Práctica 4.- El algoritmo de recorte de líneas de Cohen-Sutherland y la Tortuga.

Objetivos:

Estudiar la implementación y acción del algoritmo de recorte de líneas de Cohen – Sutherland en 2D.
Uso de patrones de dibujo para la implementación de figuras complejas.

Requerimientos:

Algoritmo de recorte de C-H
Concepto de Ventana del Mundo

Introducción:

El empleo del modelo de la tortuga para generar fácilmente dibujos en 2D fue empleado por primera vez en un lenguaje denominado LOGO y diseñado por Seymour para enseñar a programar a niños. Consiste en lo siguiente:

- Tenemos una tortuga sobre un plano (para nosotros será una tortuga invisible) a la cual le podemos decir únicamente que se **desplace en línea recta** una determinada distancia. Durante ese desplazamiento, **la tortuga puede dibujar o no su rastro en el papel según nuestra voluntad**.
- La posición de esta tortuga en el plano, viene determinada por el **punto del plano donde se encuentra y la dirección a la que mira**. Cuando le ordenemos avanzar, la tortuga irá en línea recta la distancia que indiquemos, en la dirección a la que mira.
- Podemos **cambiar la dirección a la que mira** la tortuga antes de indicarle que se desplace.

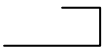
Como cambiar la dirección a través de un vector no es intuitivo, vamos a utilizar un ángulo para indicar el ángulo de giro que debe experimentar la dirección de la tortuga. El criterio será el habitual, el ángulo se medirá en sentido positivo contrario a las agujas del reloj y en **en grados sexagesimales**.

Así pues podemos implementar nuestra tortuga con estas funciones:

```
/*********************************************************/
/*INSTRUCCIONES DE LA TORTUGA*****************************/
/*********************************************************/

/*******DEFINICION DE VARIABLES GLOBALES DE CONTROL DE LA TORTUGA***/
float angulo =0.0; //angulo inicial
float posicion[2]={TAM_VENTANA_X/2.0,TAM_VENTANA_Y/2.0}; //pos inicial

/***FUNCION QUE SITUA LA DIRECCION RESPECTO DE UN ANGULO***/
void turnTo(float angle){
      angulo = angle;
}

/***FUNCION QUE GIRA LA DIRECCION UNA CANTIDAD DE GRADOS***/
void turn(float angle){
      angulo+=angle;
}

void moveTo(float x, float y){
```

```
      posicion[0] = x;
      posicion[1] = y;
}

void lineTo(float x, float y){

//ATENCION. PONER AQUI LA LLAMADA A COHEN _ SUHTHERLAND
//LAS DOS DECLARCIONES SIGUIENTES SON PARA CUANDO SE USE C-H
      int resul;
      float xdef0,ydef0, xdef1, ydef1;

glBegin(GL_LINES);
   glVertex2f(posicion[0],posicion[1]);
   glVertex2f(x,y);
   glEnd();

   posicion[0] = x;
   posicion[1] = y;
   glFlush();


void forward(float dist, int isvisible){
      const float Radsperdegree = 0.017453393;
      float x = posicion[0]+ dist * cos(Radsperdegree*angulo);
      float y = posicion[1]+ dist * sin(Radsperdegree*angulo);

      if(isvisible)
            lineTo(x,y);
    else
            moveTo(x,y);
}
```

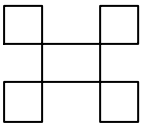
A pesar de su sencillez, es posible crear gran cantidad de motivos cuya repetición en el espacio es gráficamente muy llamativa.
Por ejemplo, el motivo



se puede crear con las siguientes instrucciones de nuestra tortuga:

```
float L=15.5;
void motivo(){
      forward(3*L,1);
      turn(90);
      forward(L,1);
      turn(90);
      forward(L,1);
      turn(90);
}
```

Bastará repetir 4 veces la llamada a esta función para crear la figura



Ej.1 Inténtalo

Vamos a crear ahora una figura más compleja y más bella y también de sorprendente simplicidad.
Ej.2 Usa esta función.

```
float length=1.0;
float incremento=2.0;
float angulo = 60;

void polyespiral(){
      int i;

      for(i=0;i<3;i++){
       forward(length,1);
         turn(angulo);
         length+= incremento;
      }

}
```

Y llámala desde la función que responde el evento de ratón. De tal manera que pinchando sucesivas veces sobre la ventana, verás que se va construyendo una figura espiral.

Prueba a darle a la variable `angulo` los valores 85.5, -144, 170.

Ejercicio 3
Ahora que te has divertido bastante vas a implementar el algoritmo de Cohen – Sutherland para recortar la figura anterior contra una ventana que vamos a dibujar dentro de nuestra área de dibujo.
En la realidad, con la Open GL se hace ya un **recorte automático** sobre la ventana del mundo definida en la instrucción `gluOrtho2D.` Como en esta práctica nos interesa hacer un recorte manual vamos a configurar la ventana del mundo a nuestros dimensiones de la la ventana windows que hace de puerto de vista, poniendo

```
gluOrtho2D(0.0,TAM_VENTANA_X,0.0,TAM_VENTANA_Y);
```

Una vez hecho esto definiremos una ventana

```
//extremos de la ventana de recorte
float xmