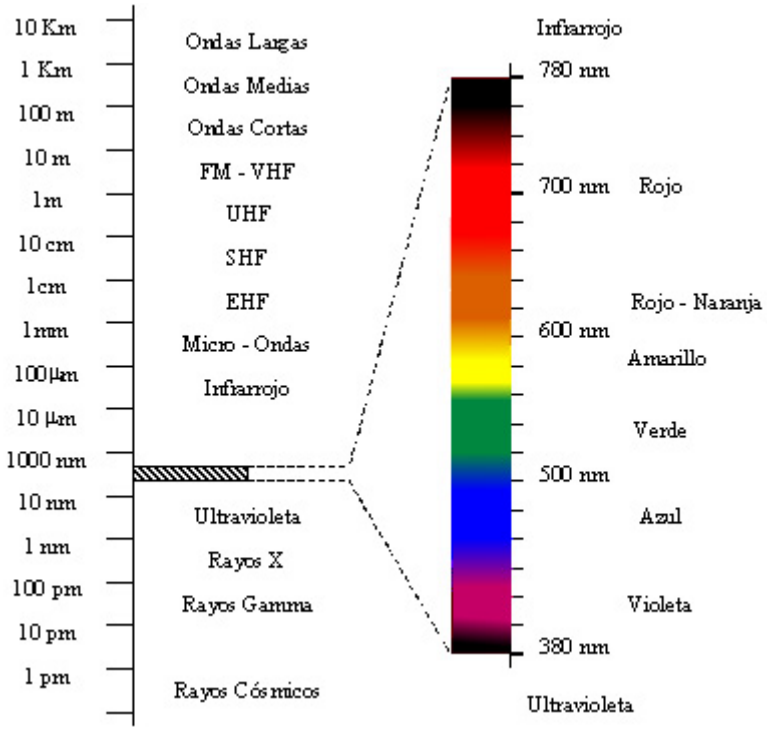
CAPÍTULO 1 - *El modo gráfico*.

Es la capacidad que tiene el micro procesador de transformar al monitor en una matriz de puntos luminosos llamados "píxeles", los cuales pueden ser ubicados en cualquier lugar físico de la pantalla. La pregunta es: ¿Cuántos puntos luminosos consecutivos pueden trazarse en forma horizontal y en forma vertical?..¿Y de cuántos colores diferentes?

Para responder a estas preguntas aparecerán conceptos importantes en el arte de la graficación. Un píxel ocupa no solamente un lugar físico en el monitor, sino que también ocupa memoria, y en los modos actuales de alta definición, suele ser bastante. Los monitores monocromáticos actualmente han pasado a la historia y la tendencia es trabajar con equipos cada vez de mayor calidad.

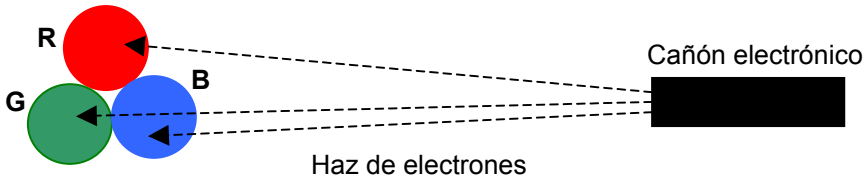
El sistema electrónico que "fabrica" el color está compuesto por dos elementos básicos: porciones microscópicas de material fotoemisor, y un "cañón electrónico" que emite un haz de electrones de alta velocidad. Los electrones al chocar contra dichas partículas generan luz, así se sencillo. Sin embargo avancemos un paso más: la ciencia ha establecido que todos los colores del espectro visible pueden ser obtenidos a partir de tres colores básicos denominados colores **PRIMARIOS**:

RED – GREEN – BLUE (Rojo – Verde – Azul) mezclados en distintas proporciones.



Si tuviésemos que hacer una analogía podríamos pensar en tres tarros de pintura de los cuales tomamos distintas cantidades de las mismas y las mezclamos en otro recipiente. Es más que eso, dentro de cada gama podemos tener distintas tonalidades. Por ejemplo podemos disponer de 16 tonos distintos de Rojos, al igual que de Verde y Azul, y recién entonces mezclar variando las proporciones. Esto nos da una idea de la cantidad impresionante de colores que pueden obtenerse.

Volviendo a nuestro monitor, el sistema utilizado para generar los puntos luminosos se denomina precisamente Sistema **RGB** y está compuesto por "ternas" de material luminiscente sobre los que impacta el haz de electrones. Toda la pantalla está rellena de estas ternas RGB y entonces uno se pregunta ¿si deseo obtener una tonalidad pareja de verde en toda la pantalla, cómo se logra esto? Obviamente existe una sola respuesta: el haz de electrones deberá impactar únicamente en el pigmento Verde de cada terna. Esto implica una tecnología de altísima precisión, ya que el haz debe impactar en un punto, apagarse, moverse al próximo encenderse e impactarlo, y de nuevo a los siguientes hasta completar la pantalla. Normalmente se trabaja con un cañón triple con un haz para cada color.



Ahora bien, cuando se ha "encendido" el último píxel de la pantalla se supone que todos los restantes están "apagados" (sin haz que los excite) ¿Entonces cómo continuamos viéndolos brillantes? Por algo que se llama "persistencia visual". El tiempo de "refrescado" del haz debe ser siempre inferior al de dicha persistencia para que nuestros ojos lo perciban como una imagen estable y sin "Flicker" o sensación de parpadeo.

¿Y las distintas tonalidades de cada color primario, por ejemplo un rojo más intenso o más pálido? Ello se obtiene con la intensidad del haz de electrones, otra de las virtudes de la complejidad electrónica.

La resolución gráfica.

Está asociada directamente al tamaño del píxel. Si una misma pantalla deber rellenarse con una cantidad mayor de píxeles, es obvio que éstos deberán tener un tamaño más pequeño. Otra vez el hardware es el que manda: podremos variar la definición del píxel solamente hasta el valor máximo permitido por la tecnología del monitor. A modo ilustrativo, un monitor **SyncMaster 753DFX** de pantalla plana, posee las siguientes resoluciones:

800 x 600
1024 x 768
1280 x 1024

Si exigimos una definición mayor que 1280 x 1024 el monitor no será capaz de responder a esta exigencia y sólo trabajará en su precisión máxima.

La ocupación de memoria.

La cantidad máxima de colores que puede conferirse a un píxel (que también es un parámetro tecnológico) la da una vez más el fabricante del monitor. Para el mismo monitor del ejemplo anterior, la "**Calidad de los colores**" puede variar entre 16 y 32 bits (por color).

Ello quiere decir que cada color primario (**RGB**) puede tomar entre 2^{16} y 2^{32} tonalidades diferentes. De esta manera podremos disponer de un máximo de 4.294 millones de tonos de **ROJO**, otros tantos de **VERDE** y otros de **AZUL** que a su vez se combinarán para dar los restantes colores del espectro.

Llevado al terreno informático implica que el código de colores para cada píxel necesita de los tres componentes (RGB), y cada uno de ellos, si estamos trabajando con 32 bits por color, requiere de 12 bytes de memoria (4 por cada color). Supongamos entonces que tenemos una imagen de 800 x 600 ¿Qué cantidad de memoria ocuparía si estamos trabajando con 32 bits por color?:

800 x 600 x 12 = 5.760.000 bytes, aproximadamente 5.5 Mb

Razonando un poco sobre lo visto hasta ahora, llegamos a la fácil conclusión de que la **calidad**, como todo en la vida, tiene un precio: un monitor de alta tecnología y una cantidad muy grande de memoria RAM. Y además nos queda el mayor insumo de tiempo que requiere procesar bloques grandes de memoria, sobre todo cuando se trabaja con animación (aquí la velocidad del microprocesador es fundamental). Ello es la razón por la cual muchos juegos de video trabajan en modos de baja resolución, con lo cual si bien se pierde calidad, en cambio se gana velocidad.

Planteados todos estos conceptos básicos ya podemos encarar nuestro propósito de graficar utilizando lenguaje "C".

Graficando en C.

No debe perderse de vista que "C" es un lenguaje de propósitos generales y no un editor gráfico como Corel o Photoshop, razón por la cual sólo nos limitaremos a los aspectos de la graficación más fundamentales como trazado de todo tipo de funciones matemáticas, generación de texto artístico y algo de animación en el plano.

BorlandC proporciona una librería especial para poder graficar: <graphics.h> que posee una cantidad realmente grande de funciones de trazado que incluyen desde un simple píxel en pantalla hasta diagrama de barras, polígonos, elipses, redefinición de la paleta de colores, etc.

Por lo tanto el primer paso para graficar en C, es declarar en la cabecera la librería anterior:

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

junto con todas las otras que necesitemos utilizar.

Inicializar el modo gráfico.

En un comienzo dijimos que el modo gráfico era una habilidad del microprocesador, pero también es necesario indicarle al micro que ejerza dicha habilidad. Para ello utilizaremos dos funciones especiales:

void detectgraph(&Gd,&Gm);
void initgraph(&Gd,&Gm,PathToDriver);

La primera de ellas: detectgraph( ) tiene por finalidad detectar la tarjeta gráfica y determinar el modo de mayor resolución de la misma, que son devueltos a través de sus parámetros Gd y Gm (que son de tipo int).

Conviene aclarar aquí que todo lo que sea salida al monitor es manejado por un hardware especial denominado **controlador gráfico** o simplemente tarjeta controladora, que actualmente se halla integrada al mother. Esta tarjeta ha sufrido bastante evolución a través del tiempo teniendo sus inicios allá por los ’80 con las viejas controladoras como la **CGA** de IBM que manejaban muy pocos colores y poseían un desagradable efecto de “nevado” en la pantalla, efecto que fuera corregido posteriormente con su sucesora la tarjeta **EGA** y **EGA64** que eliminaron el efecto de nevado y aumentaron considerablemente la definición de pantalla como así también la cantidad de colores. Luego apareció la tarjeta **VGA** con un muy buen desempeño gráfico. Entre todas ellas no debemos ser crueles y olvidar a la tan popular tarjeta monocromática **HERCULES** que poseía una definición muy alta para su época, y que tuviera muchísimo éxito.

Cabe destacar que la controladora gráfica era un parámetro tan importante, que entrada dentro del pliego de especificaciones cuando se adquiría una equipo **PC,** y normalmente consistía en una tarjeta separada del mother y enchufable en las ranuras de expansión. Incluso se establecía la cantidad de memoria **RAM** con que debían venir provistas (por ej. con 1, 2 ó 4 Mb).

Actualmente se utiliza la tarjeta **SVGA**, que, como dijimos al comienzo ya viene totalmente integrada al mother, y la memoria que ella utilizará en sus hazañas gráficas queda determinada desde el **SETUP** y es tomada del banco principal de memoria masiva.

Las distintas controladoras poseen asignado un número para cada “tipo” de tarjeta gráfica (especificadas en Gd (Graphic Driver)):

Valor	Constante
0	**DETECT** (solicita auto detección)
1	**CGA**
2	**MCGA**
3	**EGA**
4	**EGA64**
5	**EGAMONO**
6	**IBM8514**
7	**HERCMONO**
8	**ATT400**
9	**VGA**
10	**PC3270**

Si bien todas estas tarjetas se dejaron de fabricar hace muchos años, aún continúan contenidas dentro de las actuales **VGA**, ya sea por una cuestión de compatibilidad o por las razones de velocidad que analizábamos hace un rato.

Pero la historia no termina aquí, porque aún falta hablar del segundo parámetro de la función detectgraph( ): Gm (Graphic mode) o Modo Gráfico.

Cada tarjeta de la lista descripta anteriormente posee distintos modos de trabajo. Consideremos por ejemplo la VGA:

Tarjeta	Modo	Constante	Definición	Colores	Páginas
VGA	**VGALO**	0	640 x 200	16	2
	VGAMED	1	640 x 350	16	2
	VGAHI	2	640 x 480	16	1

Nótese que la modalidad VGALO permite trabajar con 128.000 píxeles, en tanto que la modalidad VGAHI posee 307.200 (más del doble). Obviamente el tamaño del píxel varía entre una modalidad y otra, pero a la hora de sopesar validad Vs. velocidad se decidirá cuál conviene más.

Si ya de entrada supiéramos la tarjeta con que cuenta nuestro equipo y su máxima modalidad, podríamos omitir la función detectgraph( ) y pasar directamente a la que verdaderamente inicializa al procesador en modo gráfico, la función:

void initgraph(&Gd,&Gm,PathToDriver);

Esta instrucción se comunica con el micro y le "ordena" que comience a operar en modo gráfico con la tarjeta **Gd** y la modalidad **Gm** (según valores de la tabla que hayamos elegido). Pero aún falta algo…los drivers (manejadores) que permitan hacer uso de la tarjeta en las modalidades que se hayan seleccionado. Estos drivers se hallan dentro de una carpeta especial, normalmente en **C:\BORLANDC\BGI** que debe indicarse en la variable cadena **PathToDriver**, y consisten en un conjunto de archivos que contienen una suerte de datos y códigos. Poseen extensiones .BGI (Binary Graphic Interface) y en el momento en que initgraph( ) se ejecuta, son cargados en memoria reservada donde permanecen protegidos mientras el modo gráfico se halle activo.

Cuando **intigraph( )** ha realizado su tarea inicializadora, el modo texto desaparece y ya no tienen vigencia ninguna de las funciones de la librería <conio.h> para manejo de texto. Si deseáramos ingresar datos por teclado estando en modo gráfico, tendríamos que diseñar nuestro propio editor gráfico (que se halla en un ejemplo más adelante). La pantalla se ha tornado totalmente oscura y si no se ejecuta ninguna función que grafique algo, permanecerá así hasta dar por finalizado el modo gráfico y retornar al modo texto…o finalizar el programa. También cabe destacar que esta inicialización realiza otras tareas:

- **Ingresa al procesador en modo gráfico.**
- **Carga en memoria reservada los drivers gráficos (.BGI)**
- **Limpia la memoria de trabajo.**
- **Inicializa los punteros gráficos.**

Si por alguna razón initgraph( ) no puede ejecutarse exitosamente, es posible detectar el error producido mediante la siguiente sintaxis:

```
initgraph(&Gd,&Gm,PathToDriver);
errorcode=graphresult( );
if(errorcode!=grOk) {
        printf("Se produjo un error gráfico: %s\n",grapherrormsg(errorcode));
        printf("Presione una tecla para finalizar…");
        getch( ); exit(1);
}
```

Donde "errorcode" es una variable int que toma el valor devuelto por la función **graph-result**( ). Esta función debe ir inmediatamente a continuación de la otra función que se desea chequear. El valor dado por la constante "**grOk**" (cero) indica que todo anduvo bien, caso contrario ocurrió algún imprevisto: no pudo setear la tarjeta gráfica, no halló los drivers en el paso indicado, etc.

No sé si habrá notado que estando en modo gráfico hemos utilizado la instrucción printf( ) para salidas a pantalla. ¡Buenas noticias! Funciona también en modo gráfico, no así nuestra clásica **cprintf**( ) de la librería conio.

Siempre resulta conveniente utilizar este código de verificación, al igual que en la parte de reservas dinámicas de memorias era útil chequear el valor devuelto por **malloc**( ).

Otra buena idea sería englobar las instrucciones de inicialización en una función de usuario, ya que utilizaremos la misma en todos los programas que trabajen con modo gráfico:

```
// ---------------------------------------------------------------------------------------
void ModoGrafico( )
{
  int Gd, Gm;
  char *Path = "C:\\BORLANDC\\BGI";

  detectgraph(&Gd,&Gm);
  initgraph(&Gd,&Gm,Path);
}
// ---------------------------------------------------------------------------------------
```

Nótese la doble barra inclinada hacia atrás. Debe ser así para que el compilador no lo interprete como un carácter de control.

Final de las operaciones gráficas.

Al ejecutarse exitosamente la función initgraph( ) quedamos formalmente habilitados para utilizar cualquier función de graficación para realizar trazados en pantalla. El programa gráfico puede ser tan extenso o complejo como se requiera, pero al finalizar las operaciones de graficación, siempre es conveniente ejecutar la instrucción:

closegraph( )

que tiene las siguientes finalidades:

- Retornar al procesador al modo texto.
- Liberar la memoria donde había almacenado los drivers gráficos.

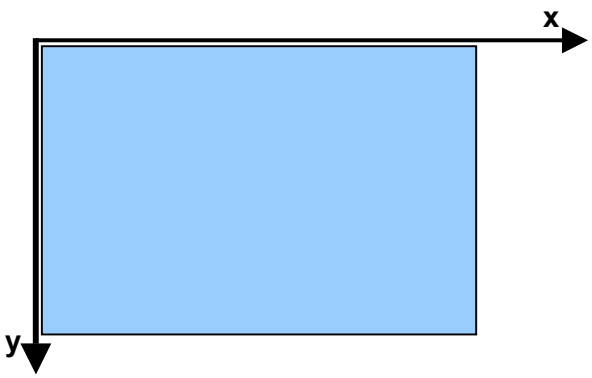
Esta operación de cierre produce normalmente unos transitorios eléctricos desagradables que se reflejan como oscilaciones de pantalla. A fin de evitarlos es conveniente aplicar el siguiente código:

```
// ---------------------------------------------------------------------------
void ModoTexto( )
{
  int RETARDO = 300;

  closegraph( );
  delay(RETARDO);
}
// ---------------------------------------------------------------------------
```

Donde closegraph( ) oscurece la pantalla al realizar sus tareas, y delay( ) establece una demora de 300 milisegundos, tiempo suficiente para que el sistema se estabilice eléctricamente y retorne suavemente al modo texto.

Sistema de ejes de pantalla.

El sistema gráfico establece el centro de coordenadas de pantalla (0,0) en el extremo superior izquierdo:



y en el modo **VGA** de máxima resolución los valores permitidos son:

Valores de “x” ..... de 0 a 639
Valores de “y” ..... de 0 a 479

NOTA: De ahora en adelante vamos a asumir que las funciones de usuario:

ModoGrafico( )
ModoTexto( )

ya se hallan definidas en el códigos del programa, por lo cual sólo la invocaremos.

Trazado de puntos luminosos: la instrucción putpixel( )

La impresión de un punto luminoso en pantalla (píxel), se lleva a cabo mediante la función predefinida:

void far putpixel(int Col, int Fila, int Color);

que toma como parámetros la posición horizontal y la posición vertical del punto, como así también su color. Para que le punto sea visible en pantalla los valores de Col y Fila deben hallarse dentro de los valores permitidos para la tarjeta y modo de graficación con que estemos trabando. Sin embargo si damos algún valor más allá de los límites naturales de la pantalla no se genera ninguna condición de error, sino que simplemente el punto **no se grafica**. Note que estos parámetros son enteros puesto que el píxel es la mínima expresión de graficación y no pueden existir fracciones de los mismos.

Otro aspecto interesante es que los parámetros de posición son tomados siempre como **enteros**, de manera que si trabajamos con valores decimales, el compilador hará la conversión automática a enteros en el momento de utilizarlos en putpixel( ).

De más está decir que antes de utilizar putpixel( ) o cualquier otra función gráfica, tendremos que haber inicializado el modo gráfico, caso contrario no son reconocidas.

Nuestra primera experiencia: llenar la pantalla con puntos de posición aleatoria.

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

void ModoGrafico ( );
void ModoTexto   ( );
// ---------------------------------------------------------------------
void main()
{
  int   Xp,Yp;     // --- coordenadas de pantalla.
  int   ColorON  = LIGHTBLUE;
  int   ColorOFF = BLACK;

  ModoGrafico();
  do {
       Xp=random(640);
       Yp=random(480);

       putpixel(Xp,Yp,ColorON);

       Xp=random(640);
       Yp=random(480);

       putpixel(Xp,Yp,ColorOFF);
     } while(!kbhit());

    ModoTexto();
}
// ---------------------------------------------------------------------
```

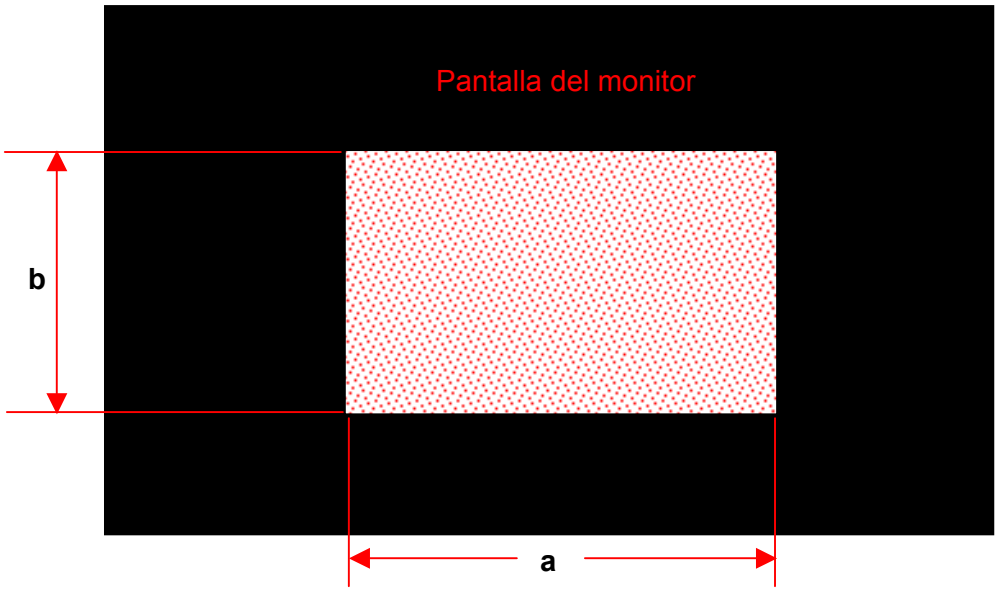
Nótese la sencillez del código que genera posiciones aleatorias entre 0 y 639 en sentido horizontal (Xp), y entre 0 y 439 en sentido vertical (Yp). Los puntitos se están dibujando y borrando permanentemente hasta tanto se pulse una tecla.

Una variación interesante de este programa sería acotar ahora los bordes para que los puntos aleatorios queden limitados a un área rectangular cualquiera. Incluso los colores podrían ser aleatorios:

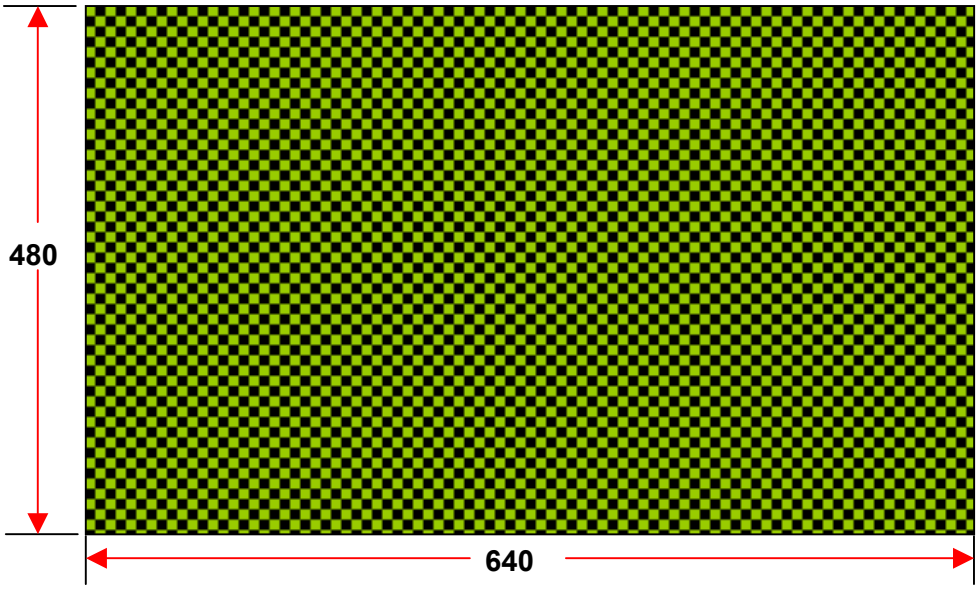
```
/* ------------ GRAFICACION EN C ----------------------------------
                LA FUNCION putpixel()
                GRAPH03.CPP


Partiendo del vértice superior izquierdo (Col,Fila) de un rectángulo
de ancho "a" y altura "b", rellenarlo con puntos aleatorios tanto en
posición como en color. El programa finalizará al pulsar una tecla
cualquiera.

------------------------------------------------------------------- */


#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<graphics.h>

void ModoGrafico (  );
void ModoTexto   (  );
// ------------------------------------------------------
void main()
{

  int   Col    = 200;
  int   Fila   = 100;
  int   a      = 250;
  int   b      = 200;
  int   Xp,Yp;
  int   Color;

  ModoGrafico(); randomize();

  do {
       Xp=Col+random(a);
       Yp=Fila+random(b);
       Color=random(16);
       putpixel(Xp,Yp,Color);
     } while(!kbhit());

  ModoTexto();
}
// ------------------------------------------------------
```

Un comentario al pasar: Ya habrá notado que estamos trabajando siempre con coordenadas que nunca sobrepasan los límites de pantalla. No es casualidad, ha sido pensado así para evitar tener que trabajar con escalas: eso vendrá más adelante.

Este programa al ejecutarse dibuja un rectángulo dentro de la pantalla y con colores aleatorios cambiantes:



¿Qué tal si ahora rompemos el esquema de los valores aleatorios y hacemos algo continuo? Por ejemplo un reticulado como el siguiente:



Pero de manera que la cantidad de cuadradito (o el tamaño de cada uno) pueda variarse a voluntad con cada corrida.

```
/* ----------- GRAFICACION EN C -----------------------------
               FUNCION putpixel( )
               GRAPH02.CPP

Implementar un programa en C que permita graficar N1 cuadritos
horizontales x N2 cuadritos verticales. También debe tener la
alternativa de poder indicar no la cantidad, sino la medida en
pixeles del ancho y de la altura de cada cuadrito. Además los
cuadritos se Irán alternando entre NEGRO y VERDE.
---------------------------------------------------------- */
```

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

void ModoGrafico (         );
void ModoTexto   (         );
int  Round       (double x );
// ---------------------------------------------------------
void main()
{

  int     i,j;
  int     Color;
  int     COLOR1 = BLACK;
  int     COLOR2 = LIGHTGREEN;
  int     Ancho  = 8;  // --- Ancho en pixeles.
  int     Alto   = 8;  // --- Altura en pixeles.
  int     N1     = 80; // --- Cuadritos horizontales.
  int     N2     = 60; // --- Cuadritos verticales.
  int     n1,n2;
  int     ParH,ParV;   // --- Paridad Horizontal y Vertical.
  int     PasoH  = 640/N1;
  int     PasoV  = 480/N2;

  ModoGrafico();

  for(ParV=1,n2=0,i=0;i<480;i++) {
      for(ParH=1,n1=0,j=0;j<640;j++) {
        if((ParH+ParV)%2!=0) Color=COLOR1; else Color=COLOR2;
        if(n1>PasoH) { n1=0; ParH++; } else n1++;
        putpixel(j,i,Color);
      }
      if(n2>PasoV) { n2=0; ParV++;} else n2++;
  }

  getch();
  ModoTexto();
}
// ---------------------------------------------------------
```

Para facilitar la tarea, establezcamos que los cuadraditos de orden par serán de color **NEGRO** y los de orden impar serán **VERDES**. Como el ejemplo ha sido resuelto para trabajar con “cantidad de cuadraditos”, debemos calcular el “Paso” entre cada uno de ellos, tanto en un sentido horizontal como en un sentido vertical, por eso tenemos dos variables:

PasoH
PasoV

que se calculan como:

PasoH =640/N1 **donde N1 y N2 son las cantidades de cuadraditos.**
PasoV =480/N2

ParH y ParV representan la paridad o **número de orden** de los cuadraditos en sentido horizontal y vertical, respectivamente.

Cuando los contadores de píxeles n1 y n2 alcanzan el ancho (PasoH) y el alto (PasoV), las variables de paridad se incrementan en 1. Según la suma de (ParH+ParV) resulte en un valor "par" o "impar", elegiremos un color u otro:

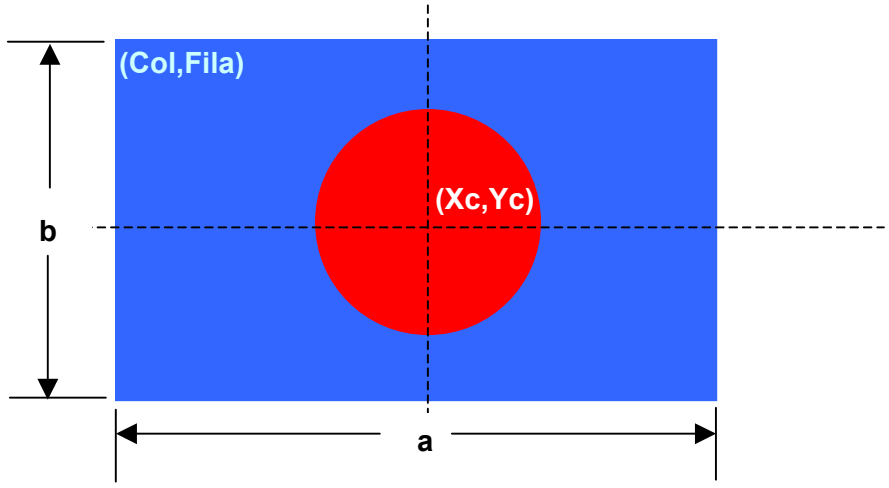
```
if((ParH+ParV)%2!=0) Color=COLOR1; else Color=COLOR2;
```

Casos un poco más complicados.

Ahora vamos a hacer intervenir algunas funciones matemáticas sencillas como ecuaciones de rectas y circunferencias, aunque más bien deberíamos decir del círculo ya que se trata de figuras llenas.

Por ejemplo:

Partiendo del vértice superior izquierdo (Col,Fila) de un rectángulo de ancho "a" y altura "b", rellenarlo con puntos aleatorios de color AZUL, pero teniendo en cuenta que dentro de él se halla un círculo de radio "R" y color ROJO. Las coordenadas del círculo son (Xc,Yc):




```
void main()
{
  int   Col    = 150;
  int   Fila   = 100;
  int   a      = 300;
  int   b      = 250;
  int   Xc     = 300;
  int   Yc     = 225;
  int   R      = 100;
  int   Xp,Yp;
  int   Color;

  ModoGrafico(); randomize();
```

```
  do {
       Xp=Col+random(a);
       Yp=Fila+random(b);

       if((pow(Xp-Xc,2)+pow(Yp-Yc,2))>=pow(R,2)) Color=LIGHTBLUE;
       else Color=LIGHTRED;
       putpixel(Xp,Yp,Color);

     } while(!kbhit());

  ModoTexto();
}
// ------------------------------------------------------------
```

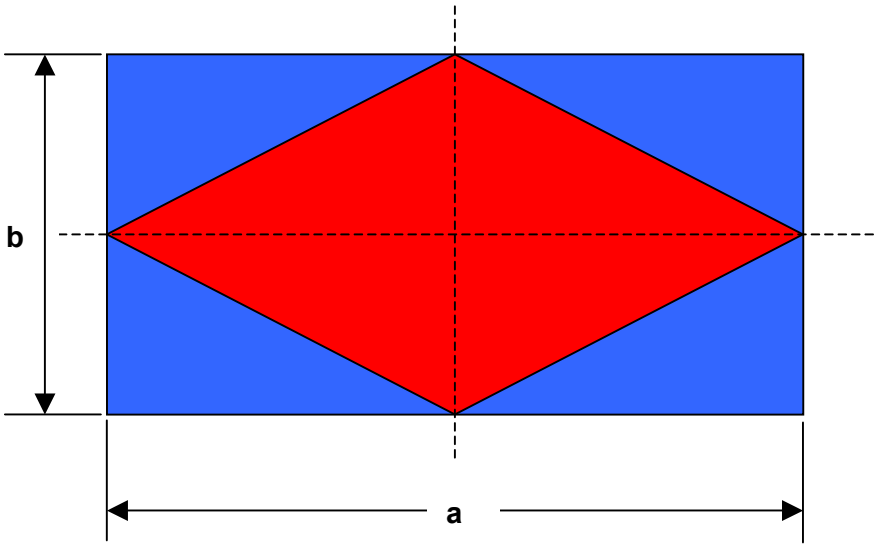
Aquí la condición clave para que los pixeles azules no invadan al círculo es:

$(Xp - Xc)^2 + (Yp - Yc)^2 >= R^2$

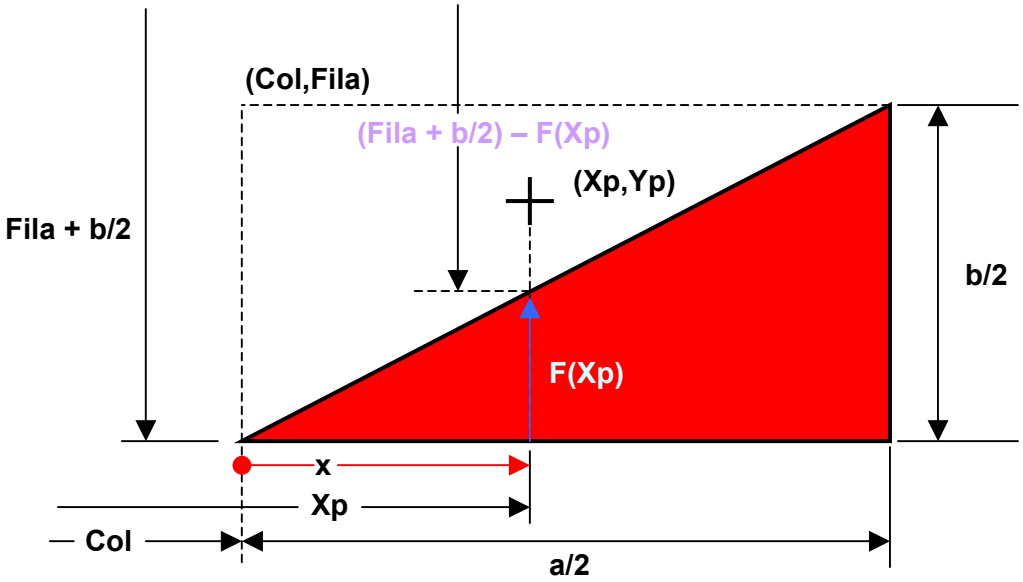
Dicho de otra manera: la distancia de los pixeles generados aleatoriamente, con respecto a las coordenadas (Xc,Yc), no debe ser menor al Radio del círculo. En ese caso asignamos un color AZUL, pues se trata del rectángulo, caso contrario un color ROJO pues se trata del círculo.

NOTA: hemos utilizado la función pow( ), por "power" o potencia, definida en la librería <math.h>. Para mayor información remitirse a ella.

Otro caso interesante es graficar, siempre como figuras llenas, un rombo dentro de un rectángulo:



Este es un caso interesante porque dada la simetría de la figura es posible trabajar en un solo cuadrante de la misma y luego sumar los incrementos para las otras tres porciones:



La pendiente de la diagonal vale: m =b/a y si consideramos provisoriamente el origen de coordenadas en el vértice agudo del triángulo, tendremos que la ecuación de esta recta viene dada por:

F(x) = m.x

pero como **x = Xp – Col**, tendríamos **F(Xp) = b/a.(Xp-Col)**

En realidad nosotros generamos puntos de coordenadas (Xp,Yp), que debemos verificar si se hallan en el triángulo superior o en el inferior, lo cual implica que la coordenada útil es:

(Fila +b/2) – F(Xp)
(Fila + b/2) – b/a.(Xp-Col)

Por lo tanto debemos chequear:

if(Yp<=(Fila +b/2) – b/a.(Xp-Col)) Color=LIGHTBLUE; else Color=LIGHTRED;

Una vez hallada la correcta ubicación del píxel de coordenadas Xp,Yp del primer cuadrante, corresponde hallar los otros tres por simetría:

```
putpixel (Xp,Yp,Color);
putpixel (Col*2+a-Xp,Yp,Color);
putpixel (Xp,Fila*2+b-Yp,Color);
putpixel (Col*2+a-Xp,Fila*2+b-Yp,Color);
```

Nótese que la simetría se da con respecto a las caras ortogonales del triángulo rojo.

He aquí el código completo:

```
// ------------------------------------------------------------
void main()
{

  int     Col   = 150;
  int     Fila  = 100;
  int     a     = 300;
  int     b     = 200;
  int     Xp,Yp;
  int     Color;
  double  m     = (double)b/(double)a;

  ModoGrafico(); randomize();

  do {
       Xp=Col +random(a/2+1);
       Yp=Fila+random(b/2+1);

       if(Yp<=(Fila+b/2-m*(double)(Xp-Col))) Color=LIGHTBLUE;
       else Color=LIGHTRED;

       putpixel(Xp,Yp,Color);
       putpixel(Col*2+a-Xp,Yp,Color);
       putpixel(Xp,Fila*2+b-Yp,Color);
       putpixel(Col*2+a-Xp,Fila*2+b-Yp,Color);

     } while(!kbhit());

  ModoTexto();
}
// ------------------------------------------------------------
```

Los problemas que impliquen rellenado de figuras existen en cantidades industriales y basta un poco de imaginación para crear docenas de ellos tan complicados o artísticos como deseemos.

Si ha sido observador, habrá notado que nos hemos manejado fundamentalmente con los ejes de graficación de la pantalla y no con ningún sistema direrente, como podría ser un par de ejes matemáticos en los cuales toma valor una función.

Ese será el tema de nuestro próximo capítulo. Sin embargo antes de finalizar con éste, presentaremos una función muy útil llamada:

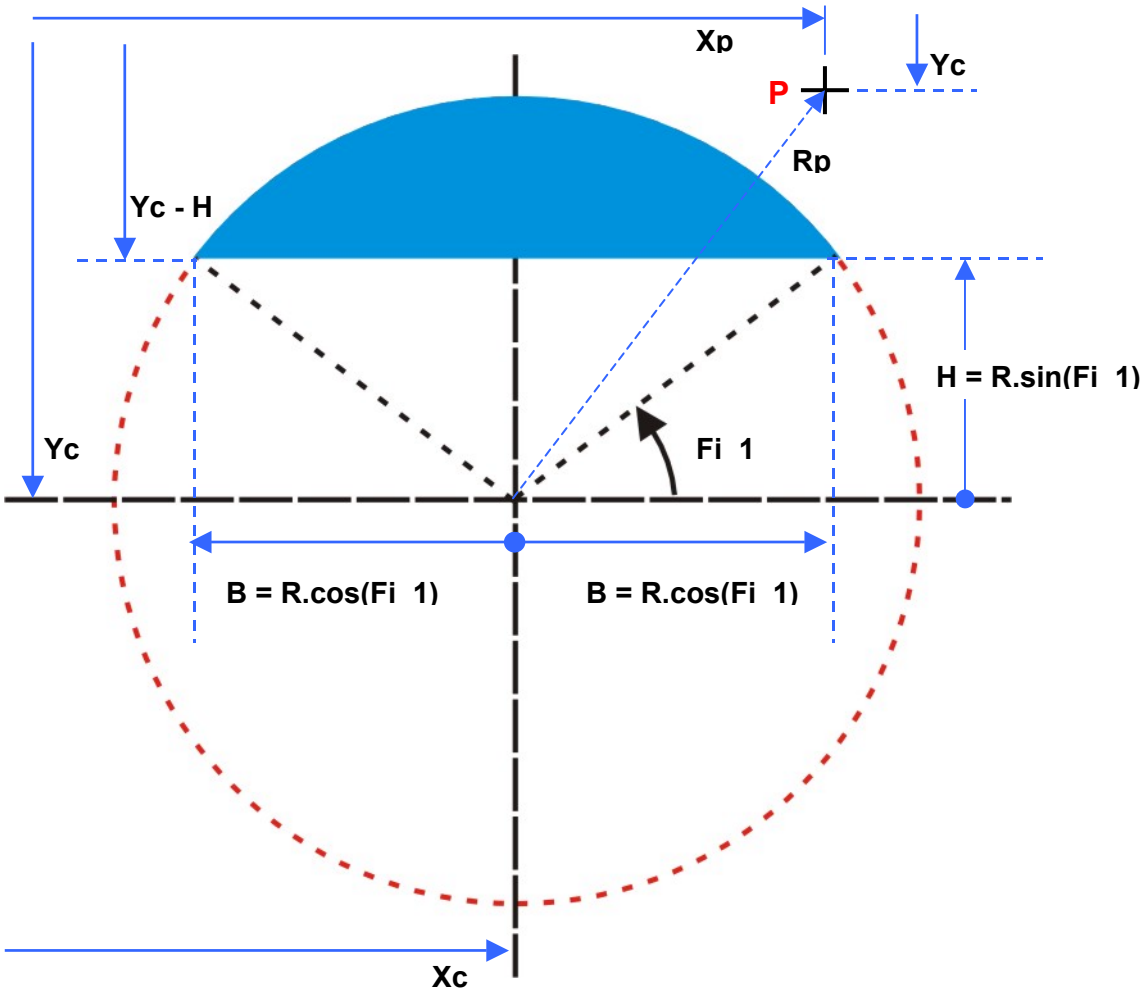
getmaxx( )
getmaxy( )

que retornan la mayor coordenada de un pixel en sentido horizontal y vertical, respectivamente, que permite el modo de graficación actual.

Un casquete circular.

Vamos a dibujar un casquete circular lleno, utilizando dos métodos: con puntos de ubicación aleatoria y luego en forma determinística, con el agregado, en ambos casos, de modificar el ángulo de arranque del casquete y poder llegar a graficar un círculo completo.

Como siempre, lo primero es un buen esquema de la tarea:



El punto indicado como **P**, de coordenadas (Xp,Yp), es un punto genérico obtenido aleatoriamente en cualquier posición dentro de los límites físicos de la pantalla mediante:

```
Xp = random(640);
Yp = random(480);
```

Ahora corresponde determinar si P se halla dentro o fuera del casquete:

```
if(Xp>=(Xc-B) && Xp<=(Xc+B))
        if(Yp<=(Yc-H) && Rp<=R)
                putpixel(Xp,Yp,Color);
```

Primero detectamos si el **Xp** se halla dentro de los límites del segmento horizontal del casquete: **[Xc – B, Xc + B**]. Si así ocurre, entonces resta analizar si la coordenada vertical **Yp** pertenece también al casquete: o sea si está por arriba de **Yc – H** y si su distancia **Rp** al centro de la circunferencia es menor que el radio **R**.

El código completo de este problema es el siguiente:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (  );
void ModoTexto   (  );

// ---------------------------------------------------------
void main()
{

  int     Xc      = 320;
  int     Yc      = 240;
  int     R       = 150;
  int     Color   = YELLOW;
  int     Xp,Yp;
  int     Parte   = 2;

  double  Rg;
  double  AngGen;
  double  H,Rp,B;
  double  Fi_1    = 5*M_PI/180.00;


  randomize(); ModoGrafico(); H=R*sin(Fi_1); B=R*cos(Fi_1);

  do {

       Xp = random(640);
       Yp = random(480);

       Rp = sqrt(pow(Xp-Xc,2)+pow(Yp-Yc,2));

       if(Xp>=(Xc-B) && Xp<=(Xc+B))
        if(Yp<=(Yc-H) && Rp<=R) {
           putpixel(Xp,Yp,Color);
           if(Parte==2) putpixel(Xp,Yc+Yc-Yp,Color);
        }

     } while(!kbhit());

  getch(); ModoTexto();
}
// ---------------------------------------------------------
```

Habrá observado el agregado de : if(Parte==2) putpixel(.....) Ello tiene por finalidad dibujar los dos casquetes simétricos que se generaría para un mismo ángulo de inicio. Si prueba haciendo el Fi_1 = 0 verá que se dibuja un círculo completo, que era parte de los pedido en el enunciado.

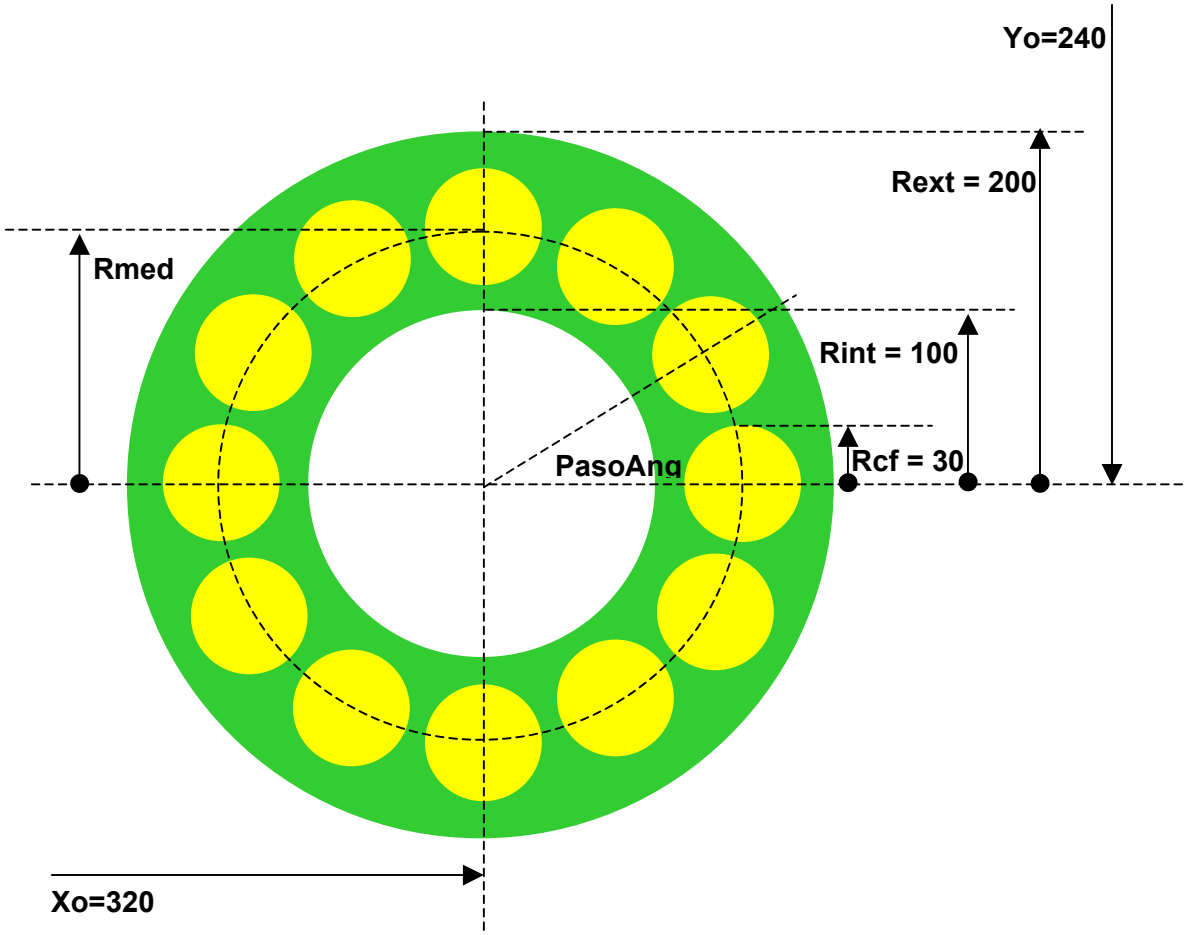
Ahora a rediseñar el programa para que el llenado se realice no con puntos aleatorios, sino de manera determinística:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>


void ModoGrafico (  );
void ModoTexto   (  );

// ------------------------------------------------------
void main()
{

  int     Xc      = 320;
  int     Yc      = 240;
  int     R       = 150;
  int     Color   = YELLOW;
  int     Npts    = 500;
  int     Parte   = 2;
  int     Xp,Yp;

  double  AngGen;
  double  Fi_1    = 2*M_PI/180.00;
  double  PasoAng = (M_PI-(2*Fi_1))/Npts;
  double  x,H,B;

  ModoGrafico();

  for(AngGen=Fi_1;AngGen<=M_PI/2;AngGen+=PasoAng) {

      B  = R*cos(AngGen);
      H  = R*sin(AngGen);
      Yp = Yc-H;

      for(Xp=Xc-B;Xp<=Xc+B;Xp++) {
        putpixel(Xp,Yp,Color);
        if(Parte==2) putpixel(Xp,Yc+H,Color);
      }
  }


  getch(); ModoTexto();
}
// ------------------------------------------------------------------
```

Dejamos a Ud. la interpretación de este sencillo y breve código (analice **SIEMPRE** sobre el dibujo acotado anterior).

Un verdadero acertijo.

Implementar un programa que, generando puntos de emplazamiento aleatorio, llegue a trazar la siguiente figura llena:



NOTA: Lo que va punteado no es parte del dibujo.

Los puntos aleatorios deben generarse de manera que siempre caigan **dentro** del anillo circular, lo cual implicaría determinar en realidad dos magnitudes aleatorias:

```
AngGen = (double)random(6283) / (double)1000;
RGen   = Rint + random(Rext – Rint + 1);
```

El primero genera un ángulo en radianes entre cero y 2*PI, y el segundo un radio genérico entre el radio interior y el radio exterior. En base a esto las coordenadas de pantalla serían:

```
Xp = Xo + RGen*cos(AngGen);
Yp = Yo + RGen*sin(AngGen);
```

Pero ahora viene la parte interesante del problema: acabamos de crear un píxel en alguna parte del anillo ¿pero cómo sabemos si este píxel está dentro de alguna de los círculos o en el anillo en sí? **ESTA** es la parte a resolver.

Llamemos **PasoAng** al ángulo en radianes entre dos círculos consecutivos, medido con respecto al centro del anillo (ver esquema), y pensemos de la siguiente manera:

Si unimos el punto aleatorio de coordenadas **(Xp,Yp**) con el centro del anillo, veremos que forma un cierto ángulo. Ahora bien, a ambos lados de este ángulo existirán círculos que podrían englobar al punto en cuestión, es lo que debemos averiguar:

```
K = (int)(AngGen / PasoAng);

Color = ColorAn;
for(i = 0;i<2;i++) {

   AngCf = (K+i)*PasoAng;
   Xc = Xo + Rmed*cos(AngCf);
   Yc = Yo + Rmed*sin(AngCf);
   if((pow(Xp-Xc,2) + pow(Yp-Yc,2)) < pow(Rcf,2)) {
       Color = ColorCf;
       break;
  }
}
putpixel(Xp,Yp,Color);
```

K es el número de pasos angulares para determinar el ángulo de los círculos **MAS PROXIMOS** al punto aleatorio. Como habrá uno a cada lado tendremos que recorrer con un lazo for( ) de 1 hasta 2, determinar los centros **Xc** y **Yc** de los círculos, y averiguar si la distancia del punto (**Xp,Yp**) con respecto a esos centros es menor que el radio.

Claro, una vez explicado parece muy sencillo, pero hay que llegar a eso ¿verdad?

CAPÍTULO 2 – Graficación de funciones matemáticas.

La graficación de funciones matemáticas implica generalmente dos problemas:

- La existencia **explícita** de **ejes de graficación** diferentes de los propios de pant.
- La necesidad de una **escala** para compatibilizar las medidas.

Además al ser los ejes simples líneas horizontales o verticales, si bien podemos graficarlo con putpixel( ), "C" dispone de una función que permite trazar directamente una línea:

line (X1, Y1, X2, Y2);

Donde X1, Y1 son las coordenadas de un extremo de la línea, y X2, Y2 las del otro extremo. Todos estos parámetros son int por las mismas razones que invocamos en la función putpixel( ): no pueden existir fracciones de píxel.

Sin embargo a diferencia de putpixel( ) carece de un parámetro de color, por lo cual deberemos recurrir a una instrucción adicional para establecer el mismo:

setcolor(ColorFrente)
setbkcolor(ColorFondo)

Siendo Color un valor entero que puede variar entre 0 y 15. A modo de referencia estos son los colores, sus valores numéricos y las constantes que los representan:

Color	Valor	Fondo	Frente
BLACK	0	**SI**	**SI**
BLUE	1	**SI**	**SI**
GREEN	2	**SI**	**SI**
CYAN	3	**SI**	**SI**
RED	4	**SI**	**SI**
MAGENTA	5	**SI**	**SI**
BROWN	6	**SI**	**SI**
LIGHTGRAY	7	**SI**	**SI**
DARKGRAY	8	**NO**	**SI**
LIGHTBLUE	9	**NO**	**SI**
LIGHTGREEN	10	**NO**	**SI**
LIGHTCYAN	11	**NO**	**SI**
LIGHTRED	12	**NO**	**SI**
LIGHTMAGENTA	13	**NO**	**SI**
YELLOW	14	**NO**	**SI**
WHITE	15	**NO**	**SI**

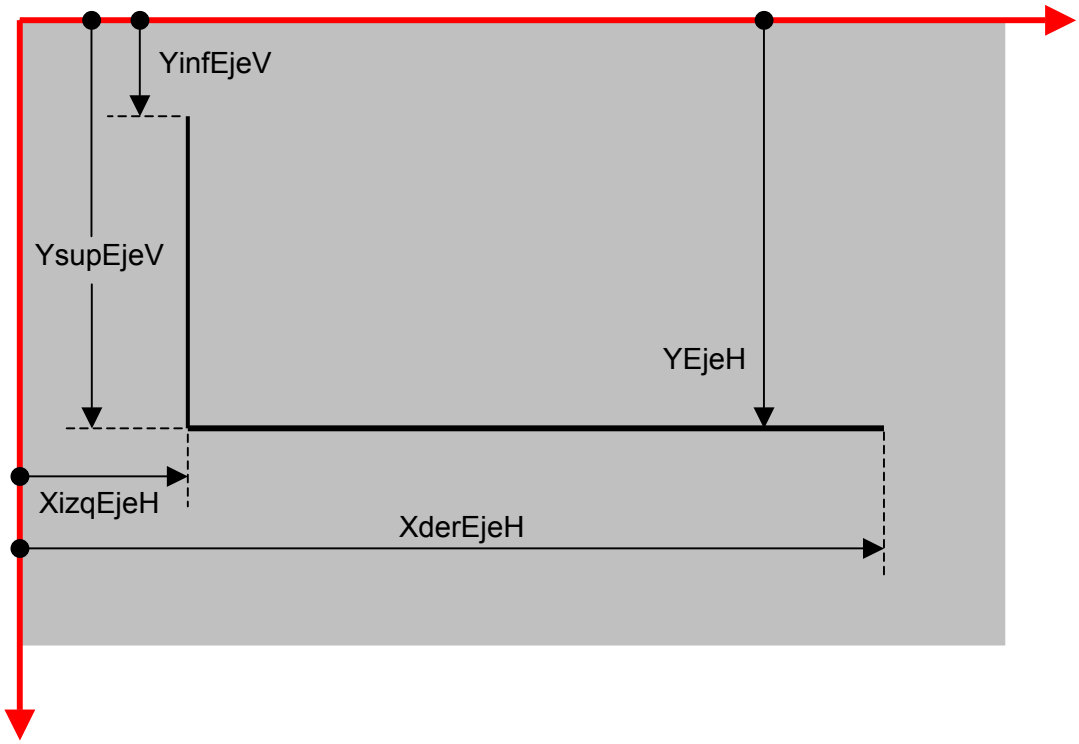
Si no se establece un color para el fondo, el compilador adopta el Black por defecto.
Los colore pueden escribirse directamente, lo cual resulta muy práctico, por ejemplo:

setcolor(LIGHTGREEN)
setcolor(YELLOW)
etc.

La partícula **LIGHT** precediendo al nombre indica que se trata de un color brillante. La limitación de los colores de fondo hace posible que podamos tener dos tonalidades del mismo color: uno como fondo y el otro como frente. Por ejemplo:

```
setbkcolor(BLUE)
setcolor(LIGHTBLUE)
```

Para visualizar cómo encarar la graficación de los ejes hagamos el siguiente esquema:



Las líneas en **ROJO** son los ejes de pantalla.
Es altamente conveniente explicitar cada línea de ejes a través de las coordenadas especificadas ya que así podremos redimensionar o reubicar los mismos con suma comodidad. Además veremos en breve que también facilitan la graficación de las funciones matemáticas.

Ahora pasemos al código para generar estos ejes coordenados:

```
/* -------------------- GRAFICACION DE FUNCIONES ------------------
                        GRAPH07.CPP

Dibujar un par de ejes coordenados.

---------------------------------------------------------------- */

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

```
void ModoGrafico ( );
void ModoTexto   ( );
// ---------------------------------------------------------------
void main()
{
  int   XizqEjeH  =  50;
  int   XderEjeH  = 550;
  int   YEjeH     = 300;
  int   XEjeV     =  50;
  int   YinfEjeV  =  50;
  int   YsupEjeV  = 300;
  int   ColorEjes = LIGHTCYAN;

  ModoGrafico();

  setcolor(ColorEjes);
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);

  getch();

  ModoTexto();
}
// ---------------------------------------------------------------
```

También podríamos haber trazado cada eje en un color diferente haciendo:

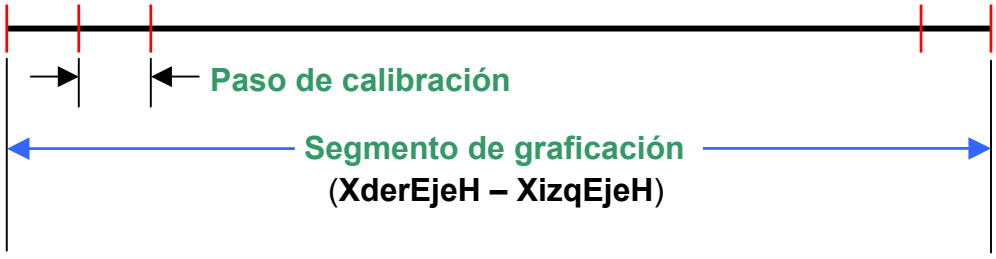
ColorEjeH = ...
ColorEjeV = ...

pero eso lo dejamos para lector.

Calibración de ejes.

La calibración de los ejes consiste en ese conjunto de rayitas que representan alguna medida de lo que se está graficando. Si por ejemplo el segmento horizontal de los ejes implica un recorrido angular de 360 grados y hemos calibrado al eje con 12 rayitas, significa que cada segmentito representa 360 / 12 = 30 grados. Así, al graficar por ejemplo una función seno( ), tendremos una idea de dónde pasa la función por cero o por algún máximo.

Aquí aparecerá alguna dificultad conceptual para graficar la calibración. Analicemos:



El “Paso” de calibración es una magnitud que indica la **cantidad de píxeles** que deben separar cada rayita de la escala graduada, y si bien dijimos que estas magnitudes

no pueden implicar fracciones de pixeles, estamos frente a un caso especial. Veamos por qué:

PasoCal = SegCal / NDiv

y supongamos: SegCal = 640 y NDiv = 25

con lo cual obtendríamos: PasoCal = 25,6
Si realizáramos este cálculo de manera de obtener un entero tendríamos un pequeño error por truncamiento, y vea lo que pasa:

PasoCal x NDiv = 25 x 25 = 625 en lugar de 640 que debería ser lo correcto.

¿Cómo solucionamos esto? trabajando con punto flotante:

double PasoCal = (double)SegCal / (double) NDiv

Y ahora dejamos que el pasaje a entero lo realice la propia función de graficación:

line (XizqEjeH + n*PasoCal, YEjeH-3, XizqEjeH+n*PasoCal, YEjeH+3);

De esta manera obtenemos un resultado de graficación muchísimo más exacto. Por ejemplo no es lo mismo hacer:

25 x 11 = 275 que truncar 25,6 x 11 = 281

Zanjado este importante concepto, pasemos a otro que suele producir serios dolores de cabeza: la división real de dos enteros. Si hubiésemos hecho:

double PasoCal;

PasoCal = SegGraf / NDiv (siendo SegGraf y NDiv enteros), hubiésemos obtenido:
PasoCal = 25.0000...

lo cual es lógico, puesto que el compilador se encuentra con una magnitud entera que debe ser dividida por otra magnitud entera. Entonces asigna registros transitorios de tipo entero, realiza la operación de división entera **y recién al resultado** lo transforma en double y los asigna a PasoCal, con lo cual resultaría: PasoCal = 25,000000 que no es lo deseado. ¿Solución? Utilizar nuestros viejos amigos los cast's.

El código completo es el siguiente:

```
/* -------------------- GRAFICACION DE FUNCIONES ------------------
                        GRAPH08.CPP

Dibujar los mismos ejes coordenados anteriores y agregarle una cali-
bración sólo al eje horizontal, por ejemplo 25 divisiones.

Agregar además algunas características como espesor de líneas.
------------------------------------------------------------------ */

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

```
#include<stdio.h>
#include<graphics.h>
void ModoGrafico ( );
void ModoTexto   ( );

// ---------------------------------------------------------------------

void main()
{

  int     XizqEjeH  =  50;
  int     XderEjeH  = 550;
  int     YEjeH     = 300;
  int     XEjeV     =  50;
  int     YinfEjeV  =  50;
  int     YsupEjeV  = 300;
  int     ColorEjes = LIGHTCYAN;
  int     NDiv      =  20;
  double  PasoCal   = (double)(XderEjeH-XizqEjeH)/(double)NDiv;
  int     ColorCal  = YELLOW;
  int     n;

  ModoGrafico();

  // --- TRAZADO DE EJES COORDENADOS -------------

  setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);

  // --- TRAZADO DE LINEAS DE CALIBRACION --------

  setcolor(ColorCal);
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);
  for(n=1;n<=NDiv;n++)
      line(XizqEjeH+n*PasoCal,YEjeH-3,XizqEjeH+n*PasoCal,YEjeH+3);

  getch();

  ModoTexto();
}
// ---------------------------------------------------------------------
```

Sin embargo hemos introducido un elemento nuevo, la instrucción:

setlinestyle (SOLID_LINE, 1, THICK_WIDTH)
setlinestyle (SOLID_LINE, 1, NORM_WIDTH)

La primera indica que toda línea será dibujada en triple espesor, en tanto que la segunda retorna a la condición de espesor simple. Consulte en el help del entorno para obtener mayores precisiones sobre sus parámetros y modos de operación.

Al fin llegó la hora de graficar nuestra función matemática.

Graficación de funciones senoidales.

Si bien para nosotros es más cómodo trabajar en ángulos sexagesimales, las funciones trigonométricas deben recibir sus argumentos en radianes, por lo cual sería conveniente declarar todo lo que sea ángulos, como datos de tipo double de la forma:

double AngFi = AngGrad x Krad

Donde Krad es el factor de conversión de grados sexagesimales a radianes y AngGrad sí podría ser una variable entera.
Una función senoidal será, en general, de la forma:

F(x) = Ampl x sin(x + Fase)

Donde Ampl puede tomar cualquier valor, incluso mucho más allá de los límites de la pantalla (lo mismo para el ángulo "x"). Entonces debemos ir pensando en trabajar con escalas.

¿Qué es una escala? Podemos definirla como:

Una relación entre la realidad y su representación.

De más está decir que nuestro medio de representación será la pantalla gráfica, y que las magnitudes a representar pueden ser de diverso tipo:

- Simples magnitudes adimensionales (funciones matemáticas puras)
- Magnitudes eléctricas expresadas en Voltios, Amperes, etc.
- Distancias en metros, kilómetros, etc.
- Sistema de fuerzas expresadas en Kg, Newtons, etc.
- Y muchas otras.

Lo que a nosotros nos interesa **para su representación** son las magnitudes en sí y no lo que ellas significan en el universo al que pertenecen. Eso para su representación, sin embargo sería bueno poder interpretar en términos de la realidad lo que hemos logrado representar. Por ejemplo a qué distancia del origen se encuentran dos móviles que avanzan con velocidad constante, o para qué valor del tiempo una onda de tensión alcanza un nivel dado, etc. Entonces surge la necesidad de otra escala: **la escala de usuario**. Entonces ya tenemos dos conceptos importantes:

- **Escala de graficación.**
- **Escala de interpretación.**

La primera relaciona la cantidad de pixeles disponibles para graficar con la magnitud a representar en esa cantidad. En cambio la segunda relaciona la magnitud representada, con el número de divisiones del eje calibrado.

Ejemplo: Debemos graficar en un espacio de 640 pixeles, una variable física que varía de 20 a 1200 microsegundos, y deseamos que el eje de referencia posea 20 divisiones de calibración.

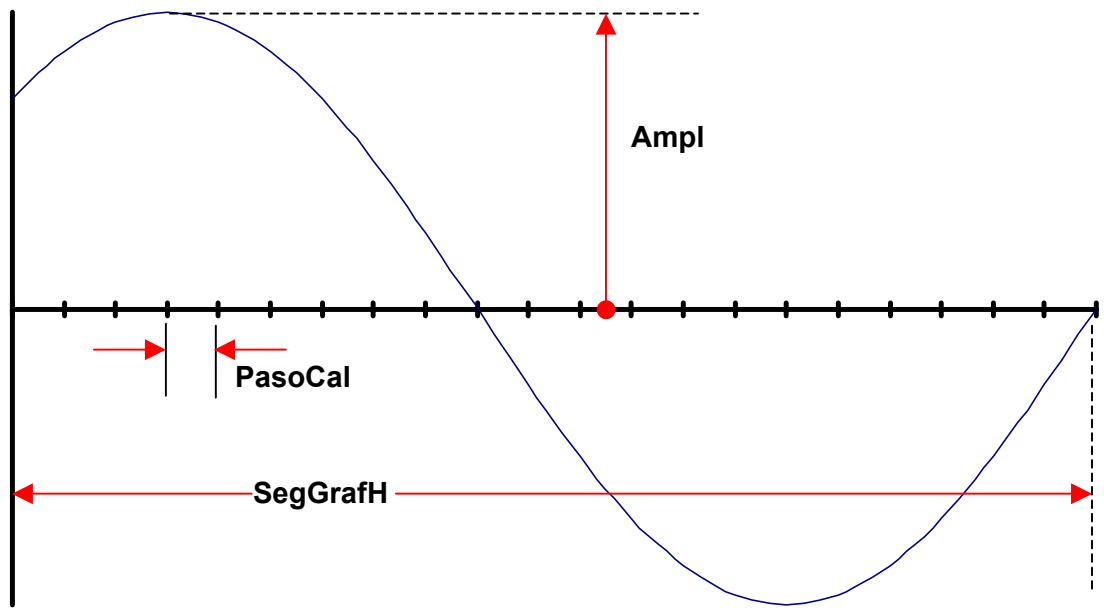
EscGraf = (double)SegGraf / (double)(Xder – Xizq) [pixel / MagnFisica]
EscUs = (double)(Xder – Xizq) / NDiv [MagnFisica / División]

La primera asegura la graficación en el espacio disponible y la segunda su interpretación en base al número de divisiones del eje. Obviamente cuanto más divisiones tengamos, más fina será la apreciación... pero todo tiene un límite: el que da la legibilidad.

Ambas escalas se hallan ligadas mediante:

EscGraf x EscUs = SegGraf / NDiv

Finalizada esta digresión necesaria sobre las escalas y su importancia, nos proponemos llegar a la siguiente gráfica:



Una función senoidal con un ángulo de fase inicial (Fase) y con una calibración de 15 grados por división. Nuestros segmentos de graficación quedarán definidos como vinimos haciendo hasta ahora:

SegGrafH = XderEjeH – XizqEjeH
SegGrafV = YEjeH – YinfEfeV

Los códigos quedarían de la siguiente manera:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<graphics.h>
#include<math.h>

void ModoGrafico ( );
void ModoTexto   ( );
// ---------------------------------------------------------------------
void main()
{

  int     Ampl      = 720;
  double  Krad      = M_PI/180.00;
  double  AngIni    =  45*Krad;
  double  AngFin    = 360*Krad;
  int     NPts      = 150;
  int     XizqEjeH  =  50;
  int     XderEjeH  = 550;
  int     YEjeH     = 240;
  int     XEjeV     =  50;
  int     YinfEjeV  =  50;
  int     YsupEjeV  = 450;
  int     ColorEjes = LIGHTCYAN;
  int     NDiv      =  36;
  int     Xp,Yp;
  double  PasoCal   = (double)(XderEjeH-XizqEjeH)/(double)NDiv;
  double  EscH      = (double)(XderEjeH-XizqEjeH)/(AngFin-AngIni);
  double  EscV      = (double)(YEjeH-YinfEjeV)/(double)Ampl;
  double  PasoAng   = (AngFin-AngIni)/(double)NPts;

  double  Ang,Fn;
  int     ColorFn   = LIGHTRED;
  int     ColorCal  = YELLOW;
  int     n;
```

Recordamos una vez más la importancia de evitar los errores por truncamiento o por redondeo (PasoCal, EscH, etc.) El resto del programa es el siguiente:

```
ModoGrafico();

  // --- TRAZADO DE EJES COORDENADOS -------------

  setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);


  // --- TRAZADO DE LINEAS DE CALIBRACION --------

  setcolor(ColorCal);
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);
  for(n=1;n<=NDiv;n++)
      line(XizqEjeH+n*PasoCal,YEjeH-3,XizqEjeH+n*PasoCal,YEjeH+3);
```

```
  // --- TRAZADO DE LA FUNCION SENOIDAL ----------

  for(Ang=AngIni;Ang<=AngFin;Ang+=PasoAng) {
      Xp=XizqEjeH+(Ang-AngIni)*EscH;
      Yp=YEjeH-EscV*Ampl*sin(Ang);
      putpixel(Xp,Yp,ColorFn);
  }

  getch();

  ModoTexto();
}
// ------------------------------------------------------------------
```

La siguiente instrucción merece nuestra atención:

Xp = XizqEjeH + (Ang-AngIni)*EscH;

ya que debe implicar que para **TODOS** los valores de la variable de graficación (Ang), los **Xp** deben hallarse sobre el eje horizontal. Sometamos a prueba esta expresión para ambos valores extremos:

Ang = AngIni → Xp = XizqEjeH + (AngIni – AngIni)*EscH
= XizqEjeH
Ang = AngFin → Xp = XizqEjeH +
(AngFin – AngIni)*
(XderEjeH -XizqEjeH)/(AngFin – AngIni)
= XderEjeH

Con lo cual obtuvimos lo correcto.

Si todavía continúa cuestionándose porqué se restó AngIni a cada valor de la variable Ang, la respuesta es simple: la función senoidal no arrancaría del extremo izquierdo del eje, sino del valor correspondiente a “Fase”, como si estuviésemos graficando desde cero grado pero con la función no definida sino a partir de Fase (45º).

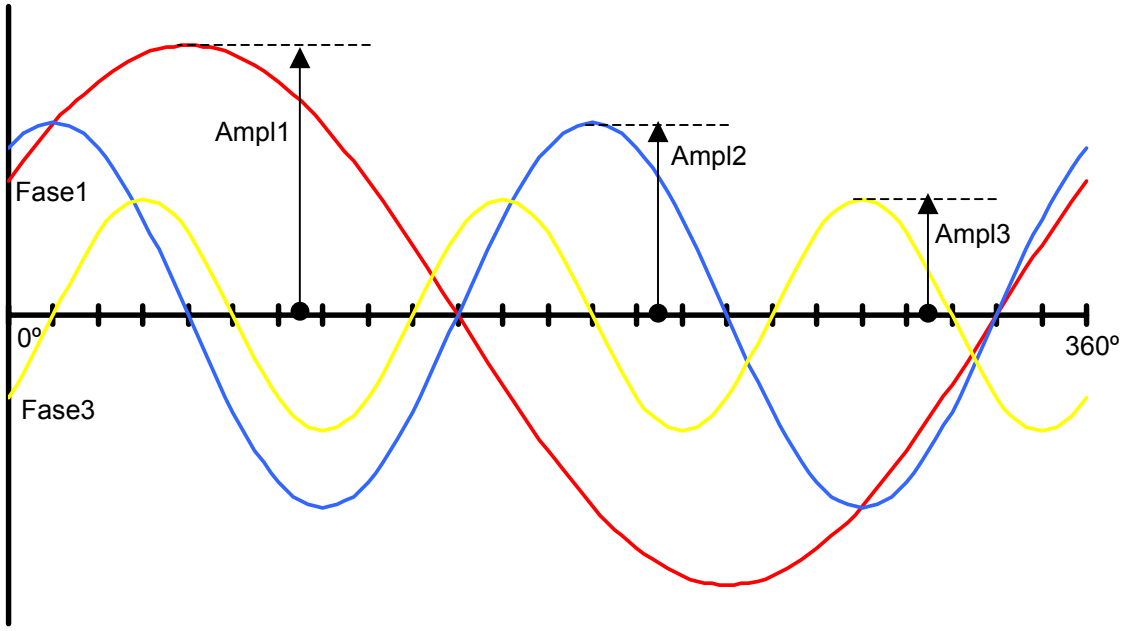
Además “Fase” puede tener cualquier valor. Para el actual, 45º, en realidad hemos representado menos de un ciclo completo puesto que la curva de 45 a 360º, pero si hubiésemos tomado Fase = -90º habríamos recorrido más de un ciclo completo, más exactamente 450º.

Sin embargo no es ésta la única forma de resolver este problema, como veremos en el próximo ejercicio.

Conjunto de ondas senoidales de distintas características.

A continuación vamos a plantearnos un problema mucho más general en cuanto a graficación de ondas senoidales, de tal suerte que un caso particular de éste, englobaría al problema que acabamos de analizar.

Nuestro objetivo actual es llegar a la siguiente gráfica:



Cada onda posee los siguientes parámetros particulares:

- **Amplitud**
- **Frecuencia**
- **Fase**

Comencemos a estudiar los primeros códigos:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<graphics.h>
#include<math.h>

void ModoGrafico ( );
void ModoTexto   ( );
// ---------------------------------------------------------------------
void main()
{
  int     Ampl1     = 720;
  int     Ampl2     = 500;
  int     Ampl3     = 300;
  int     Frec1     =   1;
  int     Frec2     =   2;
  int     Frec3     =   3;
  double  Krad      = M_PI/180.00;
  double  AngIni    = 0*Krad;
  double  AngFin    = 360*Krad;
  double  Fase1     =  30*Krad;
  double  Fase2     =  60*Krad;
  double  Fase3     = -45*Krad;
  double  Ang1,Ang2,Ang3;
```

```
int     NPts      = 350;
int     XizqEjeH  =  50;
int     XderEjeH  = 550;
int     YEjeH     = 240;
int     XEjeV     =  50;
int     YinfEjeV  =  50;
int     YsupEjeV  = 450;
int     ColorEjes = LIGHTCYAN;
int     NDiv      =  36;
int     Xp1,Yp1;
int     Xp2,Yp2;
int     Xp3,Yp3;

double  PasoCal   = (double)(XderEjeH-XizqEjeH)/(double)NDiv;
double  EscH      = (double)(XderEjeH-XizqEjeH)/(AngFin-AngIni);
double  EscV      = (double)(YEjeH-YinfEjeV)/(double)Ampl1;
double  PasoAng   = (AngFin-AngIni)/(double)NPts;

double  Ang,Fn;
int     ColorFn1  = LIGHTRED;
int     ColorFn2  = LIGHTBLUE;
int     ColorFn3  = YELLOW;
int     ColorCal  = YELLOW;
int     n;
```

En el problema anterior habíamos considerado AngIni = Fase porque en ese momento nos parecía más sencillo trabajar así. Ahora hemos cambiado la estrategia: el eje angular será común a todas las ondas y tendrá los mismos extremos para las tres, de cero a 360º. Entonces ¿dónde aparecen los desfasajes? Lo veremos en unos instantes al realizar la graficación.

Otra cosa: la escala de graficación.

Con la escala horizontal no existe ningún problema porque el recorrido angular es único: 360 grados, pero con la escala vertical tenemos tres magnitudes diferentes para hacerlas calzar en un mismo espacio de graficación: Ampl1, Ampl2 y Ampl3, pero afortunadamente no se genera ningún conflicto porque con solo considerar la magnitud más grande (el caso más desfavorable) las otras se adaptan automáticamente. Por ello hemos escrito directamente:

double EscV = (double)(YEjeH-YinfEjeV) / (double)Ampl1;

Aunque en realidad deberíamos haber chequeado con un if( ) las tres amplitudes para detectar la mayor, y quedarnos con ésta, pero en aras de la simplicidad lo hicimos por asignación directa.

Continuemos con nuestro código:

```
ModoGrafico();

// --- TRAZADO DE EJES COORDENADOS -------------

setcolor(ColorEjes);
setlinestyle(SOLID_LINE,1,THICK_WIDTH);
line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
```

```
  // --- TRAZADO DE LINEAS DE CALIBRACION --------

  setcolor(ColorCal);
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);
  for(n=1;n<=NDiv;n++)
      line(XizqEjeH+n*PasoCal,YEjeH-3,XizqEjeH+n*PasoCal,YEjeH+3);

  // --- TRAZADO DE LA FUNCION SENOIDAL ----------

  for(Ang=AngIni;Ang<=AngFin;Ang+=PasoAng) {

      Ang1=Frec1*Ang+Fase1;
      Ang2=Frec2*Ang+Fase2;
      Ang3=Frec3*Ang+Fase3;

      Xp1=XizqEjeH+Ang*EscH;
      Yp1=YEjeH-EscV*Ampl1*sin(Ang1);

      Xp2=XizqEjeH+Ang*EscH;
      Yp2=YEjeH-EscV*Ampl2*sin(Ang2);

      Xp3=XizqEjeH+Ang*EscH;
      Yp3=YEjeH-EscV*Ampl3*sin(Ang3);

      putpixel(Xp1,Yp1,ColorFn1);
      putpixel(Xp2,Yp2,ColorFn2);
      putpixel(Xp3,Yp3,ColorFn3);
  }
  getch();
  ModoTexto();
}
// ------------------------------------------------------------------
```

Nótese que el tema de los desfasajes aparece aquí:

```
Ang1 = Frec1*Ang + Fase1;
Ang2 = Frec2*Ang + Fase2;
Ang3 = Frec3*Ang + Fase3;
```

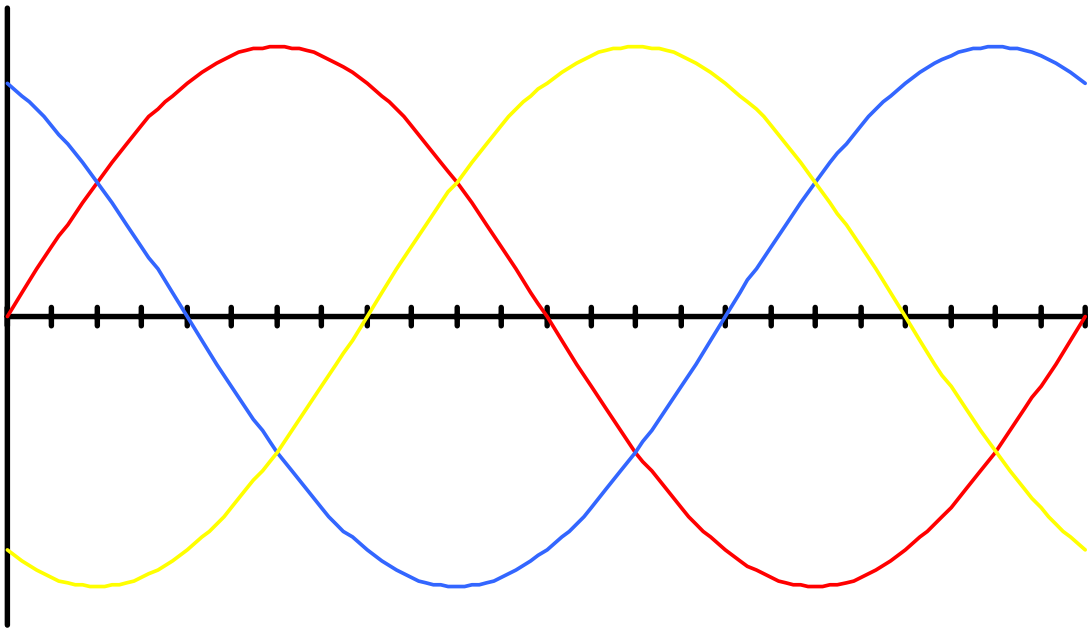
Obviamente todos en radianes.

También debe observarse que en realidad Xp1, Xp2 y Xp3 son los mismos dado que Ang y EscH es común a todo a todos ellos, pero los pusimos por separado para dar mayor claridad.

Lo interesante de haber trabajado con tanta generalidad es que todas las modificaciones que deseemos hacer resultan sumamente sencillas. Imaginemos por ejemplo que nos interese simular un sistema trifásico de tensiones. Con sólo hacer:

Ampl1 = Ampl2 = Ampl3 = 720
Frec1 = Frec2 = Frec3 = 1
Fase1 = 0 Fase2 = 120 Fase3 = 240

todo funciona de maravillas:



También podríamos pensar en graficar los valores absolutos de cada onda (rectificación de onda completa). Ventajoso ¿verdad?

EJES DESFASADOS.

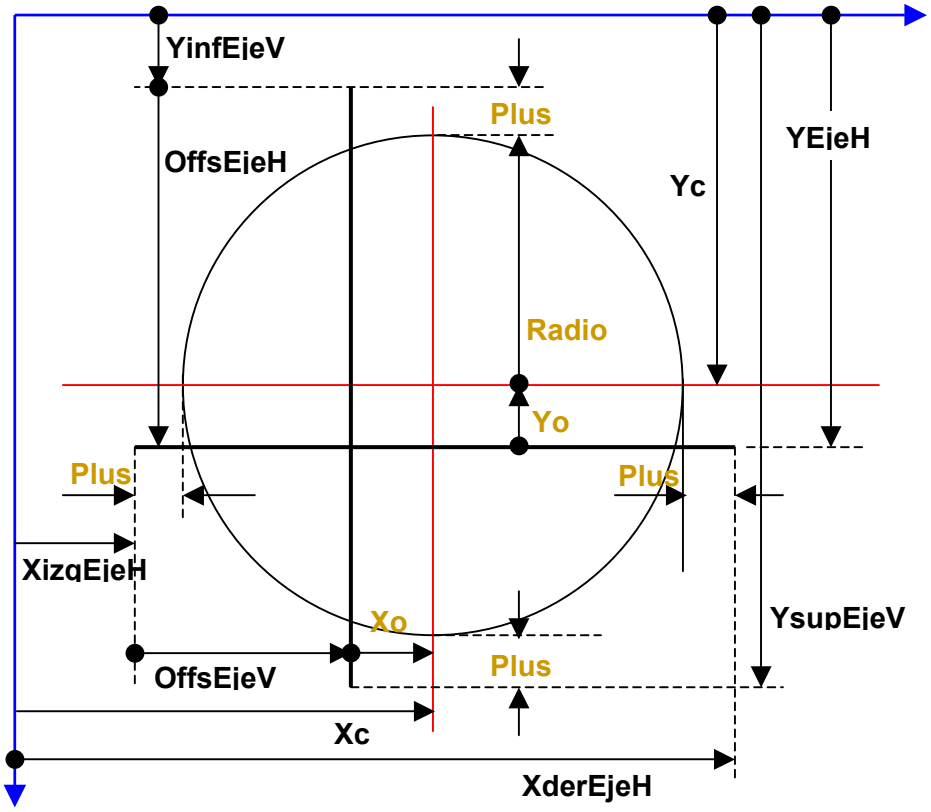
Un caso más complicado para las escalas: la circunferencia.

Como habrá notado, la disponibilidad horizontal para la graficación es mayor que la vertical, lo cual casi siempre implica la existencia de dos escalas. En el caso de la función senoidal anterior hemos trabajado en forma casi transparente con una escala angular (horizontal) y otra escala de amplitudes (vertical), pues aún cuando la función pueda sufrir alguna deformación, su aspecto visual continúa siendo el de una onda senoidal y no caben dudas.

Sin embargo con la circunferencia la historia es diferente: si su radio y emplazamiento es tal que ameriten la determinación de escalas, necesariamente deberemos decidirnos por una escala única, caso contrario aparecería una distorsión en la forma de la función que nos llevaría a pensar que estamos graficando una elipse.

NOTA: En las **escalas de reducción**, el caso más desfavorable es la **escala de menor magnitud.**

Nuestro objetivo inmediato será graficar una circunferencia desplazada con respecto a sus ejes matemáticos, y que posee un radio que exceda los límites de la pantalla gráfica. Como siempre, lo mejor para ubicarnos en el problema es hacer un esquema previo y colocar todas las magnitudes conocidas y las que deben determinarse:



Parece un poco complicado ¿verdad? En realidad no lo es tanto.

Debemos tener mucho cuidado con lo siguiente: **Xo** y **Yo** son coordenadas matemáticas y no de graficación, al igual que **Radio** y **Plus**. Este último sólo cumple la función de que la circunferencia no quede tan "pegada" a los extremos de los ejes matemáticos (los pusimos nosotros por cuenta propia).

Los valores que dicen **OffsEjeH** y **OffsEjeV** se refieren a los desplazamientos necesarios para la intersección de los mismos (obviamente en píxeles):

```
OffsEjeV = EscGraf*(Radio – Xo + Plus);
OffsEjeH = EscGraf*(Radio + Yo + Plus);
```

Esto es importante pues si hubiésemos colocado los ejes como en el caso de las ondas senoidales perderíamos una buena parte de la circunferencia.

En cuanto a la determinación de la escala de graficación, ya dijimos que sale por comparación entre las dos posibles y que se toma la menor (por tratarse de escalas de reducción):

```
EscH    = (double)(XderEjeH - XizqEjeH) / (double)(2*(Radio + Plus));
EscV    = (double)(YsupEjeV - YinfEjeV) / (double)(2*(Radio + Plus));
EscGraf = (EscV < EscH ? EscV : EscH);
```

La última expresión indica un if else implícito en una sola instrucción.
Ahora podemos determinar el punto de intersección de ambos ejes coordenados:

```
YEjeH = YinfEjeV + OffsEjeH;
XEjeV = XizqEjeH + OffsEjeV;
```

todo en píxeles.
En cuanto al lazo de graficación de la función en sí, es el siguiente:

```
for(Ang=AngIni;Ang<AngFin;Ang+=PasoAng) {

   Xp=XEjeV+EscGraf*(Xo+Radio*cos(Ang));
   Yp=YEjeH-EscGraf*(Yo+Radio*sin(Ang));
   putpixel(Xp,Yp,ColorFn);
}
```

donde:

XEjeV + EscGraf*Xo
YEjeH – EscGraf*Yo

es la ubicación en píxeles del centro de la circunferencia.
Aquí hemos utilizado la ecuación paramétrica de la circunferencia, consistente en tomar las proyecciones ortogonales del radio (en función del ángulo, que se toma como variable única):

EscGraf*Radio*cos(Ang) ..... coordenada “x” de pantalla.
EscGraf*Radio*sin(Ang) ..... coordenada “y” de pantalla.

Un detalle: habrá notado que la variable angular recorre un camino entre un valor inicial y un valor final, no necesariamente 0 y 360º. Ello nos permite graficar “arcos” de circunferencia que puede ser muy útil para algunas aplicaciones. A esto nos referimos cuando se pide trabajar en forma “general”.

El código completo para este problema es el siguiente:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<graphics.h>
#include<math.h>

void ModoGrafico ( );
void ModoTexto   ( );


// ------------------------------------------------------------
void main()
{
  int     Radio        = 720;
  int     Xo           = 200;  // con respecto a ejes matemáticos.
  int     Yo           = 150;  // con respecto a ejes matemáticos.
  int     Xc,Yc;               // con respecto a ejes de pantalla.
```

```
int     Plus         =  50;  // para dar un poco de espacio.
int     OffsEjeH;
int     OffsEjeV;

double  Krad         = M_PI/180.00;
double  AngIni       =  45*Krad;
double  AngFin       = 360*Krad;
int     NPts         = 250;  // número de pts con que se graficará.
int     XizqEjeH     =  50;
int     XderEjeH     = 550;
int     YEjeH        = 240;
int     XEjeV        =  50;
int     YinfEjeV     =  50;
int     YsupEjeV     = 450;
int     ColorEjes    = LIGHTCYAN;
int     ColorEjesAux = LIGHTGREEN;
int     NDiv         =  36;
int     Xp,Yp;
double  PasoCal      = (double)(XderEjeH-XizqEjeH)/(double)NDiv;

double  EscH;               // escala horizontal de graficación.
double  EscV;               // escala vertical de graficación.
double  EscGraf;            // escala única adoptada.

double  Ang,Fn;
double  PasoAng      = (double)(AngFin-AngIni)/(double)NPts;

int     ColorFn      = LIGHTRED;
int     ColorCal     = YELLOW;
int     n;

ModoGrafico();

EscH    = (double)(XderEjeH-XizqEjeH)/(double)(2*(Radio+Plus));
EscV    = (double)(YsupEjeV-YinfEjeV)/(double)(2*(Radio+Plus));
EscGraf = (EscV<EscH?EscV:EscH);

OffsEjeV = EscGraf*(Radio-Xo+Plus);
OffsEjeH = EscGraf*(Radio+Yo+Plus);

YEjeH = YinfEjeV+OffsEjeH;
XEjeV = XizqEjeH+OffsEjeV;

// --- TRAZADO DE EJES COORDENADOS -------------

setcolor(ColorEjes);
setlinestyle(SOLID_LINE,1,THICK_WIDTH);
line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);

// --- TRAZADO DE EJES DE LA CIRCUNFERENCIA ---

setlinestyle(SOLID_LINE,1,NORM_WIDTH);
setcolor(ColorEjesAux);
line(XizqEjeH,YEjeH-EscGraf*Yo,XderEjeH,YEjeH-EscGraf*Yo);
line(XEjeV+EscGraf*Xo,YinfEjeV,
     XEjeV+EscGraf*Xo,YsupEjeV);
```

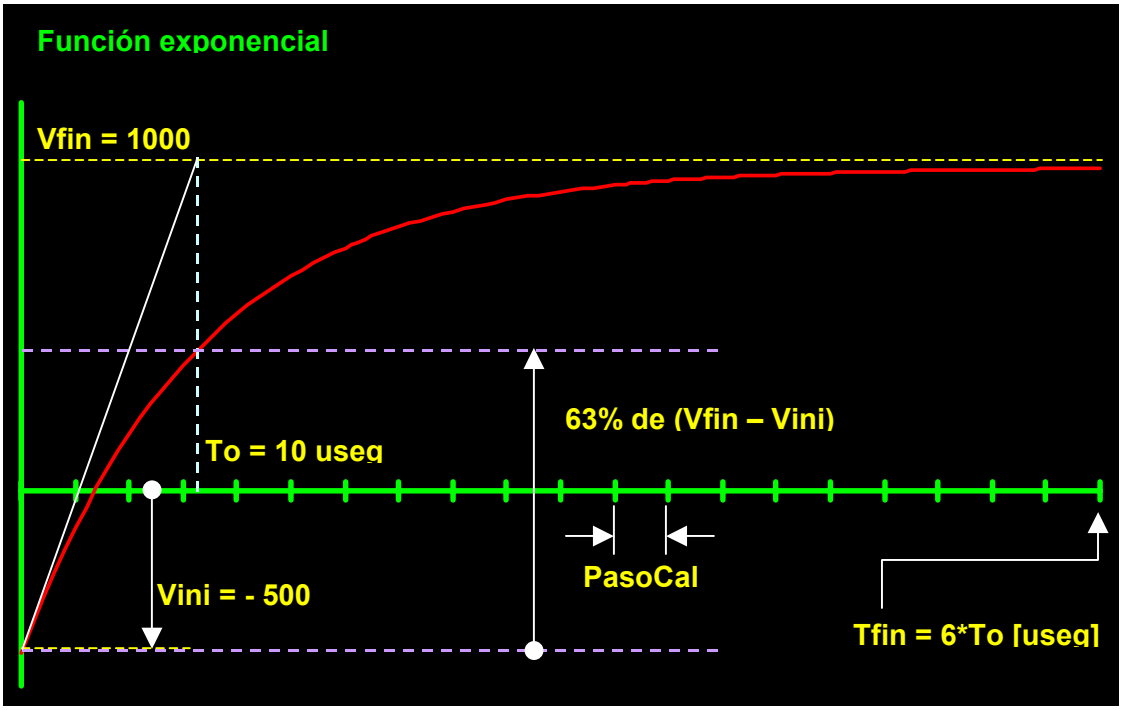
```
  for(Ang=AngIni;Ang<AngFin;Ang+=PasoAng) {

      Xp=XEjeV+EscGraf*(Xo+Radio*cos(Ang));
      Yp=YEjeH-EscGraf*(Yo+Radio*sin(Ang));
      putpixel(Xp,Yp,ColorFn);
  }

  getch();

  ModoTexto();
}
// ------------------------------------------------------------------
```

Graficación de funciones exponenciales.

Las funciones exponenciales resultan de uso muy común en ingeniería ya que muchos fenómenos físicos responden de forma exponencial. Si el lector no está muy versado en tales funciones no debe albergar ningún temor: nosotros le proporcionaremos las herramientas matemáticas y Ud. sólo se limitará al aspecto de la graficación.

Para el ejemplo que viene a continuación utilizaremos una exponencial creciente que arranca de un valor inicial **Vini** y llega hasta un valor final **Vfin**. Su expresión matemática es la siguiente:

F(t) = Vini + (Vfin – Vini)*(1 – exp(-t / To))



To es lo que se denomina la "constante de crecimiento" y es un dato de la ecuación. Es el valor de la variable independiente para el cual la función alcanza el 63,2% de su recorrido de amplitud. No nos preocupemos por este detalle.

"t" es la variable independiente, la cual haremos que varíe entre un valor inicial **Tini** y un valor final **Tfin**, por ejemplo entre 0 y 6 veces **To**.

Esta forma de trabajar con variables para los valores iniciales y finales de ciertos parámetros de la función, es muy conveniente porque nos permite modificar con facilidad las condiciones de graficación y obtener elementos particulares que nos interese. Por supuesto todo debe ajustarse automáticamente para que no ocurran distorsiones.

Por ejemplo:

```
double  EscGrafH  = (double)(XderEjeH-XizqEjeH)/(double)(Tfin-Tini);
double  EscGrafV  = (double)(YsupEjeV-YinfEjeV)/(double)(Vfin-Vini);
```

Ajustan automáticamente las escalas al modificar los valores de cada extreme.
El lazo de graficación en sí:

```
for(t =Tini; t<=Tfin; t+=Paso_t) {
      Xp=XizqEjeH+EscGrafH*(t-Tini);
      Yp=YEjeH-EscGrafV*(Vini+(Vfin-Vini)*(1-exp(-t/To)));
      putpixel(Xp,Yp,ColorFn);
}
```

también tiene en cuenta los cambios de valores iniciales y finales de los parámetros de graficación. Comprenderemos mejor esto luego de tipear y hacer correr el programa.

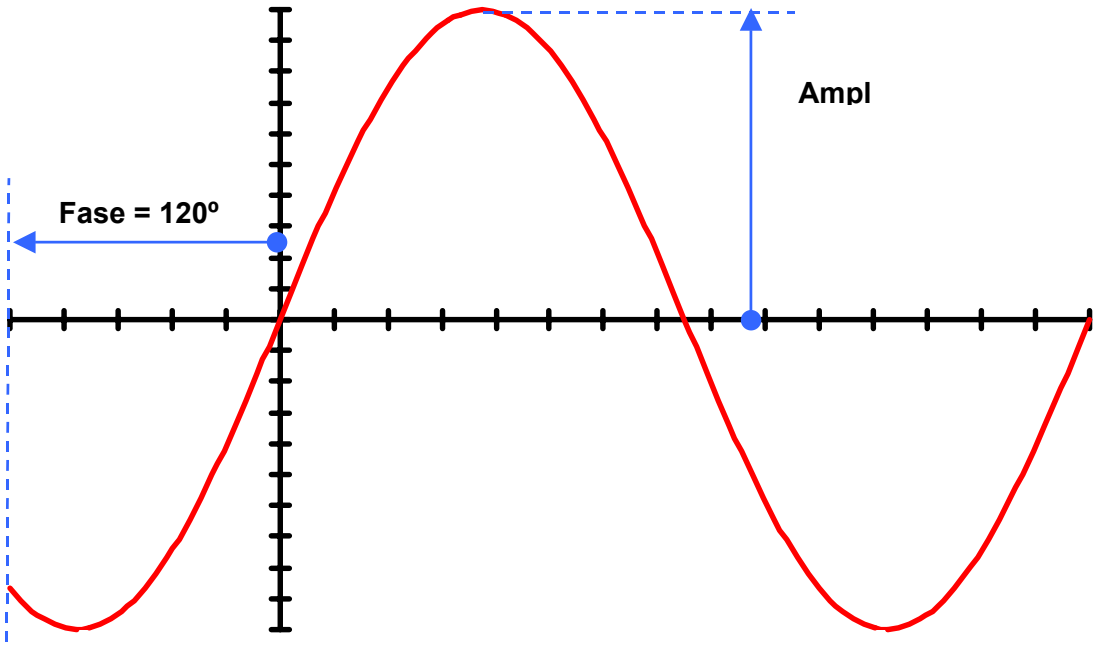
```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (  );
void ModoTexto   (  );

// ---------------------------------------------------------------------
void main()
{

  int     Vfin      = 1000;
  int     Vini      = -500;
  int     XizqEjeH  =   50;
  int     XderEjeH  =  550;
  int     YEjeH     =  250;
  int     XEjeV     =   50;
  int     YinfEjeV  =   50;
  int     YsupEjeV  =  400;
  int     Ndiv      =   20;
  int     ColorEjes = GREEN;
  int     ColorCal  = YELLOW;
  int     ColorFn   = LIGHTRED;
  int     Npts      =  500;
  int     Xp,Yp;
  int     i;

  double  To        = 10;
  double  Tini      = 0;
  double  Tfin      = 6*To;
```

```
  double  EscGrafH  = (double)(XderEjeH-XizqEjeH)/(double)(Tfin-Tini);
  double  EscGrafV  = (double)(YsupEjeV-YinfEjeV)/(double)(Vfin-Vini);
  double  PasoCal   = (double)(XderEjeH-XizqEjeH)/(double)Ndiv;
  double  EscUs     = (Tfin-Tini)/(double)Ndiv;
  double  Paso_t    = (Tfin-Tini)/Npts;
  double  t;

  char    EscUsStr[32];

  ModoGrafico();

  // --- TRAZADO DE EJES -------------------------------------

  setcolor(ColorEjes);
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);


  // --- TRAZADO DE CALIBRACION ------------------------------
  setcolor(ColorCal);
  for(i=0;i<=Ndiv;i++)
    line(XizqEjeH+i*PasoCal,YEjeH-3,XizqEjeH+i*PasoCal,YEjeH+3);


  // --- MENSAJES DE PANTALLA --------------------------------

  setcolor(LIGHTGREEN);
  sprintf(EscUsStr,"ESC=%2.2f[useg/div]",EscUs);
  outtextxy(60,50,"FUNCION EXPONENCIAL");
  outtextxy(60,60,EscUsStr);

  // --- TRAZADO DE LA FUNCION EXPONENCIAL -------------------

  for(t=Tini;t<=Tfin;t+=Paso_t) {

      Xp=XizqEjeH+EscGrafH*(t-Tini);
      Yp=YEjeH-EscGrafV*(Vini+(Vfin-Vini)*(1-exp(-t/To)));
      putpixel(Xp,Yp,ColorFn);
  }

  getch(); ModoTexto();

}
// ---------------------------------------------------------------------
```

A simple vista (y en forma aproximada) podemos ver que la función corta al eje del tiempo en t = 3,80 ó t = 4,00. ¿Queremos salir de dudas sin modificar el código del programa? Bueno, hagamos entonces **Tini** = 3.80 y veamos qué pasa. La función debería arrancar desde el origen. Si no fue así probemos con algunos valores cercanos y tendremos una idea bastante buena del punto de corte.

Otro caso de ejes desplazados.

Dibujar una función senoidal de manera que el eje vertical se halle ubicado de tal manera que la función arranque en -120 grados y se extienda hasta +360 grados.



---------- */

Para hallar la coordenada horizontal **XEjeV** del eje vertical, es necesario determinar primero la escala horizontal de graficación:

```
EscGrafH = (double)(XderEjeH-XizqEjeH) / (AngFin-AngIni)
```

Esta escala relaciona cada medida angular a graficar (ya sea en radianes o en grados, según como hayamos calculado la misma), con los pixeles necesarios:

```
XEjeV = XizqEjeH + EscGrafH*Fase
```

Con respecto a la graficación en sí de la función, tenemos dos posibilidades de realizarla:

```
for(AngRad=AngIniRad;AngRad<=AngFinRad;AngRad+=PasoAngRad) {
        // --- forma I -------------------------------------
        Xp = XEjeV + round(EscGrafH*AngRad);

        // --- forma II ------------------------------------------------------
        // Xp = XizqEjeH + round(EscGrafH*(AngRad-AngIniRad));

        Yp = YEjeH-round(EscGrafV*Ampl*sin(AngRad));
        putpixel(Xp,Yp,ColorFn);
}
```

ya sea tomando el extremo izquierdo del eje horizontal o la posición horizontal del eje vertical. En ambos casos el resultado es idéntico. Observe que en el caso de tomar el extremo izquierdo del eje horizontal tenemos que restarle el valor inicial del ángulo por las razones que expusimos al comienzo de este capítulo:

Xp = XizqEjeH + round(EscGrafH*(AngRad-AngIniRad));

Pasemos al código completo:

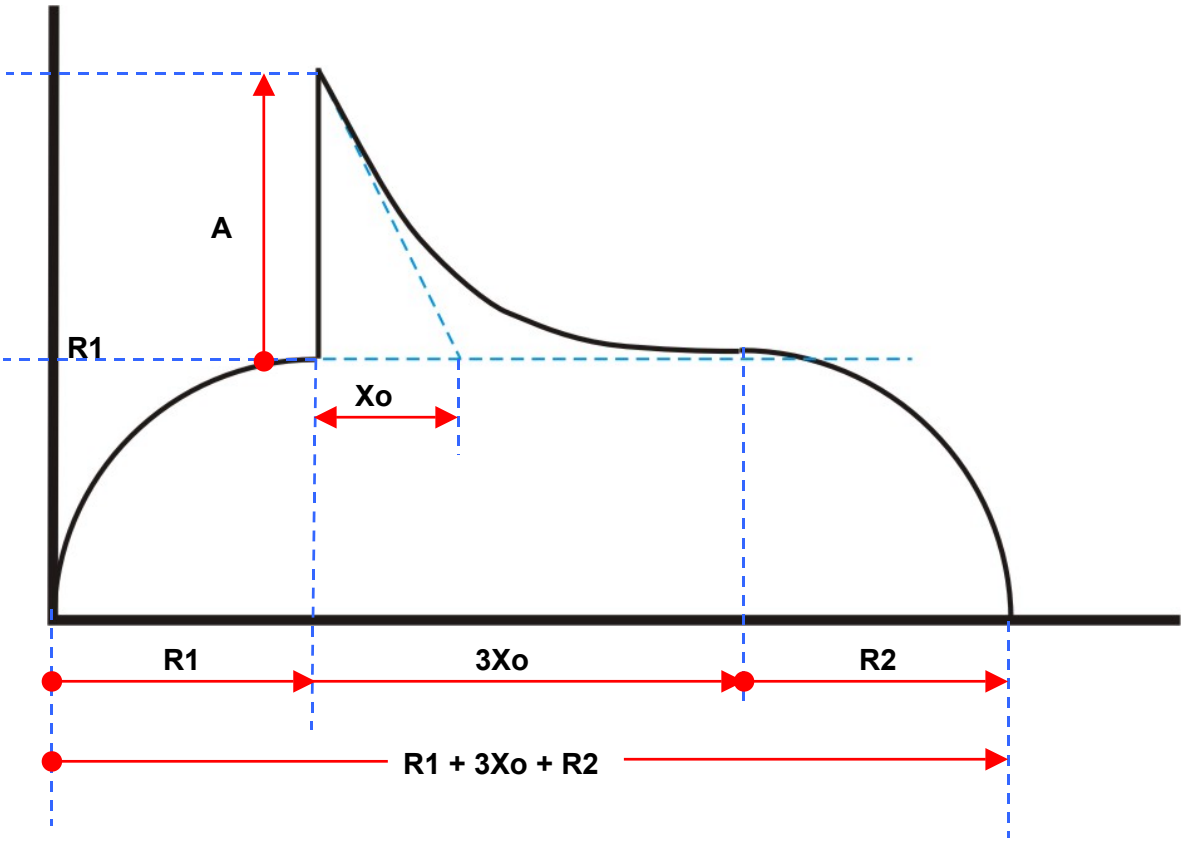
```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (        );
void ModoTexto   (        );
int  round       ( double );

// ---------------------------------------------------------------------
void main()
{

  int        Ampl = 2500;   // amplitud de la funcion senoidal.
  int        Plus =  500;   // solo para dar espacio de graficacion.
  int   AngIniGrd = -120;
  int   AngFinGrd =  360;
  int    XizqEjeH =   50;
  int    XderEjeH =  550;
  int       YEjeH =  240;
  int       XEjeV;         // sera calc.segun el desplazam. necesario.
  int    YinfEjeV =   50;

  int    YsupEjeV =  430;
  int   ColorEjes =  GREEN;
  int    ColorCal =  DARKGRAY;
  int     ColorFn =  LIGHTRED;
  int      NdivsH =   24;
  int      NdivsV =   12;
  int        Npts = 1000;
  int       Xp,Yp;
  int          n;

  double  AngRad;
  double  Krad       = M_PI/180.00;
  double  AngIniRad  = AngIniGrd*Krad;
  double  AngFinRad  = AngFinGrd*Krad;
  double  PasoAngRad = (AngFinRad-AngIniRad)/Npts;
  double  PasoCalH;
  double  PasoCalV   = (double)(YEjeH-YinfEjeV)/(double)NdivsV;
  double  EscGrafH   = (double)(XderEjeH-XizqEjeH)/(AngFinRad-
                                AngIniRad);
  double  EscGrafV   = (double)(YEjeH-YinfEjeV)/(double)(Ampl+Plus);
  double  EscUsAmpl  = (Ampl+Plus)/NdivsV;
  double  EscUsAng   = (double)360/(double)NdivsH;

  char    EscUsAmplStr[32];
```

```
  char    EscUsAngStr[32];


  ModoGrafico();

  // --- TRAZADO DE EJES COORDENADOS -----------------------

  XEjeV=XizqEjeH+fabs(EscGrafH*AngIniRad);

  setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);   // eje horizontal.
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);   // eje vertical.
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);


  // --- CALIBRACION DE LOS EJES ----------------------------

  setcolor(ColorCal);
  PasoCalH=(double)(XderEjeH-XEjeV)/(double)NdivsH;

  // parte positiva del eje horizontal ----------------------
  for(n=1;n<=(XderEjeH-XEjeV)/PasoCalH;n++)
      line( XEjeV+round(n*PasoCalH),YinfEjeV,
          XEjeV+round(n*PasoCalH),YsupEjeV);

  // parte negativa del eje horizontal ----------------------

  for(n=1;n<=(XEjeV-XizqEjeH)/PasoCalH;n++)
      line( XEjeV-round(n*PasoCalH),YinfEjeV,
          XEjeV-round(n*PasoCalH),YsupEjeV);

  // eje vertical -------------------------------------------


  for(n=1;n<=NdivsV;n++) {
      line(XizqEjeH,YEjeH-n*PasoCalV,XderEjeH,YEjeH-n*PasoCalV);
      line(XizqEjeH,YEjeH+n*PasoCalV,XderEjeH,YEjeH+n*PasoCalV);
  }


  // --- MENSAJES DE PANTALLA -----------------------------

  setcolor(RED);
  outtextxy(350,60,"EJE DESPLAZADO");

  sprintf(EscUsAmplStr,"EscAmpl = %6.2lf [1/div]",EscUsAmpl);
  sprintf(EscUsAngStr, "EscAng  = %6.2lf [§/div]",EscUsAng);

  setcolor(LIGHTGRAY);
  outtextxy(350,80,EscUsAmplStr);
  outtextxy(350,90,EscUsAngStr);

  setcolor(LIGHTGRAY);
  outtextxy(350,440,"PROGRAMADOR UNIVERSITARIO");
  outtextxy(350,450,"TALLER DE LENGUAJES I");
  outtextxy(350,460,"UNIV.NACIONAL DE TUCUMAN");
```

```
  // --- GRAFICACION DE LA FUNCION -------------------------

  for(AngRad=AngIniRad;AngRad<=AngFinRad;AngRad+=PasoAngRad) {

      Xp=XEjeV+round(EscGrafH*AngRad);

   // con instrucc que viene a continuacion se obtiene lo mismo:
   // Xp=XizqEjeH+round(EscGrafH*(AngRad-AngIniRad));

      Yp=YEjeH-round(EscGrafV*Ampl*sin(AngRad));
      putpixel(Xp,Yp,ColorFn);
  }

  getch(); ModoTexto();

}

// ---------------------------------------------------------------------
int round(double x)
{ if((x-(int)x)>=0.0) return((int)(x+1)); return((int)x); }
// ---------------------------------------------------------------------
```

CAPÍTULO 3 – *Funciones definidas por tramos.*

Puede ocurrir que en un recorrido dado de la variable independiente, tengamos definidas varias funciones diferentes. Por ejemplo:



(donde R2 es el valor que toma la exponencial al final de su recorrido).

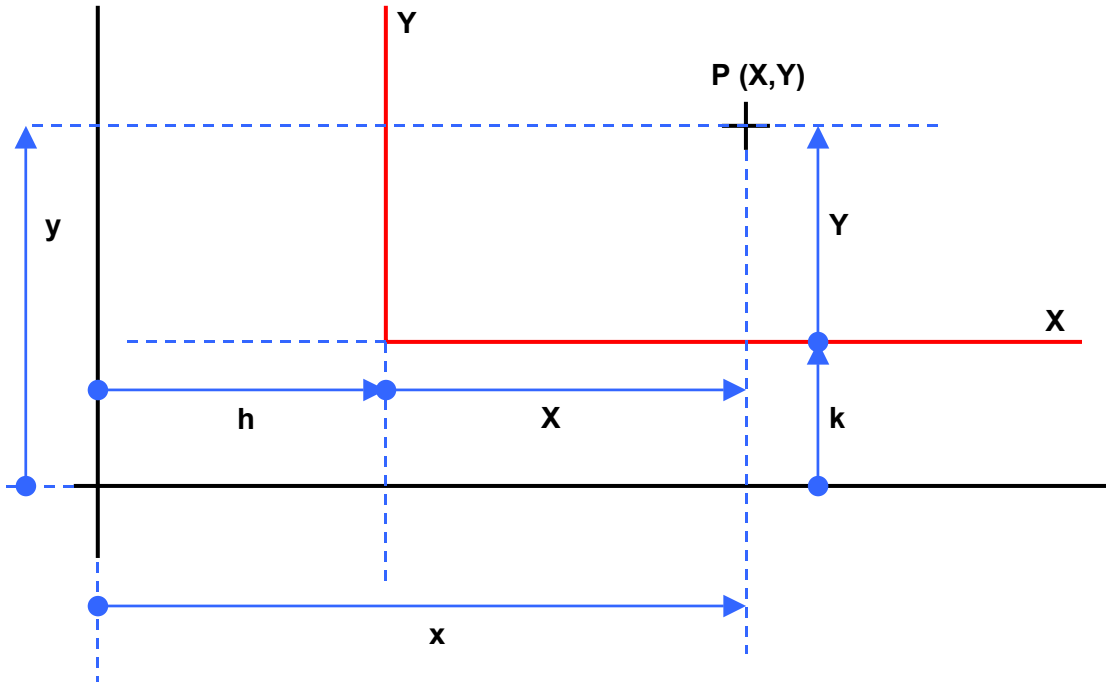
La gráfica está compuesta por 3 figuras distintas:

1. **Un arco de circunferencia.**
2. **Una exponencial decreciente desplazada.**
3. **Otro arco de circunferencia, desplazado.**

Esto nos indica que si consideramos una variable x que tome valores continuos en el rango comprendido entre **0** y **[R1 + 3Xo + R2**], deberemos tener en cuenta:

- La ecuación de cada figura según su desplazamiento.
- El rango de validez de cada ecuación.
- Chequear permanentemente **x** para determinar cuándo comienza o deja de graficarse cada función.

En base a esto sería una muy buena idea repasar un poco cómo se determina la ecuación de una función desplazada del origen.

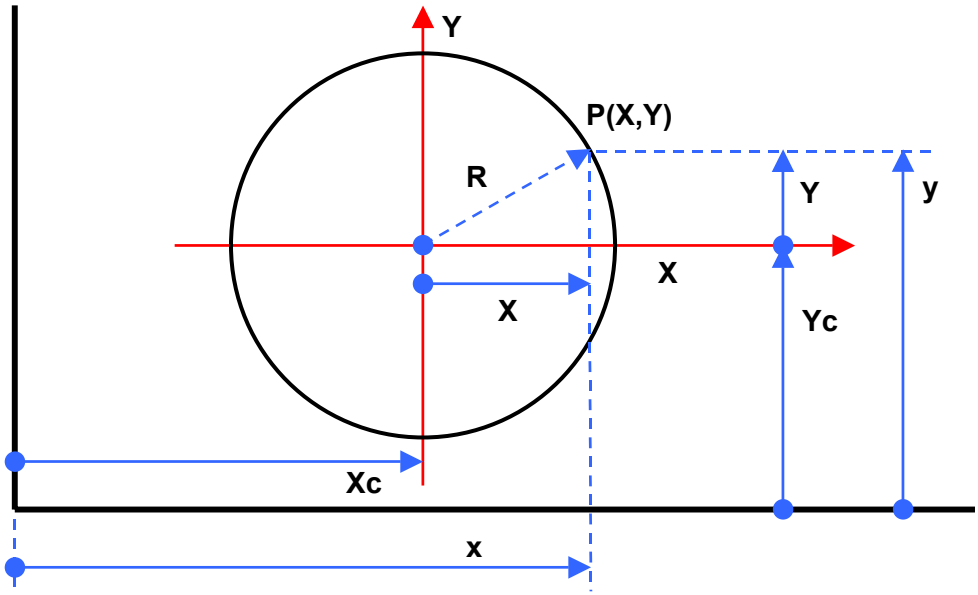


El punto genérico **P** posee coordenadas (**X,Y**) con respecto al sistema de ejes en rojo. Nuestra idea es averiguar qué coordenadas le corresponden en el sistema xy en negro.

"**h**" y "**k**" son las magnitudes de desplazamiento del sistema de ejes interior, con lo cual ya pueden determinarse las nuevas coordenadas buscadas:

X = x – h
Y = y – k

Si bien el punto **P** se halla aparentemente aislado, puede pertenecer a cualquier figura plana. Por ejemplo puede ser un punto genérico de una circunferencia:



La ecuación de la circunferencia con respecto a los ejes propios (en Rojo), viene dada por:

$X^2 + Y^2 = R^2$
además:

h = Xc
k = Yc

y aplicando la traslación:

X = x – Xc
Y = y – Yc

llegamos a: **$(x – Xc)^2 + (y – Yc)^2 = R^2$**

O bien:

x = Xc + R.cos(Fi)
y = Yc – R.sin(Fi) en forma paramétrica.

Claro que en forma paramétrica es más sencilla, pero no siempre resulta posible trabajar de esta forma, como veremos en breve. Las fórmulas de traslación son generales y se ajustan automáticamente para cualquier cuadrante, simplemente con reemplazar en su momento los desplazamientos de los ejes con sus signos correctos.

Ahora la pregunta por el millón: ¿Cómo sabemos que hemos llegado a una ecuación correcta de traslación? Igual que en el caso de las escalas y los valores extremos de ejes ¿recuerda? Por ejemplo demos a “x” valores característicos y analicemos qué obtenemos:

x = Xc

$$y = Yc + \sqrt{R^2 - (x - Xc)^2}$$

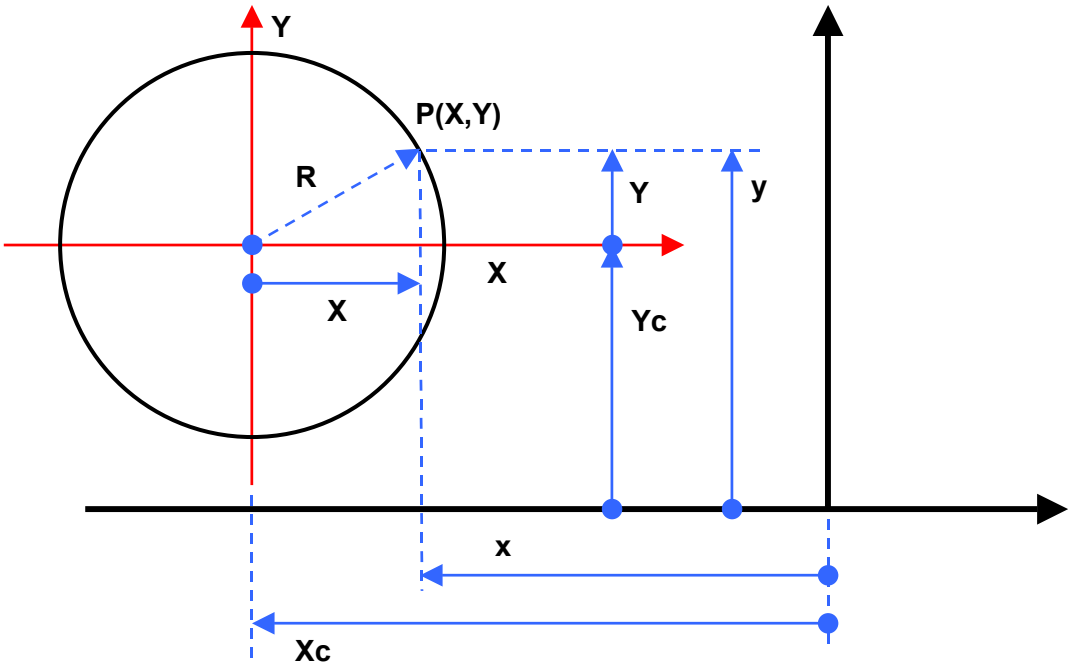
con la cual se obtienen dos valores:

y1 = Yc + R
y2 = Yc – R

que concuerdan perfectamente con lo esperado. Si hiciéramos x = Xc+R tendríamos:

$y = Yc + \sqrt{R^2 - (Xc + R - Xc)^2} = Yc$ también coherente con la realidad.

¿Y si los ejes de la circunferencia estuvieses al revés?



La ecuación de la circunferencia sobre sus ejes propios continúa siendo la misma:

$X^2 + Y^2 = R^2$

X = Xc – x
Y = y – Yc

(Xc – x)2 + (y – Yc)2 = R2 $y = Yc + \sqrt{R^2 - (Xc - x)^2}$

Si volvemos a probar con valores característicos tendremos:

x = Xc $y = Yc + \sqrt{R^2 - (Xc - Xc)^2} = Yc \pm R$ que coincide bien.

¿No lo ve bien porque el signo no quedó muy claro?
Entonces supongamos Xc = -10 . Ahora reemplacemos x = Xc

$$y = Yc + \sqrt{R^2 - (-10 - (-10))^2} = Yc \pm R$$

que nos da el mismo resultado.
De la misma forma podríamos probar en los otros dos cuadrantes, o con otras funciones tales como exponenciales, rectas, etc. Un detalle muy importante es determinar el "Rango" de la variable independiente para que no haya sorpresa de raíces negativas, denominador cero o alguna otra singularidad extraña.

Para nuestra ya familiar circunferencia este rango pude obtenerse por simple inspección o bien en forma determinística:

$y = Yc + \sqrt{R^2 - (x - Xc)^2}$ de donde se deduce que la raíz debe ser siempre:

$$\sqrt{R^2 - (x - Xc)^2} \geq 0$$

O lo que es lo mismo:

$$R^2 - (x - Xc)^2 \geq 0$$

El próximo paso será:

$(x - Xc)^2 \geq R^2$ y extrayendo las dos raíces:

$x - Xc = \pm R$ con lo cual finalmente se deduce que el rango de **x** viene dado por:

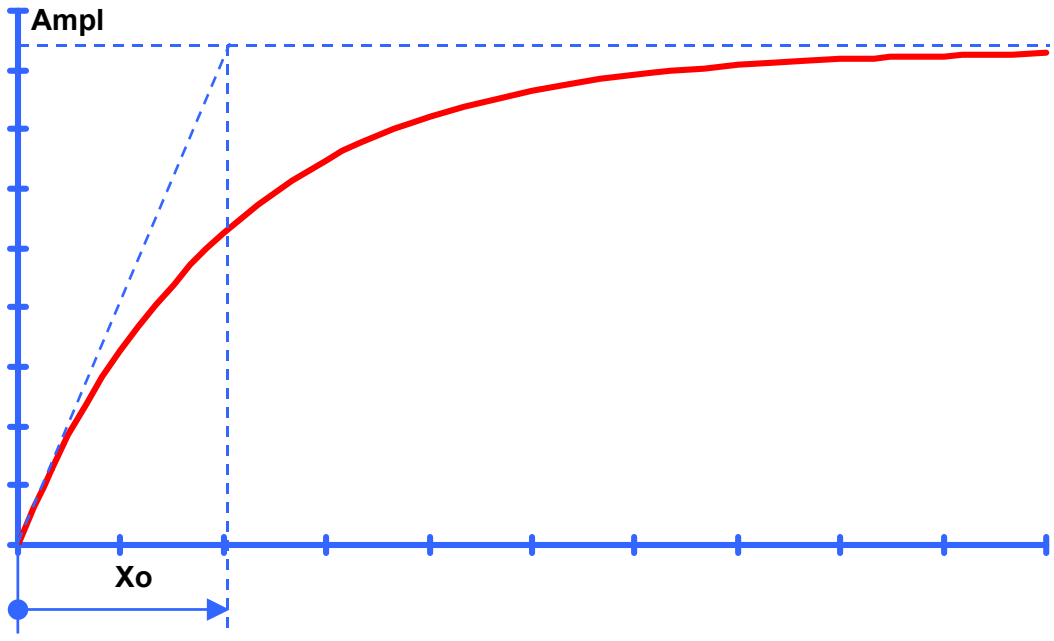
$$x \in [Xc - R, Xc + R]$$

Este concepto debemos tenerlo bien claro, ya que al trabajar con funciones definidas por tramos resulta importante saber para qué rango de **x** tiene validez cada una.

Funciones exponenciales.

Tenemos dos tipos de funciones exponenciales clásicas:

- **Exponencial creciente.**
- **Exponencial decreciente.**



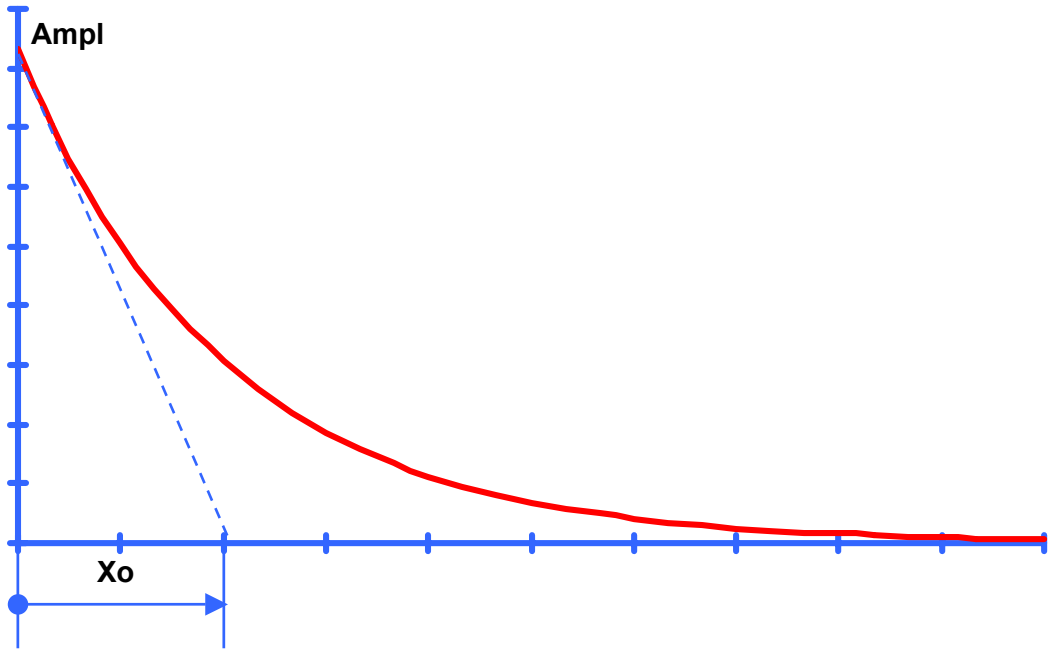
La ecuación de esta función viene dada por:

F(x) = Ampl*(1- exp(-x/Xo))

en la cual **Xo** es una magnitud fija que se denomina **constante de crecimiento** y es un valor particular de **x** que hace que la función adquiera el 63% de su valor final. Siempre es bueno analizar nuestras expresiones con algunos valores conocidos para ver si la misma se halla bien escrita. En este caso podríamos considerar:

Para x = 0 → F(0) = Ampl*(1- exp(-0/Xo)) = 0
Para x = infinito → F(inf) = Ampl*(1- 0) = Ampl

O sea que todo anduvo bien.
Si la exponencial fuese decreciente:



Su ecuación sería:

F(x) = Ampl*exp(-x/Xo)

Y ahora **Xo** sería una magnitud fija que haría que la función decaiga en un 63% de su valor inicial.
Ambas ecuaciones están, obviamente, referidas a sus ejes naturales. En el caso de existir un desplazamiento todo lo que tendríamos que hacer es:

X = x – h
Y = y - k

y reemplazar en las expresiones anteriores:

y - k = Ampl*exp(-(x-h)/Xo))
y – k = Ampl*(1- exp(-(x-h)/Xo))

Según sea el desplazamiento, los signos de **h** y **k** colocarán a la ecuación anterior en su punto correcto.

Considerando estas fórmulas desplazadas y los valores reales de h y k, podemos determinar la ecuación de cada tramos de la curva de partida;

$F(x) = \sqrt{R_1^2 - (x - R_1)^2}$ para todo x entre 0 ... R1

$F(x) = R_1 + A * e^{-(x-R_1)/X_o}$ para todo x entre R1 ... R1+3*Xo

$F(x) = \sqrt{R_2^2 - (x - (R_1 + 3X_o)^2}$ para todo x entre [R1+3Xo ... R1+3Xo+R2]

Ya estamos en condiciones de generar el código en "C":

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (        );
void ModoTexto   (        );
int  round       ( double );

// ---------------------------------------------------------------------
void main()
{
  int           A =  800;
  int          R1 = 1000;
  int          Xo =  R1/2;
  int    XizqEjeH =  100;
  int    XderEjeH =  500;
  int       YEjeH =  350;
  int       XEjeV =  XizqEjeH;
  int    YinfEjeV =  100;
  int    YsupEjeV =  YEjeH;
  int        Npts =  500;
  int     ColorFn =  RED;
  int   ColorEjes =  GREEN;
  int    ColorCal =  YELLOW;
  int    Xp,Yp,n;


  double      x,y;
  double       R2 = R1+A*exp(-((R1+3*Xo)-R1)/Xo);
  double    x_ini = 0;
  double    x_fin = R1+3*Xo+R2;
  double   Paso_x = (x_fin-x_ini)/Npts;
  double  EscGraf;
  double EscGrafH = (double)(XderEjeH-XEjeV)/(x_fin-x_ini);
  double EscGrafV = (double)(YsupEjeV-YinfEjeV)/(double)(R1+A);
  double PasoCalH;
  double    EscUs;

  char   EscUsStr[32];


  ModoGrafico();
```

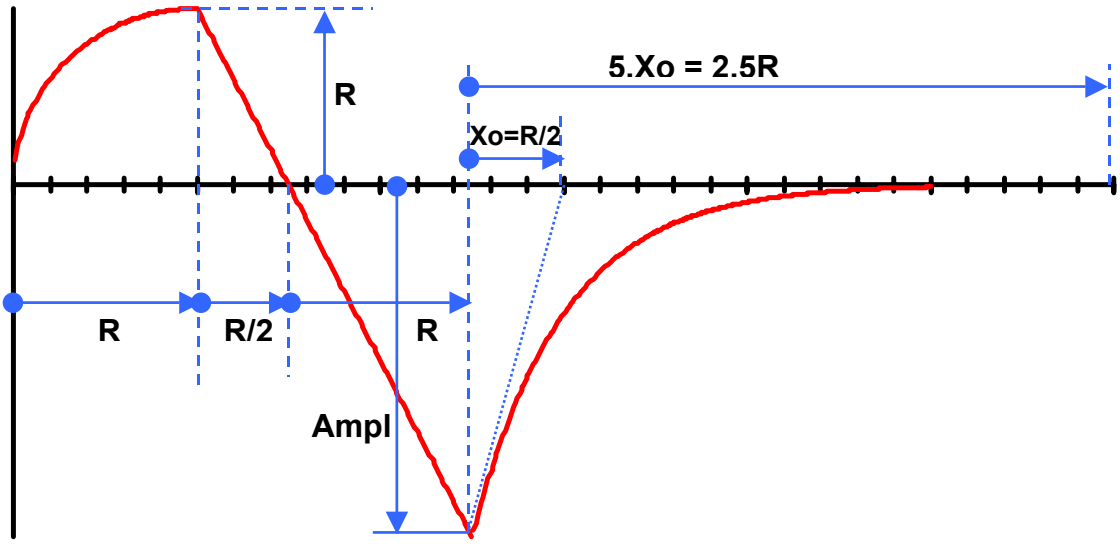
```
EscGraf  = (EscGrafV<EscGrafH?EscGrafV:EscGrafH);
PasoCalH = (EscGraf*R1)/(double)5;
EscUs    = (double)R1/(double)5;


// --- TRAZADO DE EJES -------------------------

setcolor(ColorEjes);
setlinestyle(SOLID_LINE,1,THICK_WIDTH);
line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
setlinestyle(SOLID_LINE,1,NORM_WIDTH);

// --- TRAZADO DE LA CALIBRACION ---------------------------

// sobre el eje horizontal ------

for(n=0;n<round((double)(XderEjeH-XEjeV)/PasoCalH);n++) {
    setcolor(DARKGRAY);
    line(XEjeV+n*PasoCalH,YEjeH,XEjeV+n*PasoCalH,YinfEjeV);
    setcolor(ColorCal);
    line(XEjeV+n*PasoCalH,YEjeH-4,XEjeV+n*PasoCalH,YEjeH+4);
}

// sobre el eje vertical --------


for(n=0;n<=round((YEjeH-YinfEjeV)/PasoCalH);n++) {
    setcolor(DARKGRAY);
    line(XEjeV,round(YEjeH-n*PasoCalH),
       XderEjeH,round(YEjeH-n*PasoCalH));
    setcolor(ColorCal);
    line(XEjeV-4,round(YEjeH-n*PasoCalH),
       XEjeV+4,round(YEjeH-n*PasoCalH));
}
// --- MENSAJES DE PANTALLA --------------------

sprintf(EscUsStr,"Esc=%2.2lf [1/div]",EscUs);

setcolor(LIGHTGRAY);
outtextxy(XEjeV,80,"FUNCIONES POR TRAMOS");
outtextxy(XEjeV,YsupEjeV+10,EscUsStr);

setcolor(BROWN);
outtextxy(350,440,"PROGRAMADOR UNIVERSITARIO");
outtextxy(350,450,"TALLER DE LENGUAJES I");
outtextxy(350,460,"UNIV.NACIONAL DE TUCUMAN");

// --- TRAZADO DE LA FUNCION POR TRAMOS --------

for(x=x_ini;x<=x_fin;x+=Paso_x) {
    Xp=XEjeV+EscGraf*x;

    if(x<R1) y=sqrt(pow(R1,2)-pow(x-R1,2));
    else if((x-R1)<=Paso_x) {
           setcolor(ColorFn);
           line(Xp,YEjeH-EscGraf*(A+R1),Xp,YEjeH-EscGraf*R1);
       }
    else if(x<=(R1+3*Xo)) y=R1+A*exp(-(x-R1)/Xo);
    else y=sqrt(pow(R2,2)-pow(x-(R1+3*Xo),2));
```

```
      Yp=YEjeH-EscGraf*y;

      putpixel(Xp,Yp,ColorFn);
  }
  getch(); ModoTexto();
}
// ---------------------------------------------------------------------
int round(double x)
{ if((x-(int)x)>=0.5) return((int)(x+1)); return((int)x); }
// ---------------------------------------------------------------------
```

Cuando ejecute este programa notará que el dibujo se hallará inmerso en una grilla cuyos bordes, derecho y superior, no se cierran por completo. Ello se debe a que en lugar de tomar el segmento horizontal de graficación **XderEjeH – XizqEjeH**, hemos considerado el segmento correspondiente a R1: **EscGraf*R1** y hemos dividido al mismo en 5 unidades, con lo cual determinamos el **PasoCalH**. Luego utilizamos esta magnitud para completar la rejilla:

```
for(n=0;n<round((double)(XderEjeH-XEjeV)/PasoCalH);n++) {
for(n=0;n<=round((YEjeH-YinfEjeV)/PasoCalH);n++) {
```

Otro ejemplo con una exponencial negativa:



Considere **R = 500**
La variable independiente **x** varía en forma continua entre 0 y 5.R, la cual formará parte de un lazo for( ). Sin embargo Ud. tiene diferentes funciones dentro de este rango de valores:

Arco de Circunferencia	**[0 .. R]**
Segmento de recta	**[R .. 2,5R]**
Exponencial	**[2,5R .. 5R]**

El eje horizontal dispondrá de 20 divisiones de calibración.

El semieje vertical positivo tendrá 5 divisiones con las cuales deberá determinar el **PasoCalV** y con él establecer la cantidad en el resto de los ejes y dibujarlas.

Además se indicará en pantalla la **EscUsH** y la **EscUsV** para poder verificar visualmente si todo funcionó bien.

Dibujaremos los ejes en triple espesor y aparte de la calibración crearemos una grilla en color **DARKGRAY** (tipo Excel). **NOTA**: primero debe dibujarse la grilla y luego la calibración para que ésta resalte y no quede opacada por la grilla. Recuerde las fórmulas con desplazamiento de ejes:

$y = k + raiz(R^2 – (x – h)^2)$
$y = k + Ampl.exp(-(x-h)/Xo)$
$y = m.x - 3R$ (con m = -2)

¿Serán correctas? ¡Verifíquelas!

IMPORTANTE: Observe que el Eje horizontal no se halla en el punto medio del eje vertical sino ubicado proporcionalmente según las medidas de las figuras.

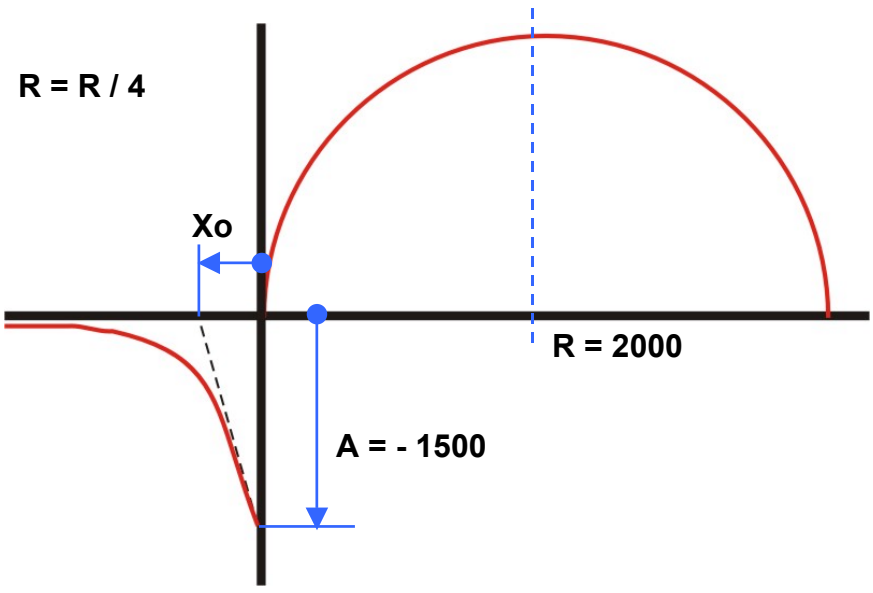
El código para resolver esta gráfica es el siguiente:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (         );
void ModoTexto   (         );
int  round       (double x );
// ------------------------------------------------------------
void main()
{
  int     XizqEjeH    = 100;
  int     XderEjeH    = 550;
  int     YEjeH;
  int     XEjeV       = 100;
  int     YinfEjeV    =  50;
  int     YsupEjeV    = 430;
  int     NdivH       =  20;
  int     NdivV       =   5;
  int     Npts        = 1000;
  int     ColorEjes   = GREEN;
  int     ColorCal    = YELLOW;
  int     ColorFn     = LIGHTRED;
  int     Xp,Yp;
  int     n;

  double  R         =  500;
  double  Xo        =  R/2;
  double  Xini      =  0;
  double  Xfin      =  5*R;
  double  Paso_x    = (Xfin-Xini)/Npts;
  double  PasoCalH  = (double)(XderEjeH-XizqEjeH)/(double)NdivH;
```

```
double  PasoCalV;
double  EscGrafH  = (double)(XderEjeH-XizqEjeH)/(Xfin-Xini);
double  EscGrafV  = (double)(YsupEjeV-YinfEjeV)/(3*R);
double  EscGraf;
double  x,y;
double  EscUsH    = (Xfin-Xini)/NdivH;
double  EscUsV    = R/NdivV;

char    EscUsHStr[32];
char    EscUsVStr[32];

ModoGrafico();

EscGraf=(EscGrafV<EscGrafH?EscGrafV:EscGrafH);

// --- TRAZADO DE EJES COORDENADOS -----------------------------

YEjeH=YinfEjeV+round(EscGraf*R);
YsupEjeV=YinfEjeV+EscGraf*3*R;

setcolor(ColorEjes);
setlinestyle(SOLID_LINE,1,THICK_WIDTH);
line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
setlinestyle(SOLID_LINE,1,NORM_WIDTH);

// --- TRAZADO DE CALIBRACION ----------------------------------

PasoCalV=(double)(YEjeH-YinfEjeV)/(double)NdivV;

for(n=0;n<=NdivH;n++) {
    setcolor(DARKGRAY);
    line(XizqEjeH+round(n*PasoCalH),YinfEjeV,
       XizqEjeH+round(n*PasoCalH),YsupEjeV);
    setcolor(ColorCal);
    line(XizqEjeH+round(n*PasoCalH),YEjeH-4,
       XizqEjeH+round(n*PasoCalH),YEjeH+4);
}

// --- semieje superior vertical --------

for(n=0;n<=5;n++) {
    setcolor(DARKGRAY);
    line(XEjeV,YEjeH-round(n*PasoCalV),
       XderEjeH,YEjeH-round(n*PasoCalV));
    setcolor(ColorCal);
    line(XEjeV-4,YEjeH-round(n*PasoCalV),
       XEjeV+4,YEjeH-round(n*PasoCalV));
}

// --- semieje inferior vertical --------

for(n=0;n<=(round((EscGraf*2*R)/PasoCalV));n++) {
    setcolor(DARKGRAY);
    line(XEjeV,YEjeH+round(n*PasoCalV),
       XderEjeH,YEjeH+round(n*PasoCalV));
    setcolor(ColorCal);
    line(XEjeV-4,YEjeH+round(n*PasoCalV),
       XEjeV+4,YEjeH+round(n*PasoCalV));
}
```

```
  // --- MENSAJES DE PANTALLA -----------------------------------

  sprintf(EscUsHStr,"EscH=%2.2lf",EscUsH);
  sprintf(EscUsVStr,"EscV=%2.2lf",EscUsV);

  setcolor(LIGHTGRAY);
  outtextxy(XEjeV,20,"FUNCIONES POR TRAMOS");
  outtextxy(XEjeV,YsupEjeV+10,EscUsHStr);
  outtextxy(XEjeV,YsupEjeV+20,EscUsVStr);

  setcolor(LIGHTGRAY);
  outtextxy(350,440,"PROGRAMADOR UNIVERSITARIO");
  outtextxy(350,450,"TALLER DE LENGUAJES I");
  outtextxy(350,460,"UNIV.NACIONAL DE TUCUMAN");

  // --- TRAZADO DE LA FUNCION EN SI ----------------------------

  for(x=Xini;x<=Xfin;x+=Paso_x) {

      Xp=round(XEjeV+EscGraf*(x-Xini));

      if(x<=R)          y = sqrt(pow(R,2)-pow(x-R,2));
      else if(x<R*2.5)  y = -2*x+3*R;
      else              y = -2*R*exp(-(x-2.5*R)/Xo);

      Yp=round(YEjeH-EscGraf*y);
      putpixel(Xp,Yp,ColorFn);
      delay(10);
  }
  getch(); ModoTexto();
}
// ------------------------------------------------------------------
```

Otro ejemplo interesante de funciones por tramos:



```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (        );
void ModoTexto   (        );
int  round       ( double );
// ---------------------------------------------------------
void main()
{
  int          R = 2000;
  int         Xo = -R/4;
  int          A = 1500;
  int   XizqEjeH = 100;
  int   XderEjeH = 550;
  int      YEjeH;
  int      XEjeV;
  int   YinfEjeV = 50;
  int   YsupEjeV = 350;
  int  ColorEjes = GREEN;
  int   ColorCal = YELLOW;
  int    ColorFn = RED;
  int       Npts = 500;
  int    Xp,Yp,n;

  double      x,y;
  double    x_ini =  3*Xo;
  double    x_fin =  2*R;
  double   Paso_x = (x_fin-x_ini)/Npts;
  double EscGrafH = (double)(XderEjeH-XizqEjeH)/(x_fin-x_ini);
  double EscGrafV = (double)(YsupEjeV-YinfEjeV)/(double)(R+A);
  double EscGraf;
  double PasoCal;
  double EscUs    = (double)(2*R)/(double)10;

  char   EscUsStr[32];

  ModoGrafico();

  EscGraf=(EscGrafV<EscGrafH?EscGrafV:EscGrafH);
  XEjeV=XizqEjeH+round(fabs(EscGraf*3*Xo));

  YEjeH=YinfEjeV+round(EscGraf*R);
  PasoCal=(double)(XderEjeH-XEjeV)/(double)10;

  // --- TRAZADO DE EJES -----------------------------

  setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);

  // --- TRAZADO DE LA CALIBRACION -----------------

  setcolor(ColorCal);
```

```
  for(n=0;n<=10;n++)
      line(XEjeV+round(n*PasoCal),YEjeH-4,
         XEjeV+round(n*PasoCal),YEjeH+4);
  for(n=0;round(n<(XEjeV-XizqEjeH)/PasoCal);n++)
      line(XEjeV-round(n*PasoCal),YEjeH-4,
         XEjeV-round(n*PasoCal),YEjeH+4);

  // semieje positivo vertical ------------------

  for(n=0;n<=round((YEjeH-YinfEjeV)/PasoCal);n++)
      line(XEjeV-4,round(YEjeH-n*PasoCal),
         XEjeV+4,round(YEjeH-n*PasoCal));

  // semieje negativo vertical ------------------

  for(n=0;n<=round((YsupEjeV-YEjeH)/PasoCal);n++)
      line(XEjeV-4,round(YEjeH+n*PasoCal),
         XEjeV+4,round(YEjeH+n*PasoCal));

  setcolor(LIGHTGRAY);
  sprintf(EscUsStr,"Esc=%2.2lf [1/div]",EscUs);

  outtextxy(XEjeV,30,"FUNCIONES POR TRAMOS");
  outtextxy(XEjeV+20,YEjeH+20,EscUsStr);

  outtextxy(400,400,"TALLER DE LENGUAJES I");
  outtextxy(400,410,"PROGRAMADOR UNIVERSITARIO");
  outtextxy(400,420,"UNIV.NACIONAL DE TUCUMAN");


  // --- TRAZADO DE LA FUNCION POR TRAMOS -------------------------

  for(x=x_ini;x<=x_fin;x+=Paso_x) {

        if(x<=0) y=-A*exp(-x/Xo); else y=sqrt(pow(R,2)-pow(x-R,2));

        Xp=XEjeV+EscGraf*x;
        Yp=YEjeH-EscGraf*y;

        putpixel(Xp,Yp,ColorFn);
        delay(10);
  }

  getch(); ModoTexto();
}

// ------------------------------------------------------------
int round(double x)
{ if((x-(int)x)>=0.5) return((int)(x+1)); return((int)x); }
// ------------------------------------------------------------
```

CAPÍTULO 4 – Graficación de funciones especiales.

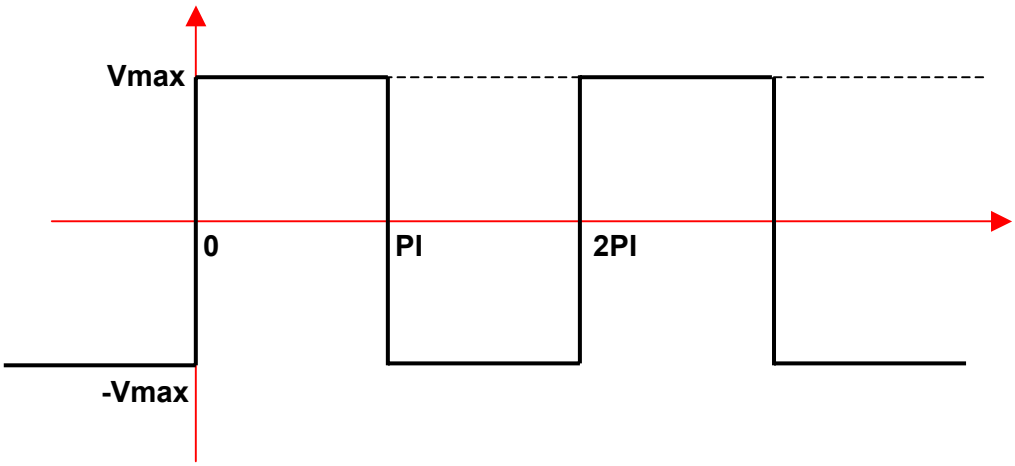
Síntesis de onda cuadrada mediante Serie de Fourier.

Un ingeniero francés llamado Charles Fourier estableció hace muchos años que una función periódica de forma cualquiera puede ser descompuesta en una suma infinita de componentes senoidales de distinta amplitud, frecuencia y fase, y existe toda una matemática para hallar estas componentes basándose en funciones pares, impares, simetría de media onda, etc.

Nuestro interés no es perdernos en laberintos matemáticos, sino partir de una serie ya resuelta y mediante la suma de algunas de sus componentes, restituir la función original. Para nuestro estudio gráfico consideraremos la siguiente expresión:

$$F(x) = \frac{4}{\pi} V \max .(\frac{sen(x)}{1} + \frac{sen(3x)}{3} + \frac{sen(5x)}{5} + ...)$$

que se obtuvo partiendo de la onda cuadrada:



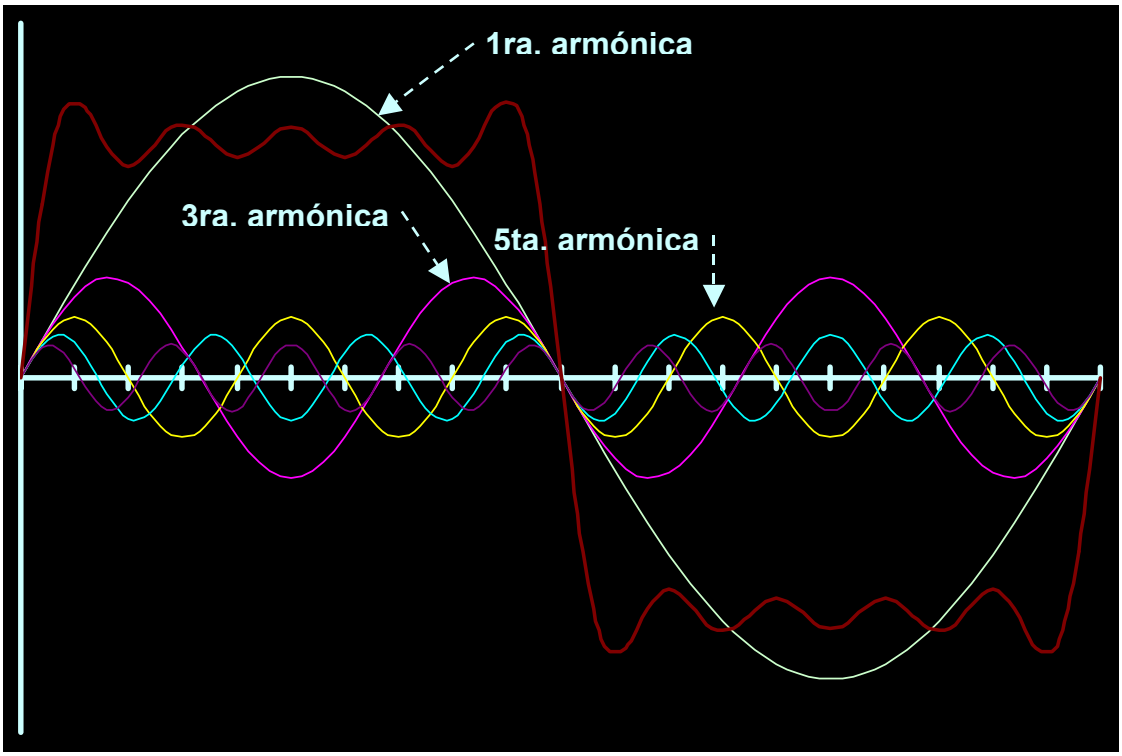
Obviamente para obtener esta onda tan perfecta partiendo de sus componentes, es necesario sumar las infinitas armónicas que la constituyen. No seremos tan exagerados, sino que con unas pocas senoidales visualizaremos la tendencia franca hacia una onda cuadrada. A continuación mostramos los códigos necesarios para implementar el programa:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (  );
void ModoTexto   (  );
```

```
// ------------------------------------------------------------------
void main()
{

  int     Vmax      = 1000;
  int     XizqEjeH  =   50;
  int     XderEjeH  =  550;
  int     YEjeH     =  250;
  int     XEjeV     =   50;
  int     YinfEjeV  =   50;
  int     YsupEjeV  =  400;
  int     Ndiv      =   20;
  int     ColorEjes = GREEN;
  int     ColorCal  = YELLOW;
  int     ColorFn;
  int     ColorFour = LIGHTGREEN;
  int     Npts      =  500;
  int     Narm      =   15;
  int     Xp,Yp;
  int     YpFour;
  int     n;

  double  AmplFour;   // LA SUMA DE TODAS LAS ARMONICAS.
  double  Ampl      = Vmax*4/M_PI;
  double  Xini      = 0;
  double  Xfin      = 2*M_PI;
  double  EscGrafH  = (double)(XderEjeH-XizqEjeH)/(double)(Xfin-Xini);
  double  EscGrafV  = (double)(YsupEjeV-YinfEjeV)/(double)(2*Ampl);
  double  PasoCal   = (double)(XderEjeH-XizqEjeH)/(double)Ndiv;
  double  Paso_x    = (Xfin-Xini)/Npts;
  double  x;

  ModoGrafico();

  // --- TRAZADO DE EJES -----------------------------------

  setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);


  // --- TRAZADO DE CALIBRACION ----------------------------

  setcolor(ColorCal);
  for(n=0;n<=Ndiv;n++)
    line(XizqEjeH+n*PasoCal,YEjeH-3,XizqEjeH+n*PasoCal,YEjeH+3);

  // --- IMPRESION DE MENSAJES -----------------------------

  setcolor(RED);
  outtextxy(60,50,"SERIES DE FOURIER");
  outtextxy(60,60,"SINTESIS DE ONDA CUADRADA");

  setcolor(BROWN);
  outtextxy(400,450,"Programador Universitario");
  outtextxy(400,460,"Taller de Lenguajes I");
  outtextxy(400,470,"Univers. Nacional de Tucuman");
```

```
  // --- GRAFICACION DE LA FUNCION ---------------------------

  for(x=Xini;x<=Xfin;x+=Paso_x) {
       Xp=XizqEjeH+EscGrafH*x; AmplFour=0;

       ColorFn=BLUE;
       for(n=1;n<=Narm;n+=2) {
            Yp=YEjeH-EscGrafV*(Ampl/n)*sin(n*(x));
            putpixel(Xp,Yp,ColorFn++);
            AmplFour+=((Ampl/n)*sin(n*(x)));
       }

       YpFour=YEjeH-EscGrafV*AmplFour;
       putpixel(Xp,YpFour,ColorFour);

  }

  getch(); ModoTexto();
}


// ------------------------------------------------------------------
```

Observe en el bloque de la graficación en sí, que para cada valor del ángulo (en radianes), hemos graficado el píxel correspondiente a cada armónica impar y al mismo tiempo hemos acumulado el valor de la sumatoria en **AmplFour** (amplitud de Fourier), la cual será graficada aparte. Esta nos da la envolvente o resultante de todas las armónicas consideradas:



Figuras de Lissajouss.

Hasta este momento hemos graficado funciones que dependían de una variable independiente, la cual estaba representada en el eje horizontal. Las figuras de Lissajouss en cambio se generan combinando, para un mismo valor angular, magnitudes senoidales en el eje **X** y magnitudes senoidales en el eje **Y**. Ambas funciones senoidales pueden tener la misma amplitud, frecuencia y fase, o no, con lo cual se obtendrán figuras diferentes tales como: una línea recta, una elipse, una circunferencia perfecta, una corona con "n" picos, etc.

x	EjeX	EjeY
x1	**A1.sen(x1)**	**A2.sen(x1)**
x2	**A1.sen(x2)**	**A2.sen(x2)**

Etc.

O sea que en lugar de tener un ángulo en el eje X, tenemos una función senoidal de ese ángulo, y lo mismo ocurre en el eje Y.

Sin embargo estas expresiones matemáticas pueden ser más generales:

A1. sen(n1. x + Fase1)
A2. sen(n2. x + Fase2)

donde "n1" y "n2" son simples multiplicadores de frecuencia angular. Por ejemplo si n1=1 y n2=3, ello implica que la función senoidal correspondiente a las ordenadas tendrá una frecuencia triple a la función correspondiente a las absisas (eje X).

Las fases indican simplemente el ángulo inicial con que arranca la función en el instante t=0.

Supongamos:

A1 = A2 = A
n1 = n2 = 1
Fase1 = Fase 2 = 0

y analicemos la tabla de valores:

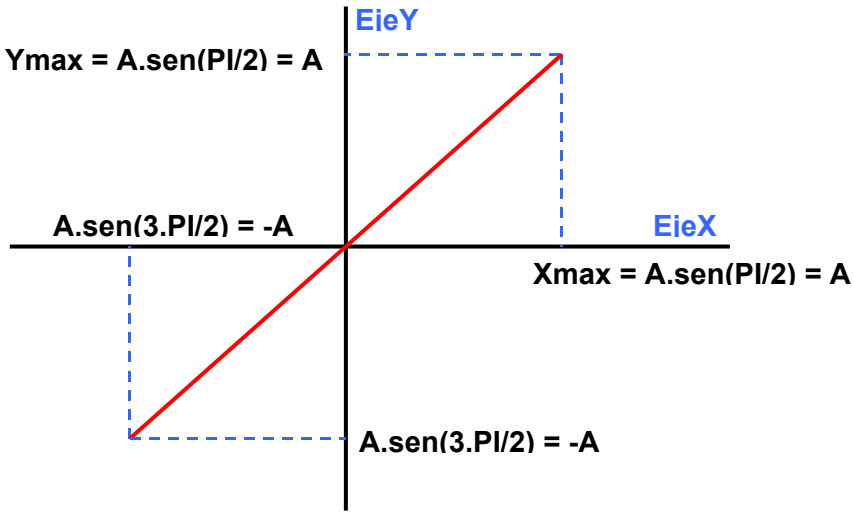
x	F1(x)	F2(x)
x1	**A.sen(x1)**	**A.sen(x)**
x2	**A.sen(x2)**	**A.sen(x)**

etc.

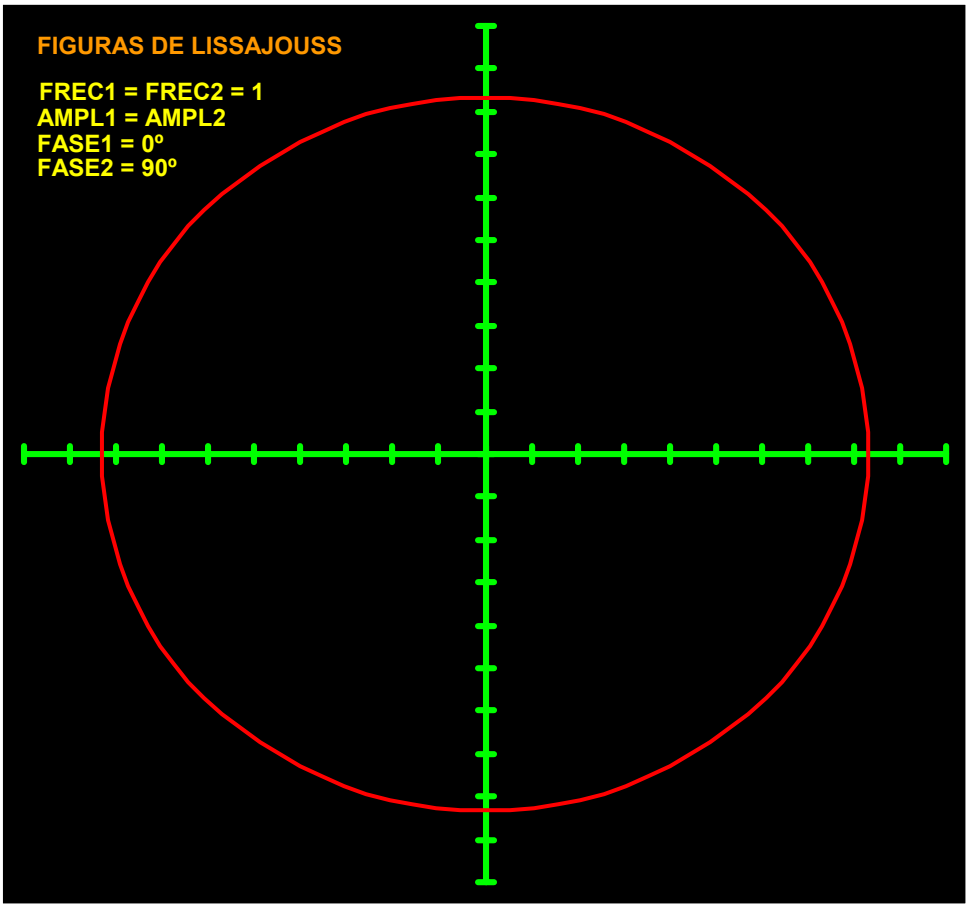
Las ordenadas son exactamente iguales a las absisas ¿y qué se obtiene con ello?:

Una recta a 45 grados.

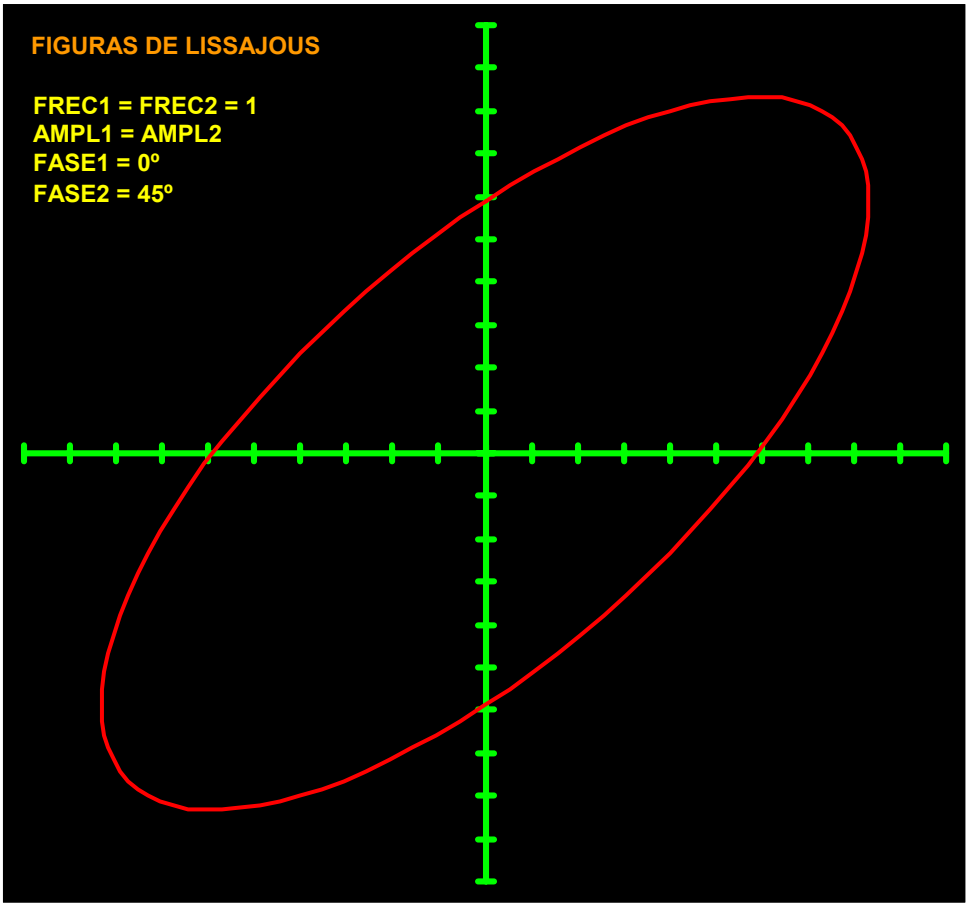
Parece extraño que con dos funciones senoidales se llegue a una figura simple lineal, pero la tabla de valores lo explica muy claramente y el resultado es:

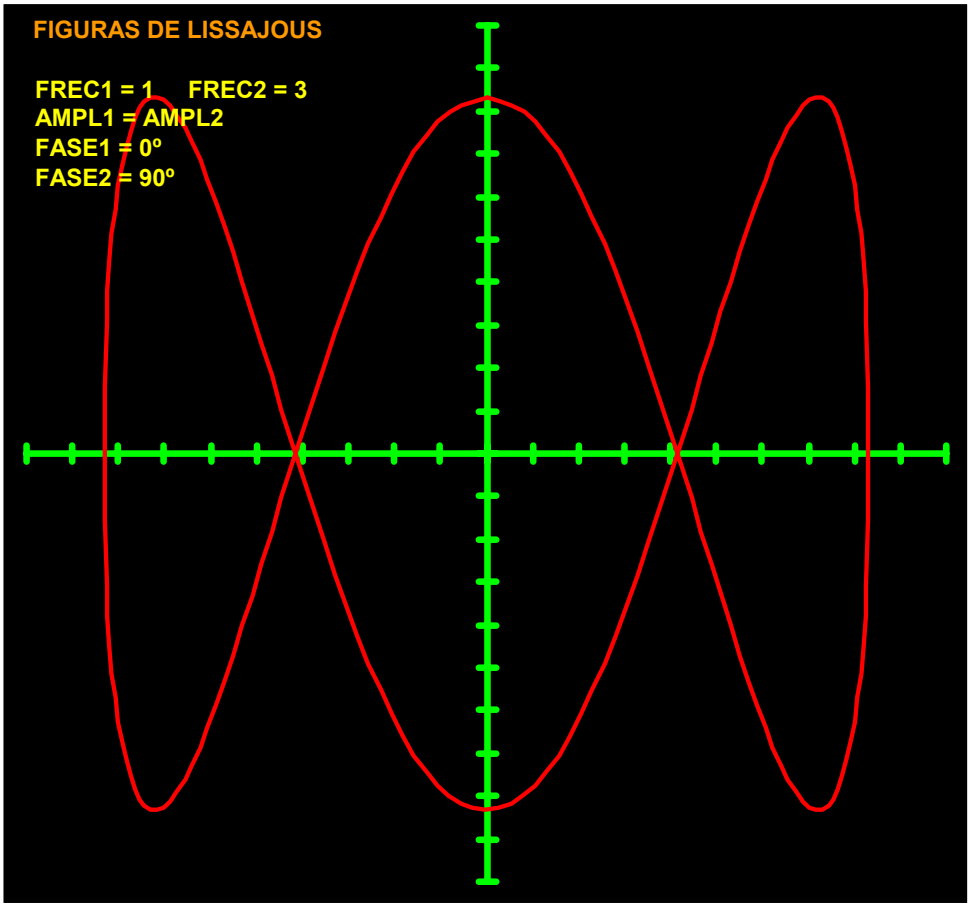


Sin en cambio tomamos un ángulo de fase de 90º obtenemos:



¿Interesante, verdad? Sigamos experimentando: ahora con Fase2 = 45º


FIGURAS DE LISSAJOUS
FREC1 = FREC2 = 1
AMPL1 = AMPL2
FASE1 = 0º
FASE2 = 45º


FIGURAS DE LISSAJOUS
FREC1 = 1 FREC2 = 3
AMPL1 = AMPL2
FASE1 = 0º
FASE2 = 90º

El código de este programa es el siguiente:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (  );
void ModoTexto   (  );

// ---------------------------------------------------------------
void main()
{
  int     Ampl1     = 1000;      // amplitud de la onda 1.
  int     Ampl2     = 1000;      // amplitud de la onda 2.
  int     Frec1     =    1;
  int     Frec2     =    1;
  int     Fase1Grd  =    0;

  int     Fase2Grd  =  -45;
  int     XizqEjeH  =   50;
  int     XderEjeH  =  550;
  int     YEjeH     =  240;
  int     XEjeV     =  300;
  int     YinfEjeV  =   20;
  int     YsupEjeV  =  460;
  int     Ndiv      =   20;
  int     ColorEjes = GREEN;
  int     ColorCal  = YELLOW;
  int     ColorFn   = LIGHTRED;
  int     Npts      =  500;
  int     Xp,Yp;
  int     i;

  double  Xini      =   0*M_PI/180.00;
  double  Xfin      = 360*M_PI/180.00;
  double  Fase1     = Fase1Grd*M_PI/180.00;
  double  Fase2     = Fase2Grd*M_PI/180.00;
  double  EscGrafH  = (double)(XderEjeH-XizqEjeH)/(double)(2*Ampl1);
  double  EscGrafV  = (double)(YsupEjeV-YinfEjeV-
                               60)/(double)(2*Ampl2);
  double  EscGraf;
  double  PasoCal   = (double)(XderEjeH-XizqEjeH)/(double)Ndiv;
  double  Paso_x    = (Xfin-Xini)/Npts;
  double  x;

  char    Ampl1Str[32];
  char    Ampl2Str[32];
  char    Frec1Str[32];
  char    Frec2Str[32];
  char    Fase1Str[32];
  char    Fase2Str[32];

  ModoGrafico();
```

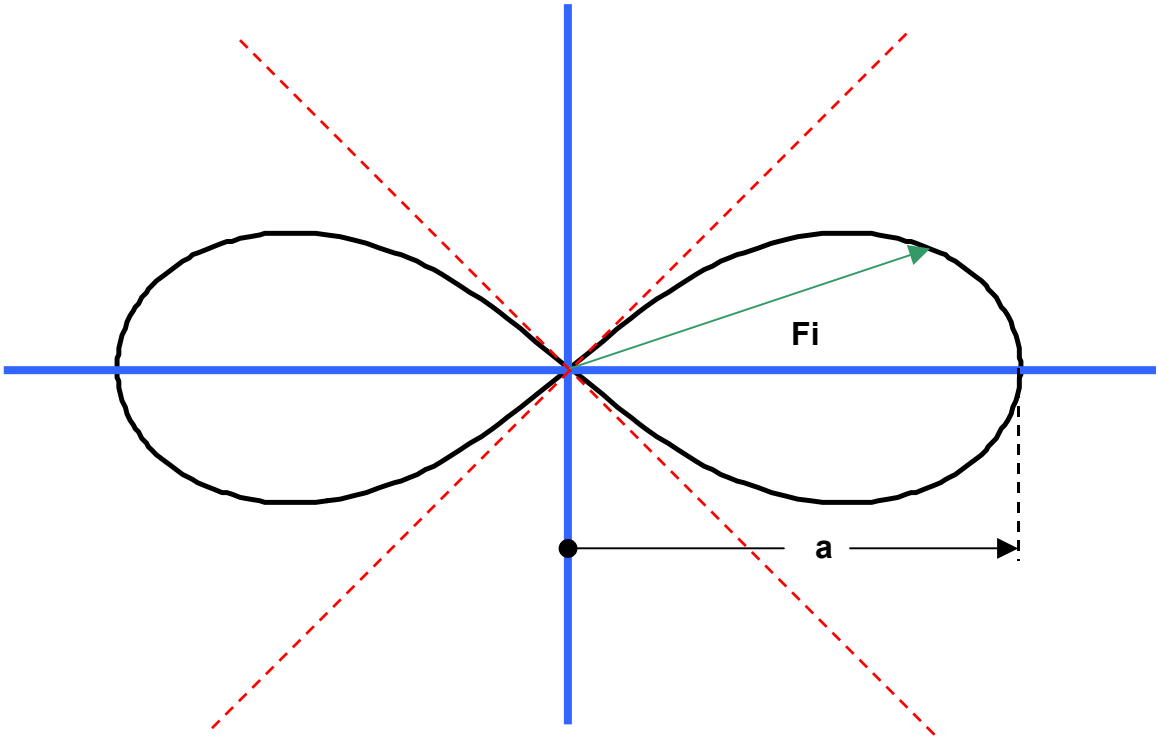
```
// --- TRAZADO DE EJES -------------------------------------

  setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);       // EJE HORIZONTAL.
  line(XEjeV,YinfEjeV-30,XEjeV,YsupEjeV+30); // EJE VERTICAL.
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);

  // --- TRAZADO DE CALIBRACION ------------------------------

  setcolor(ColorCal);
  for(i=0;i<=Ndiv/2;i++) {

    line(XEjeV+i*PasoCal,YEjeH-3,XEjeV+i*PasoCal,YEjeH+3);
    line(XEjeV-i*PasoCal,YEjeH-3,XEjeV-i*PasoCal,YEjeH+3);

    line(XEjeV-3,YEjeH-i*PasoCal,XEjeV+3,YEjeH-i*PasoCal);
    line(XEjeV-3,YEjeH+i*PasoCal,XEjeV+3,YEjeH+i*PasoCal);
  }
  // --- MENSAJES DE PANTALLA --------------------------------

  sprintf(Ampl1Str,"AMPL1 = %d",Ampl1);
  sprintf(Ampl2Str,"AMPL2 = %d",Ampl2);
  sprintf(Fase1Str,"FASE1 = %3d [Grad]",Fase1Grd);
  sprintf(Fase2Str,"FASE2 = %3d [Grad]",Fase2Grd);
  sprintf(Frec1Str,"FREC1 = %d",Frec1);
  sprintf(Frec2Str,"FREC2 = %d",Frec2);

  setcolor(YELLOW);
  outtextxy(50,50,"FIGURAS DE LISSAJOUSS");

  setcolor(RED);
  outtextxy(50, 70, Ampl1Str);
  outtextxy(50, 80, Ampl2Str);
  outtextxy(50, 90, Frec1Str);
  outtextxy(50,100, Frec2Str);
  outtextxy(50,110, Fase1Str);
  outtextxy(50,120, Fase2Str);

  setcolor(BROWN);
  outtextxy(400,450,"Programador Universitario");
  outtextxy(400,460,"Taller de Lenguajes I");
  outtextxy(400,470,"Univers. Nacional de Tucuman");

  // --- TRAZADO DE LA FUNCION DE LISSAJOUSS -----------------

  EscGraf=(EscGrafV<EscGrafH?EscGrafV:EscGrafH);

  for(x=Xini;x<=Xfin;x+=Paso_x) {
      Xp=XEjeV+EscGraf*Ampl1*sin(Frec1*x+Fase1);
      Yp=YEjeH-EscGraf*Ampl2*cos(Frec2*x+Fase2);
      putpixel(Xp,Yp,ColorFn);
  }

  getch(); ModoTexto();
}
// ---------------------------------------------------------------
```

FUNCIONES PLANAS NOTABLES.

LEMNISCATA

Es la curva generada por la ecuación polar: $\mathbf{r^2 = a^2.cos(2.Fi)}$

donde Fi es un ángulo en radianes comprendido entre **–PI/4** a **+PI/4** y "**a**" es una constante.



Para graficar correctamente esta curva debemos considerar las dos raíces posibles de "r", y a continuación convertir en coordenadas cartesianas:

$$r = a.\sqrt{\cos(2.Fi)}$$

$$r = -a.\sqrt{\cos(2.Fi)}$$

x = r.cos(Fi) **y = r.sen(Fi)**

El código para el trazado de esta curva es el siguiente:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (  );
void ModoTexto   (  );

// ---------------------------------------------------------------------
void main()
{
  int           r;
  int       hh,hv;          // para el reticulado de calibracion.
  int           a  = 1500;
  int   AngIniGrd  = 0;
  int   AngFinGrd  = 45;
  int    XizqEjeH  =  50;
  int    XderEjeH  = 550;
  int       YEjeH  = 240;
  int       XEjeV  = 300;
  int    YinfEjeV  =  50;
  int    YsupEjeV  = 430;
  int   ColorEjes  = GREEN;
  int    ColorCal  = DARKGRAY;
  int     ColorFn  = LIGHTRED;
  int        Ndiv  = 20;

  int        Npts  = 1000;
  int     Xp1,Yp1;
  int     Xp2,Yp2;
  int          n;

  double    AngGen;
  double      Krad = M_PI/180.00;
  double    AngIni = AngIniGrd*Krad;
  double    AngFin = AngFinGrd*Krad;
  double   PasoAng = (AngFin-AngIni)/Npts;
  double   PasoCal = (double)(XderEjeH-XizqEjeH)/(double)Ndiv;
  double  EscGrafH = (double)(XderEjeH-XEjeV)/(double)a;
  double     EscUs = (double)(2*a)/(double)Ndiv;

  char    EscUsStr[32];

  ModoGrafico();

  // --- TRAZADO DE EJES COORDENADOS -----------------------

  setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);   // eje horizontal.
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);   // eje vertical.
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);
```

```
  // --- TRAZADO DE CALIBRACION ----------------------------

  setcolor(ColorCal);
  hh=XderEjeH-XEjeV;
  hv=YEjeH-YinfEjeV;

  for(n=0;n<=(Ndiv/2);n++) {

      // --- horizontales ---------------------------------

      line(XEjeV+n*PasoCal,YEjeH-hv,XEjeV+n*PasoCal,YEjeH+hv);
      line(XEjeV-n*PasoCal,YEjeH-hv,XEjeV-n*PasoCal,YEjeH+hv);

      // --- verticales -----------------------------------

      if(n<8) {
        line(XEjeV-hh,YEjeH+n*PasoCal,XEjeV+hh,YEjeH+n*PasoCal);
        line(XEjeV-hh,YEjeH-n*PasoCal,XEjeV+hh,YEjeH-n*PasoCal);

      }
  }

  EscGrafH;
  sprintf(EscUsStr,"EscUs=%2.2lf",EscUs);

  // --- MENSAJES DE PANTALLA ------------------------------

  setcolor(YELLOW);
  outtextxy(50,20,"LEMNISCATA");
  setcolor(GREEN);
  outtextxy(50,30,EscUsStr);

  setcolor(BROWN);
  outtextxy(350,440,"PROGRAMADOR UNIVERSITARIO");
  outtextxy(350,450,"TALLER DE LENGUAJES I");
  outtextxy(350,460,"UNIV.NACIONAL DE TUCUMAN");

  // --- GRAFICACION DE LA FUNCION -------------------------

  for(AngGen=AngIni;AngGen<=AngFin;AngGen+=PasoAng) {

      r  =a*sqrt(cos(2*AngGen));
      Xp1=XEjeV+EscGrafH*(+r*cos(AngGen));
      Xp2=XEjeV+EscGrafH*(-r*cos(AngGen));
      Yp1=YEjeH-EscGrafH*(+r*sin(AngGen));
      Yp2=YEjeH-EscGrafH*(-r*sin(AngGen));

      putpixel(Xp1,Yp1,ColorFn);
      putpixel(Xp2,Yp1,ColorFn);
      putpixel(Xp1,Yp2,ColorFn);
      putpixel(Xp2,Yp2,ColorFn);
  }

  getch(); ModoTexto();

}
// ------------------------------------------------------------------
```

Notará que se ha trabajado con una escala única de graficación para no perder la perspectiva de las asíntotas.

FOLIO DE DESCARTES.

Esta función a diferencia de las anteriores, merece un poco de atención matemática. Podemos expresarla de dos maneras:

$$x^3 + y^3 = 3.a.x.y$$

que resulta muy difícil de despejar, y su equivalente en forma paramétrica:

$$x = \frac{3.a.t}{(1+t^3)} \qquad y = \frac{3.a.t^2}{(1+t^3)}$$

en las cuales aún debemos analizar los posibles valores de "t".
Por ejemplo si t^3 es muy grande comparado con 1, ocurre lo siguiente:

$$x \cong \frac{3.a}{t^2} \qquad y \cong \frac{3.a}{t}$$

o sea que en ambos casos si "t" es grande tanto **x** como **y** tienden a cero (al origen).
Otra zona característica ocurre cuando $\mathbf{t^3}$ es -1 o muy próximo. En este caso las coordenadas del punto tienden a irse hacia el infinito.

Esto plantea un problema práctico en la elección de la escala de graficación, porque a diferencia de una función seno( ), una circunferencia o una exponencial, que poseen una amplitud máxima sencilla, aquí las dispersiones de magnitud en las coordenadas pueden llegar a infinito si nos acercamos demasiado al valor que hace cero el denominador paramétrico.

Por si todo esto fuera poco, existe otro problema adicional: hay zonas donde los puntos tienden a concentrarse y otros donde el crecimiento es mucho más veloz y los puntos comienzan a verse aislados. Entonces pensamos en el acto en bajar el paso de graficación, que siempre nos venía dando resultado, pero en este caso volvemos al problema del acercamiento al punto crítico.

¿Entonces qué podemos hacer?

Una solución que parece funcionar bien es determinar la escala de graficación asumiendo que el mayor acercamiento al denominador cero lo haremos mediante un valor NO TAN PEQUEÑO como el paso de graficación que utilizaremos luego para tener una precisión decente. Esto quiere decir:

```
double  Paso_t  =  0.002;
double  Xmax    =  (double)(a*3*(1+0.5))/(1+pow(-(1+0.5),3));
double  Ymax    =  (double)(a*3*pow(1+0.5,2))/(1+pow(-(1+0.5),3));
```

Lo ideal sería haber puesto:

```
pow(-(1+Paso_t),3)
```

de manera que para cada "Paso" que deseemos adoptar la escala se calcule automáticamente, pero de nuevo ocurre un problema de graficación: el rizo de la función comienza a visualizarse cada vez más pequeño.

Entonces no nos queda otra que adoptar los que se denomina "una solución de compromiso": determinamos una escala razonable y a posteriori reajustamos el paso de graficación y vemos cómo va saliendo la cosa.

```
double  EscGrafH = (double)(XderEjeH-XEjeV)/fabs(Xmax);
double  EscGrafV = (double)(YEjeH-YinfEjeV)/fabs(Ymax);
```

Como la escala permanece fija y los valores paramétricos de las dos variables (x,y) dependen del Paso_t:

```
for(t =T_ini;t<=T_fin;t+=Paso_t) {
    if((D=pow(t,3))!=-1) {
        Xp=XEjeV+EscGraf*((3*a*t)/(1+D));
        Yp=YEjeH-EscGraf*((3*a*pow(t,2))/(1+D));

        if(Yp>=YinfEjeV && Yp<=YsupEjeV)
            putpixel(Xp,Yp,ColorFn);
    }
}
```

es indudable que se perderán puntos, pero lejos del rizo, que es la parte importante de la figura. A continuación viene el código que sería interesante que lo siguiera cuidadosamente:

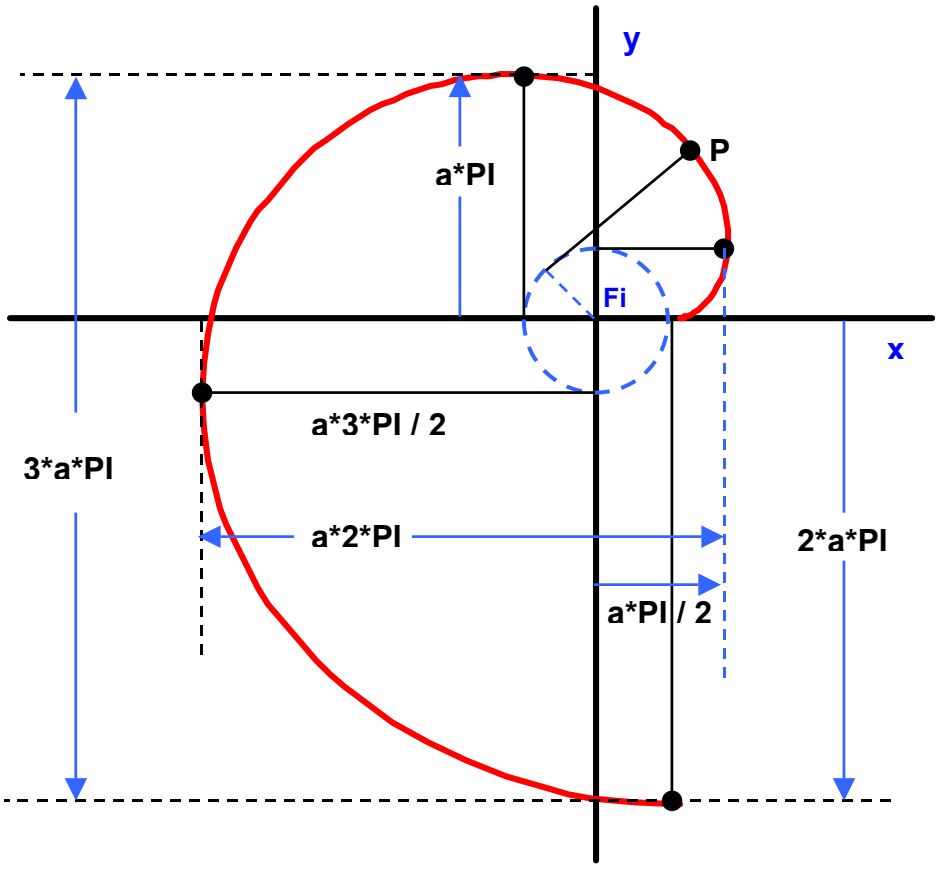
```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>


void ModoGrafico ( );
void ModoTexto   ( );
// ---------------------------------------------------------------------
void main()
{

  int     Xp,Yp;
  int     a          = 1000;
  int     XizqEjeH   =   50;
  int     XderEjeH   =  550;
  int     YEjeH      =  240;
  int     XEjeV      =  300;
  int     YinfEjeV   =  100;
  int     YsupEjeV   =  380;
  int     NdivH      =   10;     // --- en cada semieje.
  int     NdivV      =    5;
  int     ColorEjes  =  GREEN;
```

```
int     ColorCal   =  YELLOW;
int     ColorFn    =  RED;
int     ColorTexto =  RED;
int     n;

double  t;                    // --- parametro de graficacion.
double  D;                    // --- denominador de calculo.
double  T_ini     =  -50;
double  T_fin     =  +50;
double  Paso_t    =   0.002;
double  Xmax      =  (double)(a*3*(1+0.5))/(1+pow(-(1+0.5),3));
double  Ymax      =  (double)(a*3*pow(1+0.5,2))/(1+pow(-(1+0.5),3));

double  PasoCal   =  (double)(XderEjeH-XEjeV)/(double)NdivH;
double  EscGrafH  =  (double)(XderEjeH-XEjeV)/fabs(Xmax);
double  EscGrafV  =  (double)(YEjeH-YinfEjeV)/fabs(Ymax);

double  EscGraf;

ModoGrafico();

// --- TRAZADO DE EJES COORDENADOS ---------------------------

setcolor(ColorEjes);
setlinestyle(SOLID_LINE,1,THICK_WIDTH);
line(XizqEjeH,YEjeH,XderEjeH,YEjeH);   // --- eje horizontal.
line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);   // --- eje vertical.
setlinestyle(SOLID_LINE,1,NORM_WIDTH);


// --- TRAZADO DE CALIBRACION --------------------------------

setcolor(ColorCal);
for(n=0;n<=NdivH;n++) {
    line(XEjeV+n*PasoCal,YEjeH-4,XEjeV+n*PasoCal,YEjeH+4);
    line(XEjeV-n*PasoCal,YEjeH-4,XEjeV-n*PasoCal,YEjeH+4);
    if(n<=NdivV) {

     line(XEjeV-4,YEjeH-n*PasoCal,XEjeV+4,YEjeH-n*PasoCal);
     line(XEjeV-4,YEjeH+n*PasoCal,XEjeV+4,YEjeH+n*PasoCal);
    }

}

// --- MENSAJES DE PANTALLA ----------------------------------

setcolor(ColorTexto);
outtextxy(230,50,"FOLIO DE DESCARTES");

setcolor(BROWN);
outtextxy(350,440,"PROGRAMADOR UNIVERSITARIO");
outtextxy(350,450,"TALLER DE LENGUAJES I");
outtextxy(350,460,"UNIV.NACIONAL DE TUCUMAN");


// --- TRAZADO DE LA FUNCION DE DESCARTES -------------------

EscGraf=(EscGrafV<=EscGrafH?EscGrafV:EscGrafH);
```

```
  for(t=T_ini;t<=T_fin;t+=Paso_t) {
      if((D=pow(t,3))!=-1) {
        Xp=XEjeV+EscGraf*((3*a*t)/(1+D));
        Yp=YEjeH-EscGraf*((3*a*pow(t,2))/(1+D));

        if(Yp>=YinfEjeV && Yp<=YsupEjeV)
             putpixel(Xp,Yp,ColorFn);
      }
  }

  getch(); ModoTexto();
}
// ---------------------------------------------------------------------
```

INVOLUTA DE UNA CIRCUNFERENCIA.

Esta curva queda descripta por el punto extremo **P** de una cuerda enrollada en una circunferencia de radio "**a"** a medida que se desenvuelve y se mantiene tirante y tangencial:

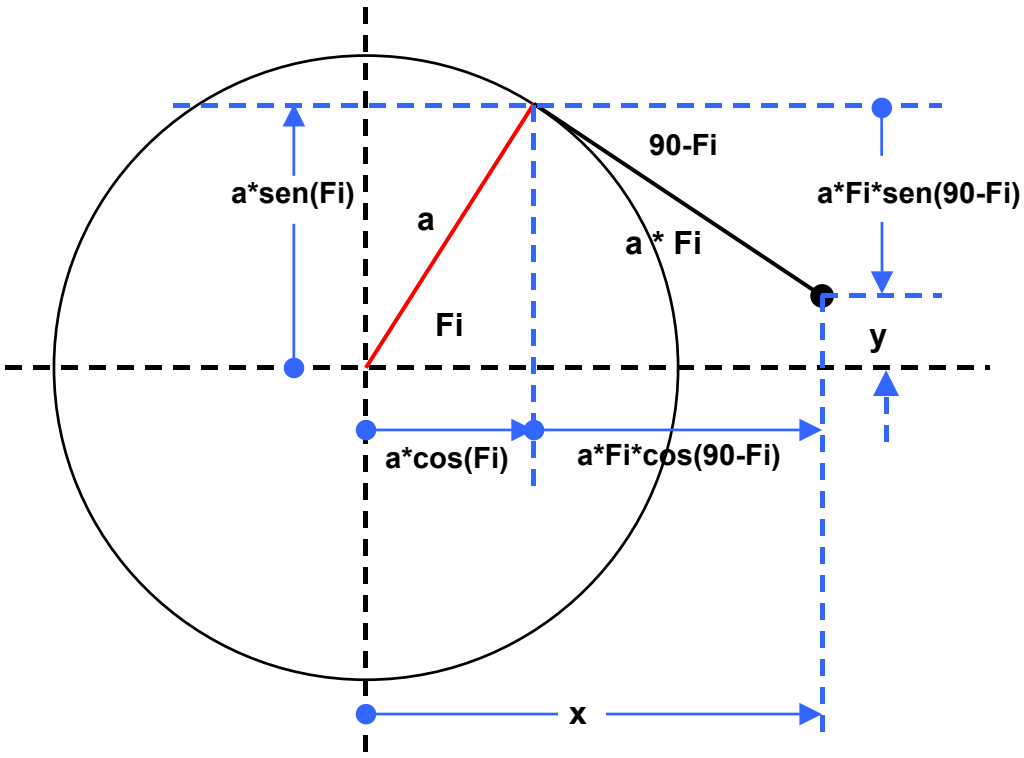


La longitud de la línea generatriz es la misma que la del arco que subtiende el radio vector de longitud "a" al girar, o sea:

a * Fi

con Fi en radianes.

Las ecuaciones paramétricas de esta figura resultan muy sencillas de obtener:



Tomando en cuenta que:

sen(90-Fi) = cos(Fi)
cos(90-Fi) = sen(Fi)

llegamos a:

x = a*(cos(Fi) + Fi*sen(Fi))
y = a*(sen(Fi) – Fi*cos(Fi))

que son las ecuaciones paramétricas buscadas.
Pasemos al código que genere esta interesante curva:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>


void ModoGrafico ( );
void ModoTexto   ( );
```

```
// ------------------------------------------------------------------
void main()
{

  int     Xp,Yp;
  int     a          = 1200;
  int     XizqEjeH   =  150;
  int     XderEjeH   =  450;
  int     YEjeH;
  int     XEjeV;
  int     YinfEjeV   =   50;
  int     YsupEjeV   =  400;
  int     Ndivs      =    3;
  int     Npts       =  500;
  int     ColorEjes  =  GREEN;
  int     ColorCal   =  YELLOW;
  int     ColorFn    =  RED;
  int     ColorTexto =  RED;
  int     n;

  double  PasoCal;
  double  RangoH    =  (double)a*(2*M_PI);
  double  RangoV    =  (double)3*a*(M_PI);
  double  EscGrafH  =  (double)(XderEjeH-XizqEjeH)/RangoH;
  double  EscGrafV  =  (double)(YsupEjeV-YinfEjeV)/RangoV;
  double  EscGraf;
  double  EscUs;
  double  Ang_Ini   =  0;
  double  Ang_Fin   =  2*M_PI;
  double  Paso_Ang  =  (Ang_Fin - Ang_Ini)/Npts;
  double  Ang;

  char    EscUsStr[32];


  ModoGrafico();

  EscGraf=(EscGrafV<=EscGrafH?EscGrafV:EscGrafH);


  // --- TRAZADO DE EJES COORDENADOS ----------------------------

  YEjeH=YinfEjeV+EscGraf*(a*M_PI);
  XEjeV=XizqEjeH+EscGraf*(a*(3*M_PI/2));

  setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);

  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);

  setlinestyle(SOLID_LINE,1,NORM_WIDTH);


  // --- TRAZADO DE CALIBRACION --------------------------------

  setcolor(ColorCal);

  PasoCal=EscGraf*a/(double)Ndivs;
```

```
  EscUs  =a/Ndivs;
  sprintf(EscUsStr,"EscUs=%2.2lf",EscUs);

  for(n=0;(XEjeV+n*PasoCal)<=XderEjeH;n++)
      line(XEjeV+n*PasoCal,YEjeH-4,XEjeV+n*PasoCal,YEjeH+4);
  for(n=0;(XEjeV-n*PasoCal)>=XizqEjeH;n++)
      line(XEjeV-n*PasoCal,YEjeH-4,XEjeV-n*PasoCal,YEjeH+4);

  for(n=0;(YEjeH-n*PasoCal)>=YinfEjeV;n++)
      line(XEjeV-4,YEjeH-n*PasoCal,XEjeV+4,YEjeH-n*PasoCal);
  for(n=0;(YEjeH+n*PasoCal)<=YsupEjeV;n++)
      line(XEjeV-4,YEjeH+n*PasoCal,XEjeV+4,YEjeH+n*PasoCal);


  // --- MENSAJES DE PANTALLA ---------------------------------

  setcolor(ColorTexto);
  outtextxy(XEjeV,20,"INVOLUTA DE UNA CIRCUNFERENCIA");
  outtextxy(XEjeV,30,EscUsStr);

  setcolor(BROWN);
  outtextxy(350,440,"PROGRAMADOR UNIVERSITARIO");
  outtextxy(350,450,"TALLER DE LENGUAJES I");
  outtextxy(350,460,"UNIV.NACIONAL DE TUCUMAN");


  // --- TRAZADO DE LA FUNCION DE DESCARTES -------------------

  for(Ang=Ang_Ini;Ang<=Ang_Fin;Ang+=Paso_Ang) {
      Xp=XEjeV+EscGraf*(a*(cos(Ang)+Ang*sin(Ang)));
      Yp=YEjeH-EscGraf*(a*(sin(Ang)-Ang*cos(Ang)));
      putpixel(Xp,Yp,ColorFn);
  }

  getch(); ModoTexto();

}
// ----------------------------------------------------------------------
```

CAPÍTULO 5 – Animación.

La animación es una técnica que permite ver una imagen en movimiento. Para lograr esto existen dos maneras:

- Tener un conjunto de imágenes del mismo objeto en distintas posiciones y visualizarlas una por una en forma veloz de manera de crear la ilusión de movimiento (en el cine por ejemplo se suelen utilizar 25 cuadros por segundo).

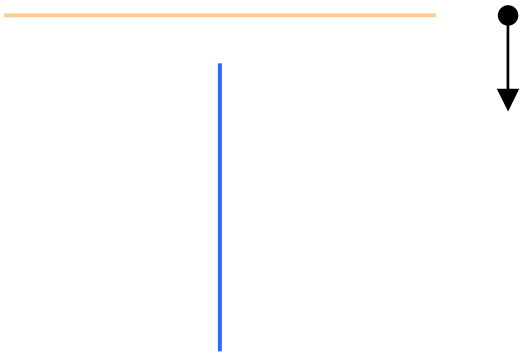
- Dibujar una figura, mostrarla en pantalla un cierto tiempo, borrarla, redibujarla en otra posición, mostrarla nuevamente por pantalla y así sucesivamente. Es muy útil para formas matemáticas y es la que utilizaremos en este curso.

Borrado de una figura.

Si partimos de la premisa que moveremos una figura borrándola y redibujándola, podrían ocurrir dos situaciones:

- Que la figura no pase por encima de ninguna otra en sus desplazamientos.
- Que sí pase por encima de otra imagen.

¿Qué es lo primero que se nos ocurre para borrar solamente una figura ya trazada y limitarnos solamente a ella (no a su entorno)?: volver a dibujar pero esta vez con el color de fondo para de esa manera tornarla invisible.



Pero he aquí un pequeño problema: tipee y haga correr el siguiente programa:

```
/* ----------------- GRAFICACION EN C -----------------------------
                        BARRA01.CPP

Dibujar una línea vertical en color Azul y un poquito por encima de
ella dibujar otra línea pero esta vez horizontal y en color Amarillo.
A continuación, y a medida que se pulse una tecla cualquiera (por eje.
la barra espaciadora), hacer que la línea horizontal vaya bajando y
pase por encima de la otra vertical.

---------------------------------------------------------------- */
```

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (  );
void ModoTexto   (  );

// ---------------------------------------------------------------------
void main()
{

  int   XizqBarraH   = 200;
  int   XderBarraH   = 440;
  int   YBarraH      = 100;
  int   ColorBarraH  = YELLOW;
  int   ColorBarraV  = LIGHTBLUE;
  int   ColorBorr    = BLACK;
  int   XBarraV      = 320;
  int   YinfBarraV   = 120;
  int   YsupBarraV   = 300;
  int   n;

  ModoGrafico();

  setcolor(ColorBarraV);
  line(XBarraV,YinfBarraV,XBarraV,YsupBarraV);

  for(n=0;n<320;n++) {
      setcolor(ColorBarraH);
      line(XizqBarraH,YBarraH+n,XderBarraH,YBarraH+n);
      if(getch()==27) break;
      setcolor(ColorBorr);
      line(XizqBarraH,YBarraH+n,XderBarraH,YBarraH+n);
  }
  ModoTexto();
}
// ---------------------------------------------------------------------
```

¿Notó lo que pasó al desplazar la línea horizontal?:
¡¡ IBA ELIMINANDO LA LINEA VERTICAL !!

Sin embargo esa no era nuestra intención, sino simplemente desplazarla graciosamente por encima de ella. Bueno... afortunadamente nuestra querida librería graphics.h viene en nuestra ayuda a través de una función que soluciona mágicamente este problema: la instrucción **setwritemode( )** que debe ir acompañada de uno de los siguientes tres parámetros:

XOR_PUT
COPY_PUT

El primero de ellos realiza una operación **XOR** entre cada píxel actual y el que está debajo de él (si lo hubiere) en la pantalla.

Por ejemplo consideremos que en el momento de utilizar setwritemode( ) teníamos una línea dibujada en gris claro (LIGHTGRAY) y que encima de ella vamos a trazar otra en color amarillo (YELLOW). Ocurriría lo siguiente:

0 0 0 0 1 0 0 0 LIGHTGRAY (original)
0 0 0 0 1 1 1 0 YELLOW (trazando actualmente).
---
0 0 0 0 0 1 1 0 XOR entre los dos colores (genera otro equivalente)

si ahora volvemos a trazar la línea amarilla exactamente en la misma posición de la anterior, se realizará nuevamente la operación **XOR**:

0 0 0 0 0 1 1 0 Color equivalente.
0 0 0 0 1 1 1 0 YELLOW (trazando actualmente)
---
0 0 0 0 1 0 0 0 LIGHTGRAY original.

desaparece la línea amarilla y **se restituye** lo que había debajo de ella. ¿No es una propiedad excelente? En vista de esto rediseñemos nuestro programa anterior:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (  );
void ModoTexto   (  );

// ---------------------------------------------------------------
void main()
{

  int   XizqBarraH   = 200;
  int   XderBarraH   = 440;
  int   YBarraH      = 100;
  int   ColorBarraH  = YELLOW;
  int   ColorBarraV  = LIGHTBLUE;
  int   ColorBorr    = BLACK;
  int   XBarraV      = 320;
  int   YinfBarraV   = 120;
  int   YsupBarraV   = 300;
  int   n;

  ModoGrafico(); setwritemode(XOR_PUT);

  setcolor(ColorBarraV);
  line(XBarraV,YinfBarraV,XBarraV,YsupBarraV);

  for(n=0;n<320;n++) {
      setcolor(ColorBarraH);
      line(XizqBarraH,YBarraH+n,XderBarraH,YBarraH+n);
      if(getch()==27) break;
      line(XizqBarraH,YBarraH+n,XderBarraH,YBarraH+n);
```

```
  }

  ModoTexto();
}
// ------------------------------------------------------------------
```

IMPORTANTE.

- La función setwritemode( ) SOLO FUNCIONA CON LINEAS o funciones gráficas standard que utilicen líneas en sus trazados. Por ejemplo:

  line( ), rectangle( ), lineto( ), linerel( ) y drawpoly( )

- Para que setwritemode( ) funcione correctamente, es imprescindible que al redibujar la línea en la misma posición, también el color sea el mismo.

Sin embargo no todos nuestros trazados utilizarán líneas, y estamos hablando de las funciones matemáticas. En este caso la función setwritemode( ) no nos sirve de nada y debemos caer en el criticado método de redibujar el píxel en el color de fondo.

A esta altura del partido ya estamos ansiosos por hacer algo más creativo, pero no demasiado complicado aún. ¿Qué tal una especie de radar? O sea una pantallita circular con un radio vector girando dentro de ella. Pasemos al código necesario:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (  );
void ModoTexto   (  );

// ------------------------------------------------------------------
void main()
{
  const int PAUSA  = 40;

  int     Xc1      = 150;
  int     Xc2      = 300;
  int     Xc3      = 450;
  int     Yc       = 240;
  int     R        =  50;
  int     r        =  10;
  int     Npasos   =  90;
  int     ColorLn  = LIGHTRED;

  int     ColorCf1 = LIGHTGREEN;
  int     ColorCf2 = MAGENTA;
  int     Xp1,Yp1;
  int     Xp2,Yp2;
  int     Xp3,Yp3;
  int     n;
  int     Sgn     = -1;
```

```
double  Krad    = M_PI/180.00;
double  Ang;
double  AngIni  = 0*Krad;
double  AngFin  = 360*Krad;
double  PasoAng = (AngFin-AngIni)/Npasos;
double  AngAux;

ModoGrafico();

setcolor(ColorCf1);
circle(Xc1,Yc,R);
circle(Xc2,Yc,R);
circle(Xc3,Yc,R);

setlinestyle(SOLID_LINE,1,THICK_WIDTH);
setcolor(ColorCf2);
for(n=0;n<10;n++) {
    if(n==8) setcolor(LIGHTGRAY);
    circle(Xc1,Yc,r+n);
    circle(Xc2,Yc,r+n);
    circle(Xc3,Yc,r+n);
}
setlinestyle(SOLID_LINE,1,NORM_WIDTH);

setcolor(RED);
outtextxy(250,150,"RADARES ACTIVADOS");

setcolor(BROWN);
outtextxy(350,440,"PROGRAMADOR UNIVERSITARIO");
outtextxy(350,450,"TALLER DE LENGUAJES I");
outtextxy(350,460,"UNIV.NACIONAL DE TUCUMAN");

// --- GRAFICACION DEL RADIO ANIMADO --------------

setwritemode(XOR_PUT);
setcolor(ColorLn);

do {
     for(Ang=AngIni;fabs(Ang)<=AngFin;Ang+=PasoAng) {

             if(kbhit()) break;

             Xp1=Xc1+R*cos(Ang);
             Yp1=Yc-R*sin(Ang);
             Xp2=Xc2+R*cos(Ang);
             Xp3=Xc3+R*cos(Ang);

             // --- CICLO DE TRAZADO DEL RADIO ---

             line(Xc1,Yc,Xp1,Yp1);
             line(Xc2,Yc,Xp2,Yp1);
             line(Xc3,Yc,Xp3,Yp1);

             delay(PAUSA);

             // --- CICLO DE BORRADO -------------

             line(Xc1,Yc,Xp1,Yp1);
             line(Xc2,Yc,Xp2,Yp1);
```

```
                line(Xc3,Yc,Xp3,Yp1);
        }

        PasoAng*=-1;

      } while(!kbhit());
}
// ------------------------------------------------------------------
```

Aquí ha aparecido otra función standard de “C”:

circle(Xc,Yc,Radio)

que permite graficar directamente circunferencias con sólo proporcionarle las coordenadas del centro y el radio (el color debemos indicarlo aparte).
Además hemos agregado un pequeño truquito, el paso angular cambiar de signo en cada giro completo del radio vector lo cual hace que el mismo cambie de sentido periódicamente.

A continuación viene un caso en que no podemos utilizar setwritemode( ). Simularemos un osciloscopio en el cual una señal senoidal se hallará desplazándose continuamente hacia la izquierda.

El primer problema a resolver es puramente matemático: **cómo logramos que la onda se desplace horizontalmente hacia un lado o hacia otro**. Para ello analicemos en primer lugar la ecuación de la senoidal:

F(x) = Ampl*sin(x)

que en forma más general podría ser:

F(x) = Ampl*sin(x+Fase)

y en este momento nos surge la idea: ¿y si dibujamos un ciclo completo, lo borramos, modificamos la fase y volvemos a repetir todo? Estamos en el buen camino:
¡ esa es la solución ¡

Solo que haremos una pequeña modificación: en lugar de modificar la fase en el argumento del seno( ), lo haremos en la variable del lazo que comande la graficación de cada ciclo:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>


void ModoGrafico ( );
void ModoTexto   ( );
// ------------------------------------------------------------------
```

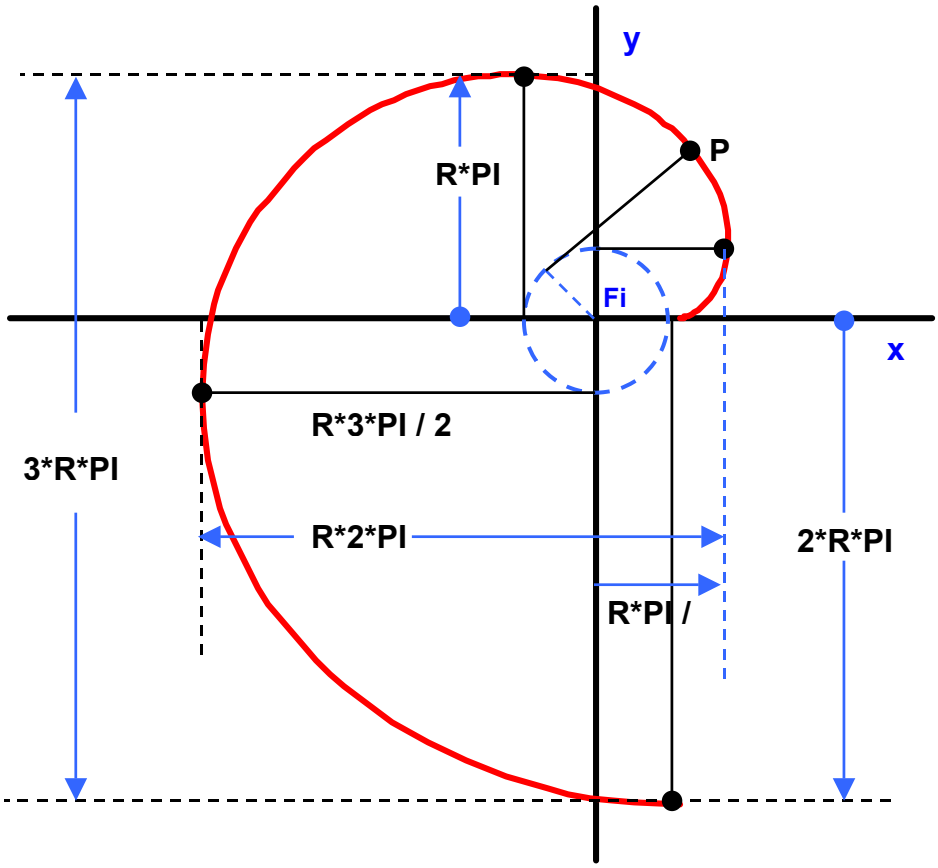
```
void main()
{
  int    Ampl       =  40;
  int    XizqEjeH   = 250;
  int    XderEjeH   = 350;
  int    YEjeH      = 240;
  int    ColorFn    = WHITE;
  int    ColorBk    = BLACK;
  int    Npts       = 150;
  int    Xc         = (XizqEjeH+XderEjeH)/2;
  int    Yc         = YEjeH;
  int    R          = (XderEjeH-XizqEjeH)/2+20;
  int    ColorCf    = LIGHTGREEN;
  int    Xp,Yp,n;

  double Krad     = M_PI/180.00;
  double Fase     = 0*Krad;
  double PasoFase = 5*Krad;
  double x_Ini    = 0*Krad;
  double x_Fin    = 360*Krad;
  double Paso_x   = (x_Fin-x_Ini)/Npts;
  double x;
  double EscGraf  = (double)(XderEjeH-XizqEjeH)/(x_Fin-x_Ini);
  ModoGrafico();

  setcolor(LIGHTRED);
  outtextxy(200,100,"SIMULACION DE OSCILOSCOPIO");
  setcolor(LIGHTGRAY);
  outtextxy(200,110,"(cambio permanente de fase)");

  setcolor(BROWN);
  outtextxy(350,440,"PROGRAMADOR UNIVERSITARIO");
  outtextxy(350,450,"TALLER DE LENGUAJES I");
  outtextxy(350,460,"UNIV.NACIONAL DE TUCUMAN");

  setcolor(ColorCf);
  for(n=0;n<5;n++) circle(Xc,Yc,R+n);

  do {
       if(Fase>x_Fin) Fase=0;

       // --- CICLO DE TRAZADO ----------------------

       for(x=x_Ini+Fase;x<=x_Fin+Fase;x+=Paso_x) {
            Xp=XizqEjeH+EscGraf*(x-Fase);
            Yp=YEjeH-Ampl*sin(x);
            putpixel(Xp,Yp,ColorFn);
       }

       delay(10);

       // --- CICLO DE BORRADO ----------------------

       for(x=x_Ini+Fase;x<=x_Fin+Fase;x+=Paso_x) {
            Xp=XizqEjeH+EscGraf*(x-Fase);
            Yp=YEjeH-Ampl*sin(x);
            putpixel(Xp,Yp,ColorBk);
       }

       Fase+=PasoFase;
```

```
      } while(!kbhit());

   ModoTexto();

}
// ------------------------------------------------------------------
```

Nótese que en el argumento de putpixel( ) los colores cambian entre ColorFn y ColorBk (o color de borrado si lo prefiere). Además debemos chequear si las variaciones de la Fase han completado un giro completo, en ese caso la reinicializamos en cero.

Generación de curvas planas notables.

En un capítulo previo habíamos visto una función matemática especial llamada involuta de la circunferencia cuya forma reproducimos para referencia:

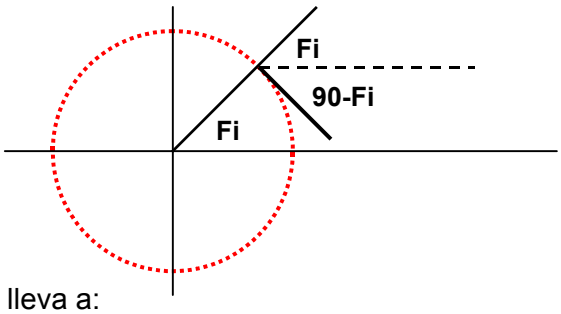


La diferencia está en que esta vez haremos lo necesario para **verla mientras se está generando**. La idea es tener a la vista todo el tiempo el segmento generatriz (la parte desenrollada de la cuerda) e ir moviéndolo y al mismo tiempo imprimir los puntos de su extremo libre que va formando la curva.

Necesitaríamos las coordenadas de los extremos de este segmento para lo cual tendremos que calcular su longitud para en paso de la graficación (recordemos que lo que va variando es el ángulo Fi del radio vector):

d = R*Fi (con Fi en radianes)

y el ángulo que forma con la horizontal es el complemento del ángulo del radio vector:



Esto nos lleva a:

Xp1=Xc+R*cos(Fi); **Xp2=Xc+round(R*(cos(Fi)+Fi*sin(Fi)));**
Yp1=Yc- R*sin(Fi); **Yp2=Yc - round(R*(sin(Fi)-Fi*cos(Fi)));**

done las coordenadas de la izquierda del segmento son las mismas que las del radio vector y las de la derecha son éstas más el agregado de la proyección del segmento generatriz.

El código completo es el siguiente:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (        );
void ModoTexto   (        );
int  round       ( double );
// -------------------------------------------------------
void main()
{
  int     Xc        = 320;
  int     Yc        = 180;
  int     R         =  45;
  int     XizqEjeH  = Xc-2*R;
  int     XderEjeH  = Xc+2*R;
  int     YEjeH     = Yc;
  int     XEjeV     = Xc;
  int     YinfEjeV  = Yc-2*R;
  int     YsupEjeV  = Yc+2*R;
  int     Xp1,Yp1;
  int     Xp2,Yp2;
  int     ColorCf   = LIGHTGRAY;
  int     ColorFn   = RED;
  int     ColorEjes = GREEN;
  int     ColorSegm = LIGHTBLUE;
  int     PAUSA     =  10;
  int     n         =   1;

  double  Krad      = M_PI/180.00;
  double  Fi_ini    =   0*Krad;
```

```
  double  Fi_fin    = 360*Krad;
  double  Paso_Fi   = 0.01;
  double  Fi;

  randomize(); clrscr(); ModoGrafico();

  // --- TRAZADO DE EJES ----------------------

  setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);

  setcolor(GREEN);
  outtextxy(400,50,"INVOLUTA DE LA CIRCUNFERENCIA");
  outtextxy(400,400,"TALLER DE LENGUAJES 1");
  outtextxy(400,410,"PROGRAMADOR UNIVERSITARIO");
  outtextxy(400,420,"UNIV.NACIONAL DE TUCUMAN");

  // --- TRAZADO DE LA INVOLUTA --------------------------

  setcolor(ColorSegm);
  setwritemode(XOR_PUT);

  for(Fi=Fi_ini;Fi<=Fi_fin;Fi+=Paso_Fi) {

      Xp1=Xc+R*cos(Fi);  Xp2=Xc+round(R*(cos(Fi)+Fi*sin(Fi)));
      Yp1=Yc-R*sin(Fi);  Yp2=Yc-round(R*(sin(Fi)-Fi*cos(Fi)));

      putpixel(Xp1,Yp1,ColorCf);   // dibuja la circunferencia base.
      line(Xp1,Yp1,Xp2,Yp2);       // dibuja segmento generatriz.
      line(Xc,Yc,Xp1,Yp1);         // dibuja el radio vector.

      delay(PAUSA);

      line(Xc,Yc,Xp1,Yp1);         // borra el radio vector.
      line(Xp1,Yp1,Xp2,Yp2);       // borra segmento generatriz.
      putpixel(Xp2,Yp2,ColorFn);   // dibuja la envolvente.

      if(Fi>=n*10*Krad) { line(Xp1,Yp1,Xp2,Yp2); n++; }
  }
  line(Xp1,Yp1,Xp2,Yp2);

  getch(); ModoTexto();
}
// ------------------------------------------------------------------
int round(double x)
{ if((x-(int)x)>=0.5) return((int)(x+1)); return((int)x); }
// ------------------------------------------------------------------
```

La instrucción agrega un efecto gráfico muy bonito: líneas salteadas que arrancan tangenciales a la circunferencia y llegan hasta la envolvente de la involuta:

if(Fi>=n*10*Krad) { line(Xp1,Yp1,Xp2,Yp2); n++; }

Figuras de Lissajouss móvil.

Otra de nuestras estrellas de reparto ya conocida, pero esta vez... **¡moviéndose!** Además no solamente girará alrededor de su eje, sino que se irá desplazando horizontalmente por la pantalla en idas y vueltas. Si no recuerda cómo se generan las figuras de Lissajouss, vaya al capítulo correspondiente y repase. Si ya lo hice, analice pues, atentamente el siguiente código:

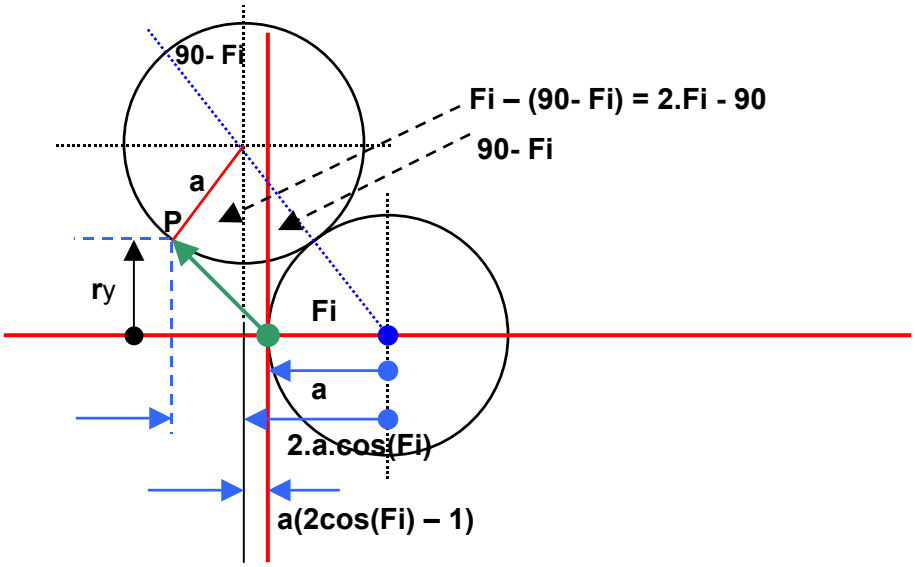
```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (        );
void ModoTexto   (        );
int  round       ( double );
// ------------------------------------------------------
void main()
{
  int     Xc        = 200;
  int     Yc        = 240;
  int     R         =  40;
  int     Frec2     =   1;
  int     ColorEjes = GREEN;
  int     ColorFn   = YELLOW;
  int     ColorBorr = BLACK;
  int     Npts      = 200;
  int     Xp,Yp;
  int     PAUSA     = 6;
  int     PasoXc    = 1;

  double  Fase2;
  double  Fi;
  double  Krad     = M_PI/180.00;
  double  Fi_ini   = 0;
  double  Fi_fin   = 2*M_PI;
  double  Paso_Fi  = (Fi_fin-Fi_ini)/Npts;
  double  PasoFase = 5*Krad;

  ModoGrafico(); setwritemode(XOR_PUT);

  setcolor(LIGHTGRAY);
  outtextxy(400,50,"FIGURAS DE LISSAJOUSS");

  setcolor(GREEN);
  outtextxy(400,400,"TALLER DE LENGUAJES 1");
  outtextxy(400,410,"PROGRAMADOR UNIVERSITARIO");
  outtextxy(400,420,"UNIV.NACIONAL DE TUCUMAN");


  // --- TRAZADO DE LISSAJOUSS --------------------

  Fase2=0; Frec2=3;
```

```
  switch(Frec2) {
       case 1 : PAUSA = 20; break;
       case 2 : PAUSA = 10; break;
       case 3 : PAUSA =  5; break;
  }

  do {
      // --- ciclo de trazado -----------------

      for(Fi=Fi_ini;Fi<=Fi_fin;Fi+=Paso_Fi) {
             Xp=Xc+R*sin(Fi);
             Yp=Yc-R*sin(Frec2*Fi+Fase2);
             putpixel(Xp,Yp,ColorFn);
      }
      delay(PAUSA);

      // --- ciclo de borrado -----------------

      for(Fi=Fi_ini;Fi<=Fi_fin;Fi+=Paso_Fi) {
             Xp=Xc+R*sin(Fi);
             Yp=Yc-R*sin(Frec2*Fi+Fase2);
             putpixel(Xp,Yp,ColorBorr);
      }

      Xc-=PasoXc; if(Xc<R || Xc>(638-R)) PasoXc*=-1;

      Fase2+=PasoFase;
      if(Fase2>2*M_PI) Fase2=0;

     }  while(!kbhit());

  getch(); ModoTexto();
}
// ------------------------------------------------------------------
int round(double x)
{ if((x-(int)x)>=0.5) return((int)(x+1)); return((int)x); }
// ------------------------------------------------------------------
```

Generación de una Cardioide.

Es la curva descripta por un punto P de una circunferencia de radio “a” que rueda por fuera de otra circunferencia de radio también “a”.

Vamos a hallar la ecuación polar de esta curva para lo cual encontraremos un radio en función del ángulo de giro de la circunferencia móvil.



$r_x = a.(2\cos(Fi) – 1) + a.\text{sen}(2Fi – 90)$
$r_x = a.[2\cos(Fi) – 1 + \text{sen}(2Fi – 90)]$
$\text{sen}(2Fi – 90) = \text{sen}(2Fi).\cos(90) – \text{sen}(90).\cos(2.Fi) = -\cos(2Fi)$
$r_x = a.[2\cos(Fi) – 1 - \cos(2Fi)]$

como en realidad este radio es negativo tendríamos:

$r_x = a.[1 + \cos(2Fi) - 2.\cos(Fi)]$

$r_y = 2.a.\text{sen}(Fi) – a.\cos(2Fi - 90)$
$r_y = a.[2.\text{sen}(Fi) – \cos(2Fi - 90)]$

$\cos(2Fi – 90) = \cos(2Fi).\cos(90) + \text{sen}(2Fi).\text{sen}(90) = \text{sen}(2Fi)$
$r_y = a.[2.\text{sen}(Fi) – \text{sen}(2Fi)]$
$\text{sen}(2Fi) = 2.\text{sen}(Fi).\cos(Fi)$

por lo tanto:

$r_y = a.[2.\text{sen}(Fi) – 2.\text{sen}(Fi).\cos(Fi)]$
$r_y = 2.a.\text{sen}(Fi)[1 – \cos(Fi)]$

Hasta este momento tenemos:

$r_x = a.[1 + \cos(2Fi) - 2.\cos(Fi)]$
$r_y = 2.a.\text{sen}(Fi)[1 – \cos(Fi)]$

Pero aún podemos hacer más:

$\cos(2Fi) = 1 – 2.\text{sen}^2(Fi)$

$r_x = a.[1 + 1 – 2.\ \text{sen}^2(Fi) + 2.\cos(Fi)] = 2.a.[1 – \text{sen}^2(Fi) – \cos(Fi)]$

Pero como **$1 - sen^2(Fi) = cos^2(Fi)$**

$r_x = 2.a.[cos^2(Fi) - cos(Fi)]$
$r_x = 2.a.cos(Fi)[cos(Fi) - 1)]$

Finalmente hemos llegado a:

$r_x = 2.a.cos(Fi)[cos(Fi) - 1)]$
$r_y = 2.a.sen(Fi)[1 - cos(Fi)]$

que son las componentes del radio generatriz. Para calcular el radio en sí debemos aplicar Pitágoras:

$r = raíz(4.a^2.cos^2(Fi)[cos(Fi)-1]^2 + 4.a^2.sen^2(Fi).[1-cos(Fi)]^2)$

Operando llegamos finalmente a:

$r = 2.a.[1-cos(Fi)]$

Tomaremos el ángulo Fi como variable independiente y determinaremos:

x = r. cos(Fi)
y = r. sen(Fi)

para realizar nuestra gráfica animada.
El centro de la circunferencia móvil es mucho más sencillo de encontrar:

Xc = 2.a.cos(Fi) – a = a.(2cos(Fi) – 1) pero como en realidad es negativo:

Xc = a.[1 – 2.cos(Fi)]
Yc = 2.a.sen(Fi)

Ya tenemos todo lo necesario para armar nuestro código en “C”:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (  );
void ModoTexto   (  );

// ---------------------------------------------------------------
void main()
{
  int        a     =  50;
  int XizqEjeH     = 100;
  int XderEjeH     = 500;
  int    YEjeH     = 240;
  int    XEjeV     = 250;
  int YinfEjeV     =  50;
  int YsupEjeV     = 410;
```

```
int    Xp,Yp;
int    Xc,Yc;
int Npts         = 300;
int ColorFn      = RED;
int ColorSegm    = LIGHTBLUE;
int ColorEjes    = GREEN;
int ColorCfBase  = BROWN;
int     ColorBk  = BLACK;

double r;
double Krad      = M_PI/180.00;
double Fi_ini    = 0*Krad;
double Fi_fin    = 360*Krad;
double Paso_Fi   = (Fi_fin-Fi_ini)/Npts;
double Fi;

ModoGrafico(); setwritemode(XOR_PUT);

// --- TRAZADO DE EJES -------------------------------

setcolor(ColorEjes);
setlinestyle(SOLID_LINE,1,THICK_WIDTH);
line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
setlinestyle(SOLID_LINE,1,NORM_WIDTH);

// --- RADIO VECTOR Y CIRCUNF. MOVIL DE ARRANQUE ----

setcolor(ColorCfBase);
Xc=XEjeV-a;
Yc=YEjeH;

circle(XEjeV+a,YEjeH,a);  // circunferencia fija.
setcolor(DARKGRAY);
line(XEjeV+a,YEjeH-a-10,XEjeV+a,YEjeH+a+10);

setcolor(ColorSegm);
Xp=XEjeV;
Yp=YEjeH;

line(Xc,Yc,Xp,Yp);
circle(Xc,Yc,a-1);

setcolor(LIGHTGRAY);
circle(XEjeV+a,YEjeH,2*a);

setcolor(YELLOW);
outtextxy(XEjeV+20,50,"GENERACION DE UNA CARDIOIDE");

setcolor(BROWN);
outtextxy(350,440,"PROGRAMADOR UNIVERSITARIO");
outtextxy(350,450,"TALLER DE LENGUAJES I");
outtextxy(350,460,"UNIV.NACIONAL DE TUCUMAN");
```

```
  // --- CICLO DE GENERACION DE LA CARDIOIDE ----------

  for(Fi=Fi_ini;Fi<=Fi_fin;Fi+=Paso_Fi) {

      // --- borra elementos anteriores -------

      setcolor(ColorSegm);
      line(Xc,Yc,Xp,Yp);
      putpixel(Xp,Yp,ColorFn);

      r=2*a*(1-cos(Fi));
      Xp=XEjeV-r*cos(Fi);
      Yp=YEjeH-r*sin(Fi);
      Xc=XEjeV+a*(1-2*cos(Fi));
      Yc=YEjeH-2*a*sin(Fi);

      // --- redibuja en posicion actual ------

      setcolor(ColorSegm);
      line(Xc,Yc,Xp,Yp);

      putpixel(Xp,Yp,ColorFn);

      delay(30);

  }

  getch(); ModoTexto();
}
// ------------------------------------------------------------------
```

En la animación hemos considerado solamente el radio de la circunferencia móvil debido a la complejidad del movimiento (como la circunferencia móvil debe borrarse utilizando el color del fondo resulta inevitable que la misma elimine parte de los puntos de la cardioide).

CAPÍTULO 6 – *Método gráfico en resoluciones matemát.*

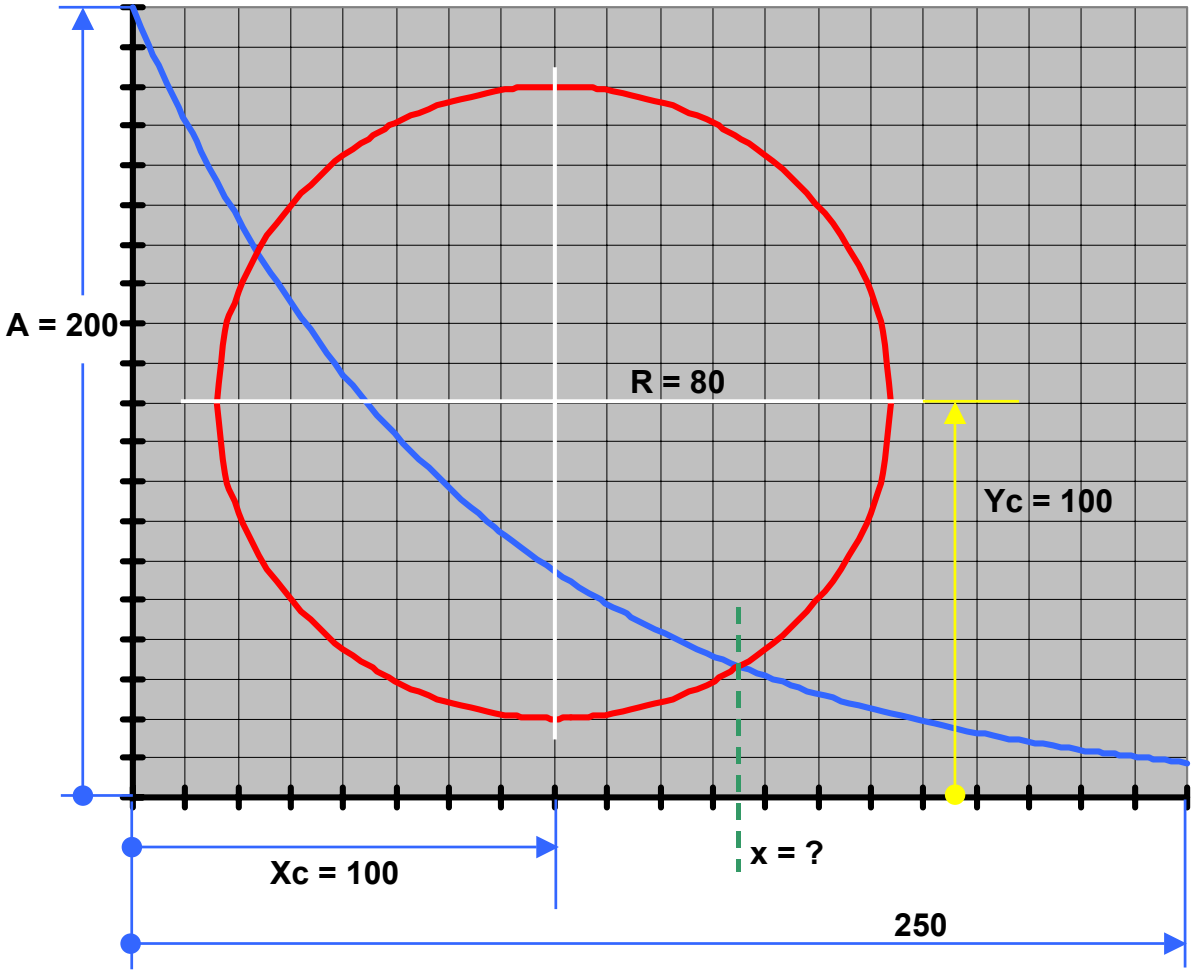
Normalmente los que estamos en Ciencias Exactas tenemos una tendencia hacia lo determinístico y a la aplicación de métodos para calcular "esto" o para calcular "aquello", pero a veces nos topamos con problemas donde lo clásico falla y debemos recurrir a métodos menos ortodoxos. Un caso típico son los sistemas complejos de ecuaciones (en el sentido de la complejidad) donde resulta imposible resolver a lápiz y papel. Por ejemplo:

$$F(x) = Xc \pm \sqrt{R^2 - (x - Xc)^2}$$

$$F(x) = A.e^{-x/Xo}$$

donde la variable independiente aparece en forma cuadrática y en forma exponencial resultando muy complicado, sino imposible, su despeje.

En esta situación la graficación puede venir en nuestra ayuda como un aliado inapreciable:



Y queremos determinar el valor de **x** indicado.

Como el dibujo debe estar hecho perfectamente a escala y los ejes calibrados, resulta sencillo en una primera aproximación calcular gráficamente el valor de **x**:

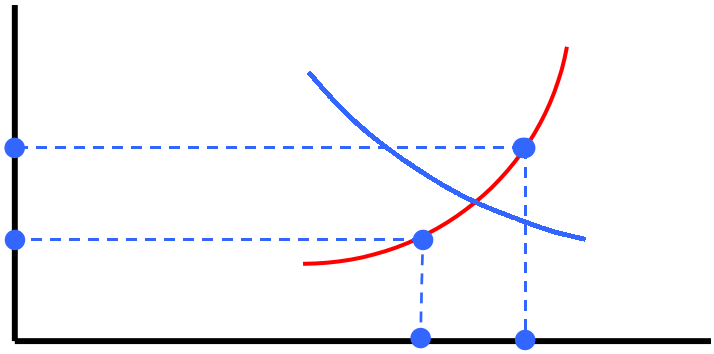
x = EscUsH*número_de_divisones

Sin embargo existe un problema: la apreciación entre divisiones depende exclusivamente del ojo del observador. Por ejemplo en la figura podríamos tener 11,5 divisiones o tal vez 11,6 y porqué no 11,4, no lo sabemos. Sin embargo sabemos algo a ciencia cierta: el resultado buscado se halla entre n=11 y n=12 divisiones. ¿Qué podríamos hacer en este caso? **UNA EXPANSION DE EJES**.

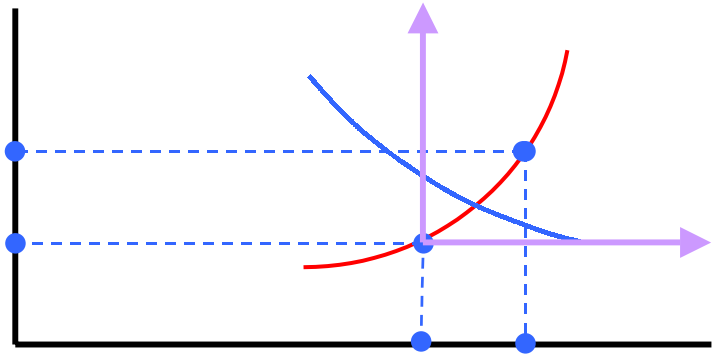
Todos los gráficos realizados hasta ahora se caracterizaban porque el extremo izquierdo del eje horizontal representaba **x_ini** y el extremo final **x_fin**. Seguiremos trabajando de esta manera y ahora haremos:

x_ini = x_ini + n*EscUsH
x_fin = x_ini + (n+1)*EscUsH

resulta como si hubiésemos "estirado" a plena escala el entorno del **x** buscado. Sin embargo al hacer esto no debemos olvidar que también existe un eje vertical que debe ser recalibrado simultáneamente:



También debemos "estirar" a plena escala los valores verticales, y más aún: deben arrancar desde el origen. Es como haber hecho:



pero llevando a **plena escala** el otro punto.
Llegó el momento de analizar los códigos para terminar de comprender:
(hay más explicaciones al final del mismo)

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<math.h>
#include<process.h>
#include<stdio.h>
#include<graphics.h>

void ModoGrafico ( );
void ModoTexto   ( );
// ---------------------------------------------------------------------
void main()
{
  int  XizqEjeH   = 100;
  int  XderEjeH   = 550;
  int  YEjeH      = 350;
  int  XEjeV      = XizqEjeH;
  int  YinfEjeV   = 100;
  int  YsupEjeV   = YEjeH;
  int  ColorEjes  =  GREEN;
  int  ColorFn1   = LIGHTRED;
  int  ColorFn2   = LIGHTCYAN;
  int  ColorCal   = YELLOW;
  int  Npts       = 1000;
  int  NdivH      = 20;
  int  NdivV      = 20;
  int  Xp,Yp1;
  int  Yp2,Yp3;
  int  n;

  double x,y;
  double R        = 80;
  double Xc       = 100;
  double Yc       = 100;
  double A        = 200;
  double Xo       =  80;
  double x_ini    = 0;       //  143.70;  137.5;
  double x_fin    = 250;     //  144.01;  150.0;
  double y_inf    = 0.00;    //   32.92;  30.00;
  double y_sup    = 200;     //   33.40;  38.00;
  double Paso_x   = (x_fin-x_ini)/Npts;
  double EscGrafH = (double)(XderEjeH-XizqEjeH)/(x_fin-x_ini);
  double EscGrafV = (double)(YsupEjeV-YinfEjeV)/(y_sup-y_inf);
  double EscGraf;
  double PasoCalH;
  double PasoCalV;
  double EscUsH   = (x_fin-x_ini)/NdivH;
  double EscUsV   = (y_sup-y_inf)/NdivV;
  double OffsV;

  char   EscUsHStr[32];
  char   EscUsVStr[32];
  char   x_iniStr[32];
  char   y_infStr[32];

  ModoGrafico();
  EscGraf=(EscGrafV<EscGrafH?EscGrafV:EscGrafH);
  PasoCalH=EscGraf*(x_fin-x_ini)/NdivH;
  PasoCalV=EscGraf*(y_sup-y_inf)/NdivV;
```

```
// --- TRAZADO DE EJES ---------------------------

setcolor(ColorEjes);
setlinestyle(SOLID_LINE,1,THICK_WIDTH);
line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
setlinestyle(SOLID_LINE,1,NORM_WIDTH);

// --- TRAZADO DE LA CALIBRACION ------------------------

for(n=0;n<=NdivH;n++) {
    setcolor(DARKGRAY);
    line(XEjeV+n*PasoCalH,YEjeH,XEjeV+n*PasoCalH,YinfEjeV);
    setcolor(ColorCal);
    line(XEjeV+n*PasoCalH,YEjeH-4,XEjeV+n*PasoCalH,YEjeH+4);
}

for(n=0;n<=NdivV;n++) {
    setcolor(DARKGRAY);
    line(XEjeV,YEjeH-n*PasoCalV,
       XEjeV+EscGraf*(x_fin-x_ini),YEjeH-n*PasoCalV);
    setcolor(ColorCal);
    line(XEjeV-4,YEjeH-n*PasoCalV,XEjeV+4,YEjeH-n*PasoCalV);
}

// --- MENSAJES DE PANTALLA ------------------------------

sprintf(EscUsHStr,"EscH=%2.2lf [1/div]",EscUsH);
sprintf(EscUsVStr,"EscV=%2.2lf [1/div]",EscUsV);
sprintf(x_iniStr,"x_ini=%2.2lf",x_ini);
sprintf(y_infStr,"y_inf=%2.2lf",y_inf);

setcolor(BROWN);

outtextxy(XEjeV,YinfEjeV-10,"METODO GRAFICO");

setcolor(LIGHTGRAY);
outtextxy(XEjeV,YEjeH+10,x_iniStr);
outtextxy(XEjeV,YEjeH+20,y_infStr);
setcolor(LIGHTRED);
outtextxy(XEjeV,YEjeH+40,EscUsHStr);
outtextxy(XEjeV,YEjeH+50,EscUsVStr);

// --- TRAZADO DE LAS FUNCIONES -------------------------

// ejes auxiliares de la circunferencia ----------

setcolor(LIGHTBLUE);
line(XEjeV+EscGraf*Xc,YEjeH-EscGraf*(Yc+R+10),
     XEjeV+EscGraf*Xc,YEjeH+EscGraf*(-Yc+R+10));
line(XEjeV+EscGraf*(Xc-R-10),YEjeH-EscGraf*Yc,
     XEjeV+EscGraf*(Xc+R+10),YEjeH-EscGraf*Yc);

// funciones en si ---------------

OffsV=EscGraf*y_inf;

for(x=x_ini;x<=x_fin;x+=Paso_x) {

    Xp =XEjeV+EscGraf*(x-x_ini);
```

```
      Yp3=YEjeH+OffsV-EscGraf*A*exp(-x/Xo);

      if((x>=(Xc-R)) && x<=(Xc+R)) {

        Yp1=YEjeH+OffsV-EscGraf*(Yc+sqrt(pow(R,2)-pow(x-Xc,2)));
        Yp2=YEjeH+OffsV-EscGraf*(Yc-sqrt(pow(R,2)-pow(x-Xc,2)));
        if(Yp1<=YEjeH) putpixel(Xp,Yp1,ColorFn1);
        if(Yp2<=YEjeH) putpixel(Xp,Yp2,ColorFn1);
      }

      if(Yp3<=YEjeH) putpixel(Xp,Yp3,ColorFn2);
      delay(5);
  }

  getch(); ModoTexto();
}
// ------------------------------------------------------------------
```

Las siguientes líneas:

```
double x_ini = 0;          // 143.70;  137.5;
double x_fin = 250;        // 144.01;  150.0;
double y_inf = 0.00;       //  32.92;  30.00;
double y_sup = 200;        //  33.40;  38.00;
```

Representan tres corridas diferentes del programa cambiando los valores extremos de cada eje. Sería una gran cosa que con sólo cambiar los valores de **x** todo el resto se reajustara automáticamente, pero cuando tenemos más de una función en una misma gráfica resulta muy engorroso sino imposible.

Note cómo también las escalas se calculan de manera general:

```
double EscGrafH = (double)(XderEjeH-XizqEjeH)/(x_fin-x_ini);
double EscGrafV = (double)(YsupEjeV-YinfEjeV)/(y_sup-y_inf);
double EscUsH   = (x_fin-x_ini)/NdivH;
double EscUsV   = (y_sup-y_inf)/NdivV;
```

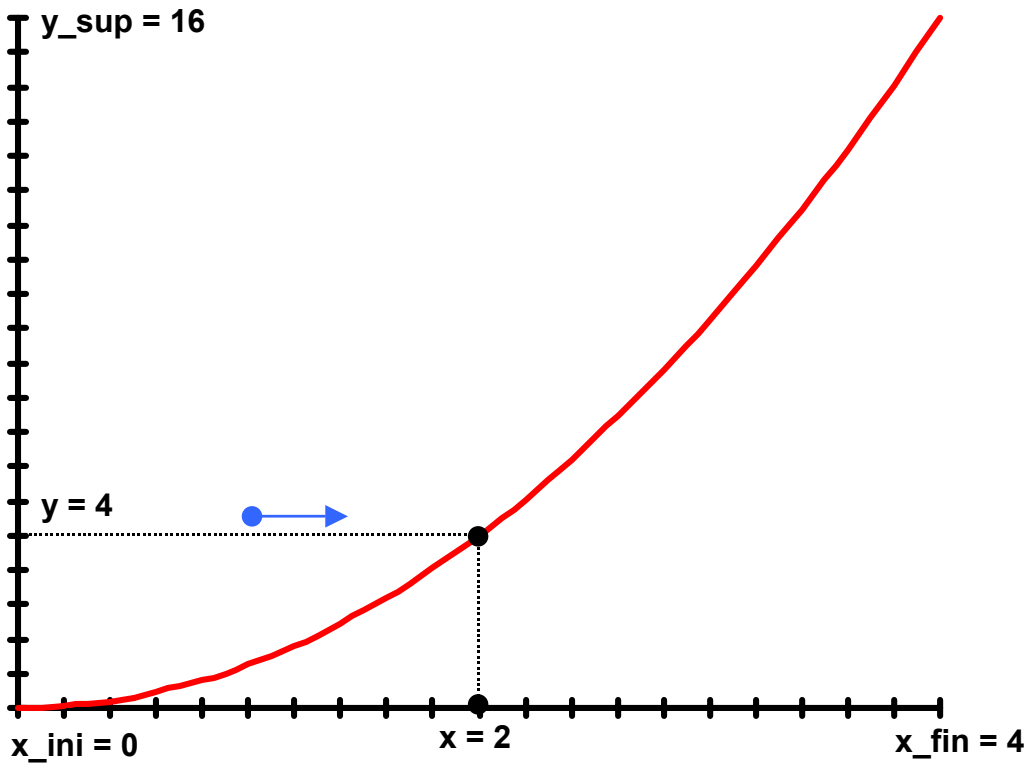
de tal suerte que cada vez que cambiamos los extremos todo se recalcula a los nuevos valores. En este otro tramo del programa:

```
OffsV=EscGraf*y_inf;

for(x=x_ini;x<=x_fin;x+=Paso_x) {
    Xp =XEjeV + EscGraf*(x - x_ini);
    Yp3=YEjeH + OffsV- EscGraf*A*exp(-x/Xo);
```

La magnitud **OffsV** “baja” cada tramo ampliado vertical hasta el origen de ejes, y la operación **(x – x_ini)** hace lo mismo pero en sentido horizontal.

Raíz cuadrada.

La siguiente gráfica representa la **función $y = x^2$**



Sin embargo si partimos del eje vertical y tocamos la curva, obtendremos sobre el eje horizontal el valor de la raíz cuadrada. Este será nuestro objetivo para este problema, pero de una manera muy particular:

Partiremos de un valor dado **N**, calcularemos el valor que le corresponde en pixeles sobre el eje vertical y a partir de allí nos iremos moviendo de píxel en píxel (no en valores de la variable matemática x sino en pixeles) hasta detectar la existencia de la curva. En ese momento tendremos el total de pixeles recorridos y conociendo la escala de graficación horizontal determinaremos el valor de la raíz buscada. Para saber si hemos tocado la curva emplearemos la instrucción:

getpixel(Col,Fila)

que retorna un entero equivalente al color que se halla en la posición (Col,Fila) de la pantalla gráfica.

Como ocurría en el ejemplo anterior, tendremos una apreciación subjetiva cuando la raíz caiga entre divisiones, por ejemplo la raíz de **N= 3** que da un valor fraccionario. Tendremos que valernos del truco de "estirar" los ejes como hicimos antes, pero esta vez con una ventaja: como estamos trabajando con una sola curva basta con reasignar solamente los valores de **x_ini** y **x_fin** y el resto puede recalcularse con toda comodidad en forma automática. Veamos los códigos:

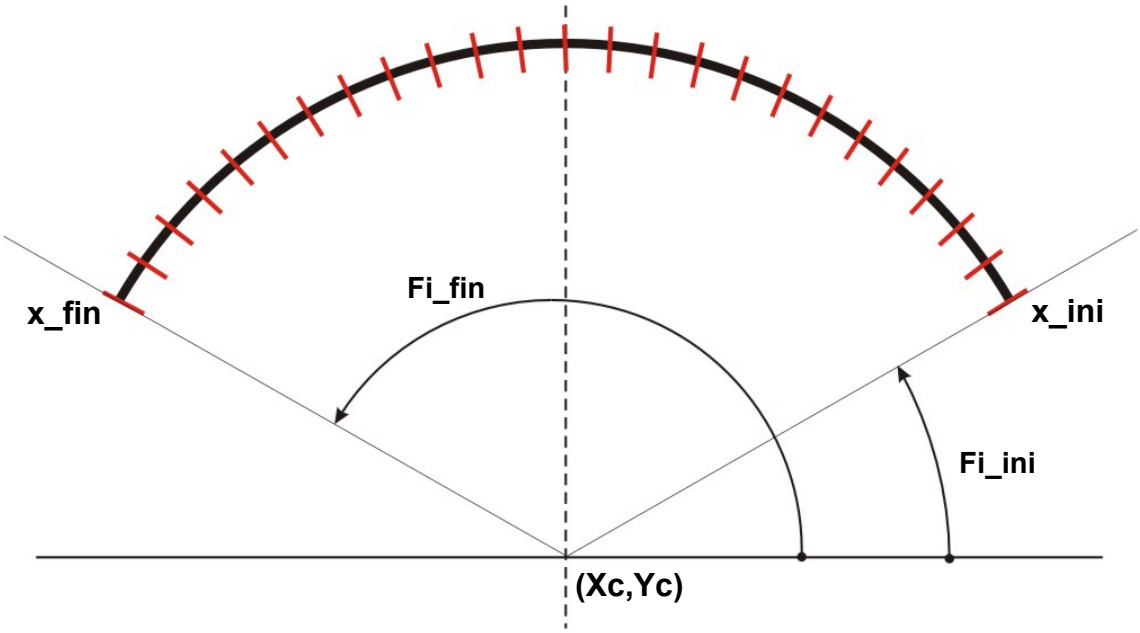
```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (         );
void ModoTexto   (         );
int  round       (double x );
// ------------------------------------------------------------
void main()
{

  int     XizqEjeH    =  50;

  int     XderEjeH    = 600;
  int     YEjeH       = 450;
  int     XEjeV       =  50;
  int     YinfEjeV    =  50;
  int     YsupEjeV    = 450;
  int     NdivH       =  40;
  int     NdivV       =  32;
  int     Npts        = 1000;
  int     ColorEjes   = GREEN;
  int     ColorCal    = YELLOW;
  int     ColorFn     = LIGHTRED;
  int     ColorLinAux = WHITE;
  int     ColorTexto  = RED;
  int     Xp,Yp;
  int     n;

  double  Xini      =  0;
  double  Xfin      =  4;
  double  Paso_x    = (Xfin-Xini)/Npts;
  double  PasoCalH  = (double)(XderEjeH-XizqEjeH)/(double)NdivH;
  double  PasoCalV  = (double)(YEjeH-YinfEjeV)/(double)NdivV;
  double  EscGrafH  = (double)(XderEjeH-XizqEjeH)/(Xfin-Xini);
  double  Rango     = fabs(pow(Xfin,2)-pow(Xini,2));
  double  EscGrafV  = (double)(YsupEjeV-YinfEjeV)/Rango;
  double  x;
  double  EscUsX    = (Xfin-Xini)/NdivH;
  double  EscUsY    = Rango/NdivV;
  double  RaizDe    = 3;
  double  RaizX;

  char    EscUsXStr[64];
  char    EscUsYStr[64];
  char    XiniStr[32];
  char    XfinStr[32];
  char    YiniStr[32];
  char    RaizXStr[32];

  ModoGrafico();

  // --- TRAZADO DE EJES COORDENADOS -------------------------------

  setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
```

```
line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
setlinestyle(SOLID_LINE,1,NORM_WIDTH);


// --- TRAZADO DE CALIBRACION ----------------------------------

setcolor(ColorCal);
for(n=0;n<=NdivH;n++) {
    setcolor(DARKGRAY);
    line(XizqEjeH+n*PasoCalH,YEjeH,XizqEjeH+n*PasoCalH,YinfEjeV);
    setcolor(ColorCal);
    line(XizqEjeH+n*PasoCalH,YEjeH-4,XizqEjeH+n*PasoCalH,YEjeH+4);

    if(n<=NdivV) {
       setcolor(DARKGRAY);
       line(XEjeV,YEjeH-n*PasoCalV,XderEjeH,YEjeH-n*PasoCalV);
       setcolor(ColorCal);
       line(XEjeV-4,YEjeH-n*PasoCalV,XEjeV+4,YEjeH-n*PasoCalV);
    }
}

// --- MENSAJES DE PANTALLA ------------------------------------

sprintf(EscUsXStr,"EscX=%2.5lf [1/div]",EscUsX);
sprintf(EscUsYStr,"EscY=%2.5lf [1/div]",EscUsY);
sprintf(XiniStr,  "Xini=%2.5lf"        ,Xini);
sprintf(XfinStr,  "Xfin=%2.5lf"        ,Xfin);
sprintf(YiniStr,  "Yo  =%2.5lf"        ,pow(Xini,2));

setcolor(ColorTexto);
outtextxy( 50,30,EscUsXStr);
outtextxy( 50,40,EscUsYStr);
setcolor(LIGHTMAGENTA);
outtextxy(240,40,"RAIZ CUADRADA");
setcolor(RED);
outtextxy(XEjeV,YEjeH+10,XiniStr);
outtextxy(XderEjeH-textwidth(XiniStr),YEjeH+10,XfinStr);
outtextxy(XEjeV,YEjeH+20,YiniStr);

setcolor(LIGHTGRAY);
outtextxy(380,40,"Autor:Ing. JUAN MANUEL CONTI");

setcolor(GREEN);
outtextxy(380,10,"PROGRAMADOR UNIVERSITARIO");
outtextxy(380,20,"TALLER DE LENGUAJES I");
outtextxy(380,30,"UNIV.NACIONAL DE TUCUMAN");


// --- TRAZADO DE LA FUNCION EN SI -----------------------------

for(x=Xini;x<=Xfin;x+=Paso_x) {

    Xp=round(XEjeV+EscGrafH*(x-Xini));
    Yp=round(YEjeH-EscGrafV*(pow(x,2)-pow(Xini,2)));
    putpixel(Xp,Yp,ColorFn);
}
```

```
  // --- OBTENCION DE LA RAIZ -----------------------------------

  Yp=YEjeH-round(EscGrafV*(RaizDe-pow(Xini,2)));

  for(Xp=XizqEjeH;Xp<=XderEjeH;Xp++) {

      if(getpixel(Xp,Yp)==ColorFn) {

       RaizX=Xini+(Xp-XizqEjeH)/EscGrafH;
       sprintf(RaizXStr,"Raiz=%2.5lf",RaizX);
       for(;Yp<=YEjeH;Yp++) {
            putpixel(Xp,Yp,ColorLinAux);
            delay(15);
       }
       break;
      }

      putpixel(Xp,Yp,ColorLinAux);
      delay(15);
  }

  setcolor(YELLOW);
  outtextxy(YsupEjeV+10,XizqEjeH+300,RaizXStr);

  getch(); ModoTexto();
}
// ------------------------------------------------------------------
int round(double x)
{ if((x-(int)x)>=0.5) return(x+1); else return((int)x); }
// ------------------------------------------------------------------
```

CAPÍTULO 7 – Escalas angulares.

Son ampliamente utilizadas para los instrumentos de aguja. En las escalas anteriores teníamos [píxel / unidad], puesto que dividíamos un segmento de graficación en una magnitud a representar. Ahora tendremos **[rad / unidad]** o bien **[º / unidad]** ya que dividiremos un espacio angular en una magnitud a representar. Para captar el concepto hagamos un esquema:



Cada extremo de la escala se corresponde con uno de los extremos de la magnitud a medir, por ejemplo Tensión, Corriente, Velocidad, Temperatura, etc. y la diferencia entre **Fi_fin** y **Fi_ini** representa el ángulo disponible para el movimiento de la aguja.

La posición de reposo de la aguja (cuando no está midiendo nada) puede ser:

- **En el extremo izquierdo.**
- **En el extremo derecho.**
- **En el medio.**

Las dos primeras es simplemente una cuestión de elección (normalmente las agujas en un instrumento se mueven siempre de izquierda a derecha). La posición del medio suele utilizarse cuando existen excursiones positivas y negativas en la magnitud física a medir (por ejemplo un termómetro), e incluso las dos mitades no necesariamente deben ser simétricas, pero por simplicidad supongamos que lo sean.

En el ejemplo que viene a continuación simularemos un termómetro que mide temperaturas entre 0 y 100ºC y en el cual la posición de reposo se halla en el extremo izquierdo de la escala. Además la escala se hallará calibrada con 20 divisiones, lo cual implica que cada una de ellas representará 5 ºC de variación.

En el paso final, haremos desplazar la aguja (utilizando lo ya visto en animación) a lo largo de las 20 posiciones de la calibración.

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

const int ESC = 27;

void ModoGrafico (        );
void ModoTexto   (        );
int  round       ( double );

// ---------------------------------------------------------------
void main()
{
  int        Rdial = 200;
  int          Rag = 210;
  int           Xc = 320;
  int           Yc = 300;
  int    Fi_iniGrd =  30;
  int    Fi_finGrd = 150;
  int    ColorDial = GREEN;
  int     ColorCal = WHITE;
  int      ColorAg = YELLOW;
  int      ColorBk = BLACK;
  int        Ndivs = 20;
  int      Xp1,Yp1;
  int      Xp2,Yp2;
  int            n;

  double    x,Offs;
  double    Paso_x = 5;
  double      Krad = M_PI/180;
  double Fi_iniRad = Fi_iniGrd*Krad;
  double Fi_finRad = Fi_finGrd*Krad;
  double   Fi_cero = Fi_finRad;
  double     x_ini = 0;
  double     x_fin = 100;  // grados centrigrados.
  double    EscAng = (Fi_finRad-Fi_iniRad)/(x_fin-x_ini);
  double   PasoCal = (Fi_finRad-Fi_iniRad)/Ndivs;
  double     EscUs = (x_fin-x_ini)/Ndivs;

  char    EscUsStr[32];
  char        xStr[32];

  ModoGrafico();

  // --- CIRCULITO DEL EJE -----------------------

  setcolor(RED);
  circle(Xc,Yc,10);
```

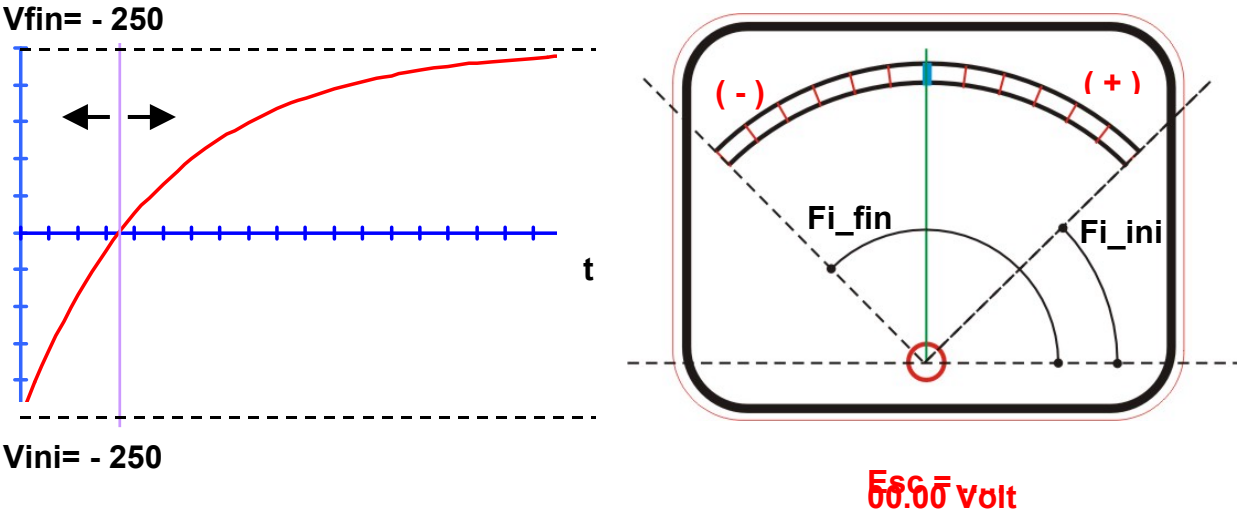
```
  // --- TRAZADO DEL DIAL ------------------------

  setcolor(ColorDial);
  for(n=0;n<5;n++)
      arc(Xc,Yc,Fi_iniGrd,Fi_finGrd,Rdial-2+n);

  // --- TRAZADO DE LA CALIBRACION ---------------

  setcolor(ColorCal);
  for(n=0;n<=Ndivs;n++) {

      Xp1=Xc+(Rdial+4)*cos(Fi_finRad-n*PasoCal);
      Yp1=Yc-(Rdial+4)*sin(Fi_finRad-n*PasoCal);

      Xp2=Xc+(Rdial-4)*cos(Fi_finRad-n*PasoCal);
      Yp2=Yc-(Rdial-4)*sin(Fi_finRad-n*PasoCal);

      line(Xp1,Yp1,Xp2,Yp2);
  }

  // --- MENSAJES DE PANTALLA ------------------------

  setcolor(LIGHTGRAY);

  sprintf(EscUsStr,"Esc=%2.2lf [$/div]",EscUs);

  outtextxy(Xc-50,50,"TERMOMETRO A AGUJA");
  outtextxy(Xc-50,Yc+20,EscUsStr);

  outtextxy(Xc-80,440,"PROGRAMADOR UNIVERSITARIO");
  outtextxy(Xc-80,450,"TALLER DE LENGUAJES I");
  outtextxy(Xc-80,460,"UNIV.NACIONAL DE TUCUMAN");

  // --- TRAZADO DE LA AGUJA -------------------------

  setcolor(ColorAg);
  setwritemode(XOR_PUT);

  for(x=x_ini;x<=x_fin;x+=Paso_x) {

      Offs=EscAng*(x-x_ini);  // es el movimiento angular debido a "x"
      sprintf(xStr,    "Tem=%2.2lf [$]",x);
      setcolor(ColorAg);
      line(Xc,Yc,Xc+Rag*cos(Fi_cero-Offs),Yc-Rag*sin(Fi_cero-Offs));
      setcolor(BROWN);
      outtextxy(Xc-30,Yc+30,xStr);

      if(getch()==ESC) break;

      setcolor(ColorAg);
      line(Xc,Yc,Xc+Rag*cos(Fi_cero-Offs),Yc-Rag*sin(Fi_cero-Offs));
      setcolor(ColorBk);
      outtextxy(Xc-30,Yc+30,xStr);
  }
  ModoTexto();
}


// ------------------------------------------------------------
```

```
int round(double x)
{ if((x-(int)x)>=0.5) return((int)(x+1)); return((int)x); }
// ------------------------------------------------------------
```

Voltímetro de aguja.

Vamos a proponernos realizar el siguiente problema:

Una onda exponencial de tensión que va de **Vini** a **Vfin** debe ser graficada en un par de ejes coordenados. Una línea auxiliar vertical se desplazará en sentido horizontal con las teclas de edición, y un voltímetro de aguja irá visualizando el valor de amplitud de la exponencial. Además debe mostrar en forma digital los valores que va indicando la aguja.




```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

const int Der = 77;
const int Izq = 75;
const int ESC = 27;

// --- DATOS DEL VOLTIMETRO --------------------

int     XcAg        = 470;
int     YcAg        = 260;
int     RintDial    = 145;
int     RextDial    = 155;
int     Rag         = 160;
int     ColorAg     = LIGHTBLUE;
int     ColorDial   = WHITE;
int     AngIniGrd   =  45;
int     AngFinGrd   = 135;
```

```
int     Ndivs      =  18;

double  Krad       = M_PI/180.00;
double  AngReposo  = M_PI/2;
double  AngIniRad  = AngIniGrd*Krad;
double  AngFinRad  = AngFinGrd*Krad;
double  Vini       = -250;
double  Vfin       = +250;
double  PasoAngCal = (AngFinRad-AngIniRad)/Ndivs;
double  EscAngAg   = (AngFinRad-AngIniRad)/(Vfin-Vini);
double  AngAg;

void ModoGrafico (        );
void ModoTexto   (        );
void Voltimetro  (        );
void Aguja       ( double );
int  round       ( double );

// ---------------------------------------------------------------
void main()
{
  int     XizqEjeH     =  50;
  int     XderEjeH     = 300;
  int     YEjeH;
  int     XEjeV        =  50;
  int     YinfEjeV     =  100;
  int     YsupEjeV     = 430;
  int     Npts         = 100;
  int     NdivsH       =  16;
  int     NdivsV       =  20;
  int     ColorEjes    = GREEN;
  int     ColorFn      = RED;
  int     ColorCal     = YELLOW;
  int     ColorLnAux   = DARKGRAY;
  int     Xp,Yp,n;

  char    Tecla;

  double  Vact;
  double  t;
  double  To        =  10;
  double  T_ini     =   0;
  double  T_fin     =   4*To;
  double  Paso_t    = (T_fin-T_ini)/Npts;
  double  PasoAux   = (T_fin-T_ini)/40;
  double  PasoCalH  = (double)(XderEjeH-XizqEjeH)/NdivsH;
  double  PasoCalV  = (double)(YsupEjeV-YinfEjeV)/NdivsV;
  double  EscGrafH  = (double)(XderEjeH-XizqEjeH)/(T_fin-T_ini);
  double  EscGrafV  = (double)(YsupEjeV-YinfEjeV)/(Vfin-Vini);

  char    VactStr[32];

  ModoGrafico();




  // --- TRAZADO DE EJES COORDENADOS -------------------------
```

```
YEjeH=YinfEjeV+round(EscGrafV*Vfin);

setcolor(ColorEjes);
setlinestyle(SOLID_LINE,1,THICK_WIDTH);
line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
setlinestyle(SOLID_LINE,1,NORM_WIDTH);

// --- TRAZADO DE LA CALIBRACION ---------------------------

// --- calibracion del eje horizontal --
setcolor(ColorCal);
for(n=0;n<=NdivsH;n++)
    line(XEjeV+n*PasoCalH,YEjeH-4,XEjeV+
       n*PasoCalH,YEjeH+4);

// --- calibracion del eje vertical ----------
for(n=0;n<=((YEjeH-YinfEjeV)/PasoCalV);n++)
    line(XEjeV-4,YEjeH-round(n*PasoCalV),
       XEjeV+4,YEjeH-round(n*PasoCalV));
for(n=0;n<=((YsupEjeV-YEjeH)/PasoCalV);n++)
    line(XEjeV-4,YEjeH+round(n*PasoCalV),
       XEjeV+4,YEjeH+round(n*PasoCalV));

// --- MENSAJES DE PANTALLA ------------------

setcolor(GREEN);
outtextxy(XizqEjeH,20,"VOLTIMETRO DE AGUJA");
setcolor(LIGHTGRAY);
outtextxy(XizqEjeH,35,"->  avanza barra");
outtextxy(XizqEjeH,45,"<-  retrocede barra");
outtextxy(XizqEjeH,55,"ESC finaliza");

setcolor(GREEN);
outtextxy(350,440,"PROGRAMADOR UNIVERSITARIO");
outtextxy(350,450,"TALLER DE LENGUAJES I");
outtextxy(350,460,"UNIV.NACIONAL DE TUCUMAN");


// --- TRAZADO DE LA FUNCION EXPONENCIAL ------------------

setcolor(ColorFn);
moveto(XEjeV,YEjeH-EscGrafV*(Vini+(Vfin-Vini)*(1-exp(-T_ini/To))));

for(t=T_ini;t<=T_fin;t+=Paso_t) {
    Xp=XEjeV+round(EscGrafH*(t-T_ini));
    Yp=YEjeH-round(EscGrafV*(Vini+(Vfin-Vini)*(1-exp(-t/To))));
    lineto(Xp,Yp);
}

Voltimetro(); setwritemode(XOR_PUT); Aguja(Vini);
```

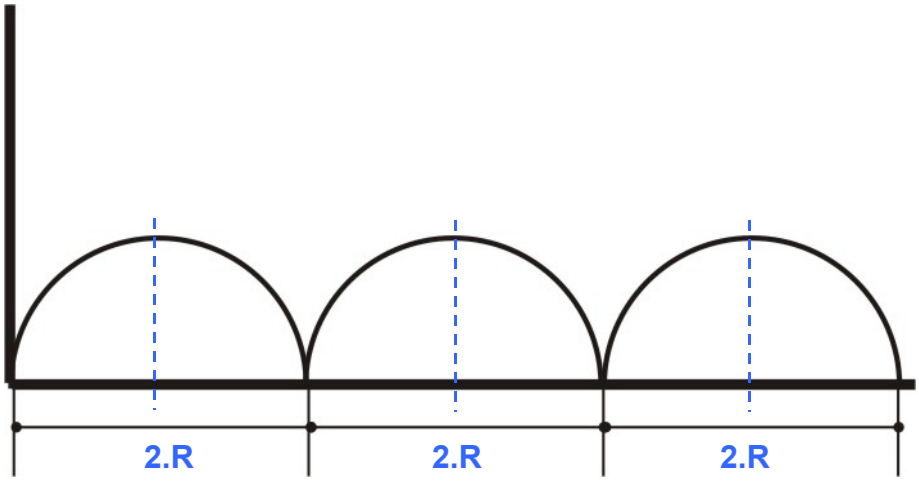
```
// --- MOVIMIENTOS DE LA BARRA AUXILIAR VERTICAL ----------

setcolor(ColorLnAux);
line(XizqEjeH,YsupEjeV,XizqEjeH,YinfEjeV);
t=T_ini; setwritemode(XOR_PUT); setcolor(ColorLnAux);

Xp=XEjeV+round(EscGrafH*(t-T_ini));
line(Xp,YsupEjeV,Xp,YinfEjeV);

Vact=Vini+(Vfin-Vini)*(1-exp(-t/To));
sprintf(VactStr,"%2.2lf",Vact);
setcolor(YELLOW);
outtextxy(XcAg-30,YcAg+40,VactStr);

do {
     if((Tecla=getch())==0)    {
       switch(Tecla=getch()) {

          case Der : if((t+PasoAux)<=T_fin) {
                     setcolor(BLACK);
                     outtextxy(XcAg-30,YcAg+40,VactStr);
                     Vact=Vini+(Vfin-Vini)*(1-exp(-t/To));
                     sprintf(VactStr,"%2.2lf",VactStr);

                     //--- borra linea auxiliar actual ---
                     setcolor(ColorLnAux);
                     line(Xp,YsupEjeV,Xp,YinfEjeV);

                     Aguja(Vact);

                     //--- redibuja linea aux en nueva posic.

                     t+=PasoAux;
                     Xp=XEjeV+round(EscGrafH*(t-T_ini));
                     line(Xp,YsupEjeV,Xp,YinfEjeV);

                     Vact=Vini+(Vfin-Vini)*(1-exp(-t/To));
                     sprintf(VactStr,"%2.2lf",Vact);
                     setcolor(YELLOW);
                     outtextxy(XcAg-30,YcAg+40,VactStr);
                     Aguja(Vact);
                  }

                 break;

          case Izq : if((t-PasoAux)>=T_ini) {
                     setcolor(BLACK);
                     outtextxy(XcAg-30,YcAg+40,VactStr);
                     Vact=Vini+(Vfin-Vini)*(1-exp(-t/To));
                     sprintf(VactStr,"%2.2lf",VactStr);

                     //--- borra linea auxiliar actual ---
                     setcolor(ColorLnAux);
                     line(Xp,YsupEjeV,Xp,YinfEjeV);

                     Aguja(Vact);

                     //--- redibuja lniea auxi en nueva posic.

                     t-=PasoAux;
```

```
                        Xp=XEjeV+round(EscGrafH*(t-T_ini));
                        line(Xp,YinfEjeV,Xp,YsupEjeV);

                        Vact=Vini+(Vfin-Vini)*(1-exp(-t/To));
                        sprintf(VactStr,"%2.2lf",Vact);
                        setcolor(YELLOW);
                        outtextxy(XcAg-30,YcAg+40,VactStr);
                        Aguja(Vact);
                    }

                    break;
          }
        }

     } while(Tecla!=ESC);

  ModoTexto();

}
// ---------------------------------------------------------------------
void Voltimetro(void)
{
  int Resq = 20;   // radio de la esquina.

  // --- TRAZADO DEL DIAL ------------------------

  setcolor(ColorDial);
  arc(XcAg,YcAg,AngIniGrd,AngFinGrd,RintDial);
  arc(XcAg,YcAg,AngIniGrd,AngFinGrd,RextDial);

  // --- TRAZADO DE LA CAJA -----------------------------

  setcolor(BROWN); setlinestyle(SOLID_LINE,1,THICK_WIDTH);

  line(XcAg+Rag*cos(AngFinRad)-10+Resq,YcAg-Rag-10,
       XcAg+Rag*cos(AngIniRad)+10-Resq,YcAg-Rag-10);
  line(XcAg+Rag*cos(AngFinRad)-10+Resq,YcAg+20,
       XcAg+Rag*cos(AngIniRad)+10-Resq,YcAg+20);

  line(XcAg+Rag*cos(AngFinRad)-10,YcAg-Rag-10+Resq,
       XcAg+Rag*cos(AngFinRad)-10,YcAg+20-Resq);
  line(XcAg+Rag*cos(AngIniRad)+10,YcAg-Rag-10+Resq,
       XcAg+Rag*cos(AngIniRad)+10,YcAg+20-Resq);

  arc (XcAg+Rag*cos(AngFinRad)-10+Resq,YcAg-Rag-10+Resq,90,180,Resq);
  arc (XcAg+Rag*cos(AngIniRad)+10-Resq,YcAg-Rag-10+Resq,0,90,Resq);
  arc (XcAg+Rag*cos(AngFinRad)-10+Resq,YcAg+20-Resq,180,270,Resq);
  arc (XcAg+Rag*cos(AngIniRad)+10-Resq,YcAg+20-Resq,270,360,Resq);


  // la linea fina de la carcaza ------------------------

  setlinestyle(SOLID_LINE,1,NORM_WIDTH);
  setcolor(LIGHTRED);

  line(XcAg+Rag*cos(AngFinRad)-10+Resq,YcAg-Rag-15,
       XcAg+Rag*cos(AngIniRad)+10-Resq,YcAg-Rag-15);
  line(XcAg+Rag*cos(AngFinRad)-10+Resq,YcAg+25,
       XcAg+Rag*cos(AngIniRad)+10-Resq,YcAg+25);
```

```
  line(XcAg+Rag*cos(AngFinRad)-15,YcAg-Rag-10+Resq,
       XcAg+Rag*cos(AngFinRad)-15,YcAg+20-Resq);
  line(XcAg+Rag*cos(AngIniRad)+15,YcAg-Rag-10+Resq,
       XcAg+Rag*cos(AngIniRad)+15,YcAg+20-Resq);

  arc (XcAg+Rag*cos(AngFinRad)-10+Resq,YcAg-Rag-
       10+Resq,90,180,Resq+5);
  arc (XcAg+Rag*cos(AngIniRad)+10-Resq,YcAg-Rag-10+Resq,0,90,Resq+5);
  arc (XcAg+Rag*cos(AngFinRad)-10+Resq,YcAg+20-Resq,180,270,Resq+5);
  arc (XcAg+Rag*cos(AngIniRad)+10-Resq,YcAg+20-Resq,270,360,Resq+5);


  // --- TRAZADO DE LA ESCALA DEL DIAL ------------------

  setcolor(ColorDial);
  for(AngAg=AngIniRad;AngAg<=AngFinRad;AngAg+=PasoAngCal)
      line(XcAg+RintDial*cos(AngAg),YcAg-RintDial*sin(AngAg),
         XcAg+RextDial*cos(AngAg),YcAg-RextDial*sin(AngAg));

  setcolor(GREEN);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
  line(XcAg,YcAg-RintDial,XcAg,YcAg-RextDial);
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);

  circle(XcAg,YcAg,5);

  outtextxy(XcAg-3,YcAg-RintDial+5,"0");
  outtextxy(XcAg-95,YcAg-RintDial-10,"(-)");
  outtextxy(XcAg+80,YcAg-RintDial-10,"(+)");

}
// ---------------------------------------------------------------
void Aguja(double Vact)
{

  int    ColorAct = getcolor();
  double AngActAg = AngReposo-Vact*EscAngAg;
  double XExtrAg  = XcAg+Rag*cos(AngActAg);
  double YExtrAg  = YcAg-Rag*sin(AngActAg);

  setcolor(ColorAg);
  line(XcAg,YcAg,XExtrAg,YExtrAg);
  setcolor(ColorAct);
}
// ---------------------------------------------------------------
```

CAPÍTULO 8 – *Funciones cíclicas.*

Asumamos que deseamos obtener la siguiente gráfica:



que se trata de una sucesión de semicircunferencias.
Sin analizamos que la curva se repite siempre a partir de un paso 2R, podríamos establecer que el mismo es un período, y que por lo tanto la función es periódica. Sin embargo la definición de función periódica establece que:

F(x + Xo) = F(x) donde Xo es el período.

Si aplicamos esta definición a la función senoidal **y = 100.sen(30º)** tendríamos:

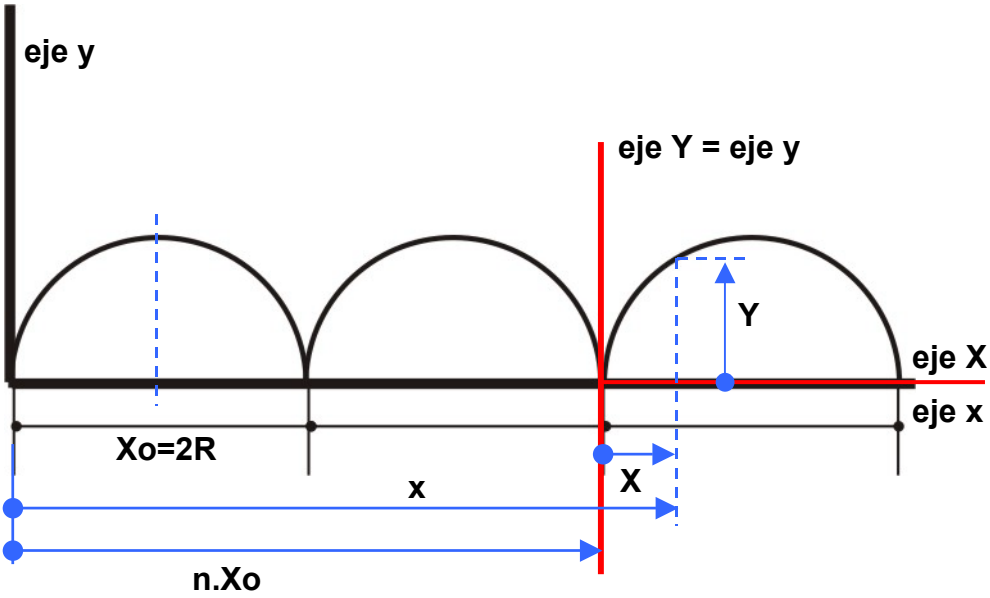
F(30º) = 100 x 0,5 = 50
F(30º + 360º) = F(390º) = 100 x sen(390º) = 50

y si hiciéramos **F(30º + n. 360)** seguiríamos obteniendo el valor 50.

Sin embargo la ecuación de la semicircunferencia de arriba es:

$$F(x) = \sqrt{R^2 - (x - R)^2}$$

siendo sólo válida para x's comprendidos entre **0 y 2.R**, más allá no existe y se obtienen raíces negativas. Evidentemente ésta no se trata de una función periódica desde el punto de vista estrictamente matemático. Sin embargo trabajemos un poco y veamos qué podemos hacer para entrar en el formalismo requerido.



La ecuación de la semircunferencia en su propio sistema de ejes (Rojos) sería:

$$Y = y = \sqrt{R^2 - (X - R)^2}$$

Pero:

$$X = x - n.Xo$$

Donde "n" es el número de períodos que **preceden** a la semicircunferencia actual:

$$n = (\text{int})\frac{x}{Xo}$$

Juntando todo esto obtendríamos:

$$y = \sqrt{R^2 - (x - \left[X_0.(\text{int})(\frac{x}{X_0}) \right] - R)^2}$$

que **sí** es una función periódica, puesto que "ajusta" automáticamente los valores continuos de **x** reduciendo siempre a "primer cuadrante" donde en realidad está definida la semicircunferencia.

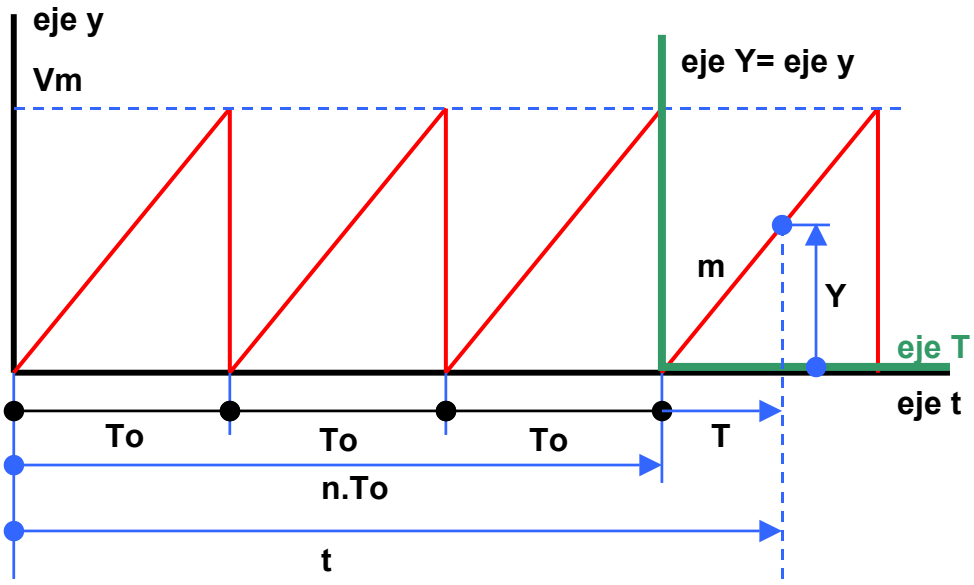
En el código que viene a continuación, en lugar de tener una única expresión para **y**, hemos preferido colocar los desplazamientos aparte, para darle mayor claridad:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>
```

```
void ModoGrafico (  );
void ModoTexto   (  );

// ------------------------------------------------------
void main()
{
  int         R  = 2500;
  int  XizqEjeH  =  100;
  int  XderEjeH  =  600;
  int  YEjeH     =  350;
  int  XEjeV     = XizqEjeH;
  int  YinfEjeV  = 100;
  int  YsupEjeV  = YEjeH;
  int  ColorFn   = RED;
  int  ColorEjes = GREEN;
  int  ColorCal  = YELLOW;
  int  Npts      = 500;
  int  Ndivs     =  15;
  int  Xp,Yp,n;

  double  X,x;
  double  x_ini    = 0;
  double  x_fin    = 3*2*R;
  double  Paso_x   = (x_fin-x_ini)/Npts;
  double  EscGrafH = (double)(XderEjeH-XEjeV)/(x_fin-x_ini);
  double  EscGrafV = (double)(YEjeH-YinfEjeV)/(double)R;
  double  PasoCal  = (double)(XderEjeH-XEjeV)/(double)Ndivs;
  double  EscGraf;
  double  EscUs    = (x_fin-x_ini)/Ndivs;

  char    EscUsStr[32];

  ModoGrafico();
  EscGraf=(EscGrafV<EscGrafH?EscGrafV:EscGrafH);

  // --- TRAZADO DE EJES -----------------------------------

  setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);

  // --- TRAZADO DE LA CALIBRACION -------------------------
  // calibracion y reticulado vertical ----------------------

  for(n=0;n<=Ndivs;n++) {
      setcolor(DARKGRAY);
      line(XEjeV+n*PasoCal,YEjeH,XEjeV+n*PasoCal,YinfEjeV);
      setcolor(ColorCal);
      line(XEjeV+n*PasoCal,YEjeH-4,XEjeV+n*PasoCal,YEjeH+4);
  }
  // calibracion y reticulado horizontal --------------------

  for(n=0;n<=(YEjeH-YinfEjeV)/PasoCal;n++) {
      setcolor(DARKGRAY);
      line(XEjeV,YEjeH-n*PasoCal,XderEjeH,YEjeH-n*PasoCal);
      setcolor(ColorCal);
      line(XEjeV-4,YEjeH-n*PasoCal,XEjeV+4,YEjeH-n*PasoCal);
  }
```

```
  // --- MENSAJES DE PANTALLA -------------------------------

  setcolor(LIGHTGRAY);
  sprintf(EscUsStr,"Esc=%2.2lf [1/div]",EscUs);
  outtextxy(100,80,"FUNCIONES CICLICAS");
  outtextxy(100,YEjeH+10,EscUsStr);

  outtextxy(400,YEjeH+40,"TALLER DE LENGUAJES I");
  outtextxy(400,YEjeH+50,"PROGRAMADOR UNIVERSITARIO");
  outtextxy(400,YEjeH+60,"UNIV.NACIONAL DE TUCUMAN");

  // ---- TRAZADO DE LA FUNCION CICLICA -------------

  for(x=x_ini;x<=x_fin;x+=Paso_x) {
        Xp=XEjeV+EscGraf*(x-x_ini);
        X=x-2*R*(int)(x/(2*R));
        Yp=YEjeH-EscGraf*sqrt(pow(R,2)-pow(X-R,2));
        putpixel(Xp,Yp,ColorFn);
        delay(10);
  }
  getch(); ModoTexto();
}
// ------------------------------------------------------------
```

Onda diente de sierra.



El planteo es exactamente el mismo que las semicircunferencias del problema anterior. Si “t” es una variable continua que arranca en el origen y toma valores en forma continua, deberemos hacer siempre esa especie de reducción a primer cuadrante.

La ecuación de la recta en su sistema de referencia es:

y = m.T
T = t – n.To
n = (int)(t/To)

con lo cual llegamos a:

y = m.[t – To.(int)(t/To)]

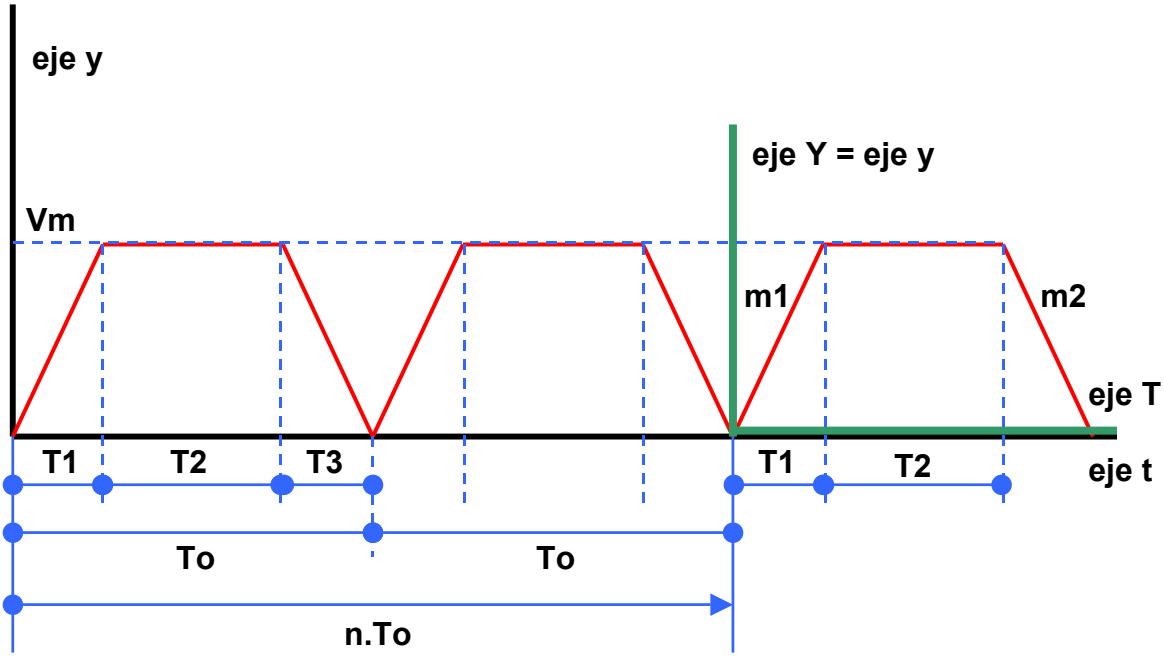
siendo **m = Vm / To**

Sólo nos resta generar el código que realice la gráfica:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (  );
void ModoTexto   (  );

// ------------------------------------------------------
void main()
{
  int  XizqEjeH  =  100;
  int  XderEjeH  =  600;
  int  YEjeH     =  350;
  int  XEjeV     = XizqEjeH;
  int  YinfEjeV  = 100;
  int  YsupEjeV  = YEjeH;
  int  ColorFn   = RED;
  int  ColorEjes = GREEN;
  int  ColorCal  = YELLOW;
  int  Npts      = 1000;
  int  NdivsH    =  15;
  int  NdivsV    =  20;
  int  Xp,Yp,n;

  double  T,t;
  double  Vm       = 2500;
  double  To       = 20;
  double  t_ini    = 0;
  double  t_fin    = 5*To;
  double  Paso_t   = (t_fin-t_ini)/Npts;
  double  EscGrafH = (double)(XderEjeH-XEjeV)/(t_fin-t_ini);
  double  EscGrafV = (double)(YEjeH-YinfEjeV)/(double)Vm;
  double  PasoCalH = (double)(XderEjeH-XEjeV)/(double)NdivsH;
  double  PasoCalV = (double)(YEjeH-YinfEjeV)/(double)NdivsV;
  double  EscGraf;
  double  EscUsH   = (t_fin-t_ini)/NdivsH;
  double  EscUsV   = Vm/NdivsV;

  char    EscUsHStr[32];
  char    EscUsVStr[32];

  ModoGrafico();

  // --- TRAZADO DE EJES ---------------------------------

  setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
```

```
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);

  // --- TRAZADO DE LA CALIBRACION -------------------------
  // calibracion y reticulado vertical ----------------------

  for(n=0;n<=NdivsH;n++) {
      setcolor(DARKGRAY);
      line(XEjeV+n*PasoCalH,YEjeH,XEjeV+n*PasoCalH,YinfEjeV);
      setcolor(ColorCal);
      line(XEjeV+n*PasoCalH,YEjeH-4,XEjeV+n*PasoCalH,YEjeH+4);
  }
  // calibracion y reticulado horizontal --------------------

  for(n=0;n<=NdivsV;n++) {
      setcolor(DARKGRAY);
      line(XEjeV,YEjeH-n*PasoCalV,XderEjeH,YEjeH-n*PasoCalV);
      setcolor(ColorCal);
      line(XEjeV-4,YEjeH-n*PasoCalV,XEjeV+4,YEjeH-n*PasoCalV);
  }

  // --- MENSAJES DE PANTALLA -------------------------------

  setcolor(LIGHTGRAY);
  sprintf(EscUsHStr,"EscH=%6.2lf [1/div]",EscUsH);
  sprintf(EscUsVStr,"EscV=%6.2lf [V/div]",EscUsV);

  outtextxy(100,80,"FUNCIONES CICLICAS");
  outtextxy(100,YEjeH+10,EscUsHStr);
  outtextxy(100,YEjeH+20,EscUsVStr);

  setcolor(RED);
  outtextxy(400,YEjeH+40,"TALLER DE LENGUAJES I");
  outtextxy(400,YEjeH+50,"PROGRAMADOR UNIVERSITARIO");
  outtextxy(400,YEjeH+60,"UNIV.NACIONAL DE TUCUMAN");

  // ---- TRAZADO DE LA FUNCION CICLICA -------------

  setcolor(ColorFn);
  for(t=t_ini;t<=t_fin;t+=Paso_t) {

        Xp=XEjeV+EscGrafH*(t-t_ini);
        T=t-To*(int)(t/To);

        if(T<=Paso_t)
           if(t>Paso_t)
             line(Xp,YEjeH,Xp,YEjeH-EscGrafV*Vm);

        Yp=YEjeH-EscGrafV*(Vm/To)*T;
        putpixel(Xp,Yp,ColorFn);
        delay(5);
  }
  getch(); ModoTexto();
}
// ------------------------------------------------------
```

Onda trapecial.



Este es un caso un poco más complicado debido a que existen tres funciones diferentes dentro de cada período, pero eso no nos asusta porque ya manejamos muy bien nuestro método sistemático de traslación de ejes:

T1= 5
T2= 10
T3= 5
To= T1 + T2 + T3

m1= Vm / T1
m2= -Vm / T3

La recta con pendiente **m2** al estar referida a los ejes T-Y, tiene la siguiente ecuación:

y2= m2.T + b

pero como conocemos un punto de paso de la misma podemos escribir:

0= m2*(To) + b → b= -m2.To
b= (Vm / T3)*To

Ya estamos en condiciones de escribir las ecuaciones de cada tramo de las funciones que forman cada período:

y1= m1*T para todo t en el intervalo [0 .. T1]
y2 = Vm para todo t en el intervalo [T1 .. (T1+T2)]
y3 = m2.T + b para todo t en el intervalo [(T1+T2) .. To]

Juntemos todo esto en un buen código:

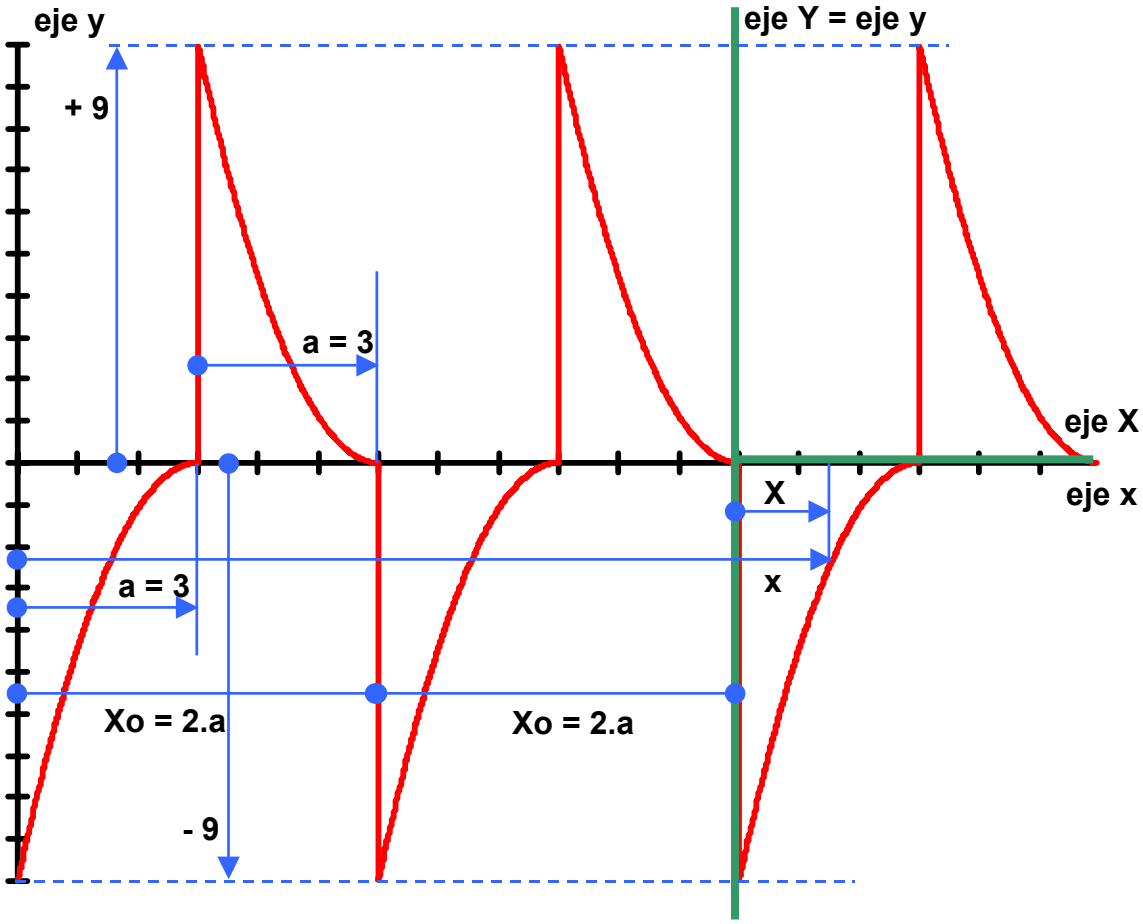
```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (        );
void ModoTexto   (        );
int  round       ( double );

// ---------------------------------------------------------
void main()
{
  int  XizqEjeH  =  100;
  int  XderEjeH  =  600;
  int  YEjeH     =  300;
  int  XEjeV     = XizqEjeH;
  int  YinfEjeV  = 100;
  int  YsupEjeV  = YEjeH;
  int  ColorFn   = LIGHTRED;
  int  ColorEjes = GREEN;
  int  ColorCal  = YELLOW;
  int  Npts      = 1500;
  int  NdivsH    =  24;

  int  NdivsV    =  20;
  int  Xp,Yp,n;

  double  T,t,y;
  double  T1       = 5;
  double  T2       = 10;
  double  T3       = 5;
  double  To       = T1+T2+T3;
  double  Vm       = 2500;
  double  m1       = Vm/T1;
  double  m2       = -Vm/T3;
  double  b        = (Vm/T3)*To;

  double  t_ini    = 0;
  double  t_fin    = 3*To;
  double  Paso_t   = (t_fin-t_ini)/Npts;
  double  EscGrafH = (double)(XderEjeH-XEjeV)/(t_fin-t_ini);
  double  EscGrafV = (double)(YEjeH-YinfEjeV)/(double)Vm;
  double  PasoCalH = (double)(XderEjeH-XEjeV)/(double)NdivsH;
  double  PasoCalV = (double)(YEjeH-YinfEjeV)/(double)NdivsV;
  double  EscGraf;
  double  EscUsH   = (t_fin-t_ini)/NdivsH;
  double  EscUsV   = Vm/NdivsV;

  char    EscUsHStr[32];
```

```
char    EscUsVStr[32];
ModoGrafico();

// --- TRAZADO DE EJES -----------------------------------

setcolor(ColorEjes);
setlinestyle(SOLID_LINE,1,THICK_WIDTH);
line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
setlinestyle(SOLID_LINE,1,NORM_WIDTH);

// --- TRAZADO DE LA CALIBRACION -------------------------
// calibracion y reticulado vertical ----------------------

for(n=0;n<=NdivsH;n++) {
    setcolor(DARKGRAY);
    line(XEjeV+round(n*PasoCalH),YEjeH,

         XEjeV+round(n*PasoCalH),YinfEjeV);
    setcolor(ColorCal);
    line(XEjeV+round(n*PasoCalH),YEjeH-4,
         XEjeV+round(n*PasoCalH),YEjeH+4);
}
// calibracion y reticulado horizontal --------------------

for(n=0;n<=NdivsV;n++) {
    setcolor(DARKGRAY);
    line(XEjeV,YEjeH-round(n*PasoCalV),XderEjeH,
         YEjeH-round(n*PasoCalV));
    setcolor(ColorCal);
    line(XEjeV-4,YEjeH-round(n*PasoCalV),
         XEjeV+4,YEjeH-round(n*PasoCalV));
}


// --- MENSAJES DE PANTALLA -------------------------------

setcolor(LIGHTGRAY);
sprintf(EscUsHStr,"EscH=%6.2lf [1/div]",EscUsH);
sprintf(EscUsVStr,"EscV=%6.2lf [V/div]",EscUsV);

outtextxy(100,80,"FUNCIONES CICLICAS");
outtextxy(100,YEjeH+10,EscUsHStr);
outtextxy(100,YEjeH+20,EscUsVStr);

setcolor(RED);
outtextxy(400,YEjeH+40,"TALLER DE LENGUAJES I");
outtextxy(400,YEjeH+50,"PROGRAMADOR UNIVERSITARIO");
outtextxy(400,YEjeH+60,"UNIV.NACIONAL DE TUCUMAN");

// ---- TRAZADO DE LA FUNCION CICLICA -------------


setcolor(ColorFn);
for(t=t_ini;t<=t_fin;t+=Paso_t) {

      Xp=XEjeV+EscGrafH*(t-t_ini);
      T=t-To*(int)(t/To);

      if(T<=T1) y=m1*T;
```

```
        else if(T<=(T1+T2)) y=Vm;
        else y=m2*T+b;

        Yp=YEjeH-round(EscGrafV*y);
        putpixel(Xp,Yp,ColorFn);
        delay(5);
  }

  getch(); ModoTexto();
}
// ------------------------------------------------------
int round(double x)
{ if((x-(int)x)>=0.5) return((int)(x+1)); return((int)x); }
// ------------------------------------------------------
```

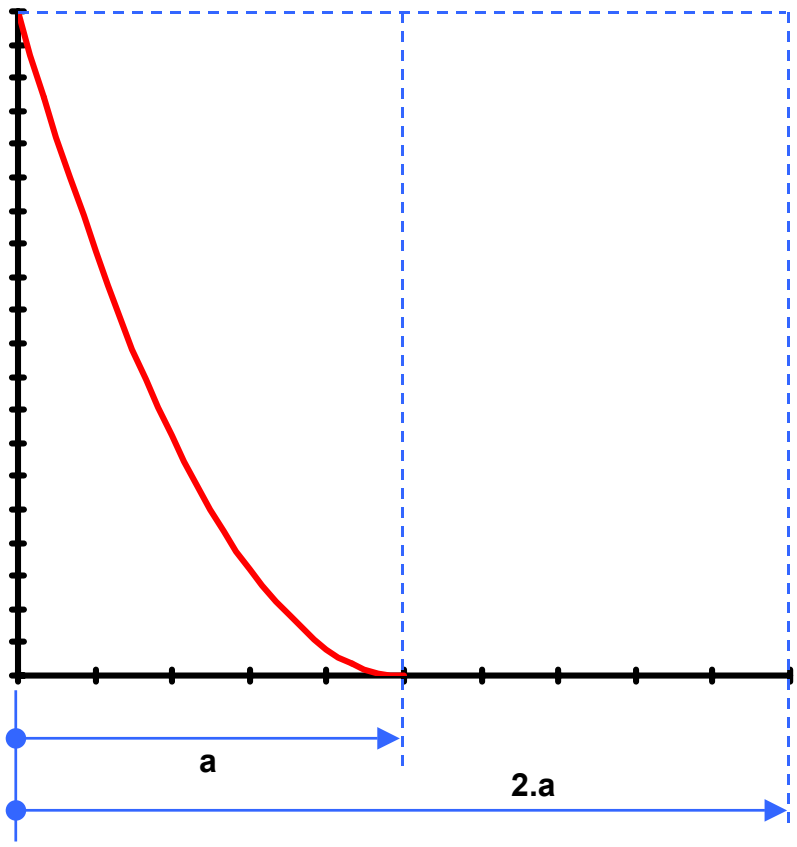
Onda cíclica parabólica.

La siguiente gráfica:



está conformada por arcos de parábola de la forma $y = x^2$, pero tomando únicamente la parte izquierda tanto en valores positivos como negativos.

Aquí lo importante es determinar las ecuaciones de estos arcos para lo cual dibujaremos:



cuya ecuación viene dada por:

$y = (x- a)^2$ para x's entre 0 y "a"

pero como en nuestro problema la curva es negativa, deberemos hacerle una pequeña modificación:

$y = -(x – a)^2$

para el mismo rango. Para el otro tramo entre a y 2.a volvemos a tener una parábola desplazada (esta vez una magnitud 2.a):

$y = (x - 2.a)^2$ para x's entre a y 2.a

Entonces en forma genérica para cualquier período:

$y = – (X – a)^2$ para X's entre 0 y a
$y = (X – 2a)^2$ para X's entre a y 2.a

X = x – n.Xo
n = (int)(x/Xo)
X = x – Xo.(int)(x/Xo)

con lo cual llegamos finalmente a:

$y = [(x – Xo*(int)(x/Xo)) – a]^2$ para X's entre 0 y a
$y = [(x – Xo*(int)(x/Xo)) – 2.a]^2$ para X's entre a y 2.a

Los códigos necesarios para la graficación son los siguientes:

```
#include<conio.h>
#include<stdlib.h>
#include<dos.h>
#include<process.h>
#include<stdio.h>
#include<math.h>
#include<graphics.h>

void ModoGrafico (        );
void ModoTexto   (        );
int  round       ( double );

// ------------------------------------------------------
void main()
{

  int   XizqEjeH  = 50;
  int   XderEjeH  = 550;
  int   YEjeH     = 240;
  int   XEjeV     = XizqEjeH;
  int   YinfEjeV  = 100;
  int   YsupEjeV  = 380;
  int   ColorFn   = LIGHTRED;
  int   ColorEjes = GREEN;
  int   ColorCal  = YELLOW;
  int   Npts      = 1800;
  int   NdivH     = 18;
  int   NdivV     = 18;
  int   Xp,Yp,n;


  double a        = 1;
  double X1       = 3;
  double X2       = 3;
  double Xo       = X1+X2;
  double X,x,y;
  double x_ini    = 0;
  double x_fin    = 3*Xo;
  double Paso_x   = (x_fin-x_ini)/Npts;
  double EscGrafH = (double)(XderEjeH-XEjeV)/(x_fin-x_ini);
  double EscGrafV = (double)(YsupEjeV-YinfEjeV)/(2*pow(a*X1,2));
  double PasoCalH = (double)(XderEjeH-XEjeV)/NdivH;
  double PasoCalV = (double)(YsupEjeV-YinfEjeV)/NdivV;
  double EscUsH   = (x_fin-x_ini)/NdivH;
  double EscUsV   = (2*pow(a*X1,2))/NdivV;

  char   EscUsHStr[32];
  char   EscUsVStr[32];

  ModoGrafico();
```

```
  // --- TRAZADO DE EJES -----------------------------

setcolor(ColorEjes);
  setlinestyle(SOLID_LINE,1,THICK_WIDTH);
  line(XizqEjeH,YEjeH,XderEjeH,YEjeH);
  line(XEjeV,YinfEjeV,XEjeV,YsupEjeV);
  setlinestyle(SOLID_LINE,1,NORM_WIDTH);

  // --- TRAZADO DE LA CALIBRACION -------------------

  setcolor(ColorCal);
  for(n=0;n<=NdivH;n++) {
      setcolor(DARKGRAY);
      line(XEjeV+round(n*PasoCalH),YinfEjeV,
         XEjeV+round(n*PasoCalH),YsupEjeV);
      setcolor(ColorCal);
      line(XEjeV+round(n*PasoCalH),YEjeH-4,
         XEjeV+round(n*PasoCalH),YEjeH+4);
  }

  for(n=0;n<=NdivV/2;n++) {

      setcolor(DARKGRAY);
      line(XEjeV,YEjeH-round(n*PasoCalV),
         XderEjeH,YEjeH-round(n*PasoCalV));
      setcolor(ColorCal);
      line(XEjeV-4,YEjeH-round(n*PasoCalV),
         XEjeV+4,YEjeH-round(n*PasoCalV));
      setcolor(DARKGRAY);
      line(XEjeV,YEjeH+round(n*PasoCalV),
         XderEjeH,YEjeH+round(n*PasoCalV));
      setcolor(ColorCal);
      line(XEjeV-4,YEjeH+round(n*PasoCalV),
         XEjeV+4,YEjeH+round(n*PasoCalV));
  }


  sprintf(EscUsHStr,"EscH=%2.2lf [1/div]",EscUsH);
  sprintf(EscUsVStr,"EscV=%2.2lf [1/div]",EscUsV);

  setcolor(LIGHTGRAY);
  outtextxy(XEjeV,80,"FUNCIONES CICLICAS");
  outtextxy(XEjeV,YsupEjeV+20,EscUsHStr);
  outtextxy(XEjeV,YsupEjeV+30,EscUsVStr);

  setcolor(BROWN);
  outtextxy(350,440,"PROGRAMADOR UNIVERSITARIO");
  outtextxy(350,450,"TALLER DE LENGUAJES I");
  outtextxy(350,460,"UNIV.NACIONAL DE TUCUMAN");

  // --- TRAZADO DE LA FUNCION CICLICA ---------------


  for(x=x_ini;x<=x_fin;x+=Paso_x) {

      X=x-Xo*(int)(x/Xo);
      if(X<=X1) y=-pow(a*(X-X1),2);
      else y=pow(a*(X-2*X1),2);
```

```
      Xp=XEjeV+EscGrafH*(x-x_ini);
      Yp=YEjeH-EscGrafV*y;

      putpixel(Xp,Yp,ColorFn);
      delay(10);
  }
  getch(); ModoTexto();
}
// ------------------------------------------------------
int round(double x)
{ if((x-(int)x)>=0.5) return((int)(x+1)); return((int)x); }
// ------------------------------------------------------
```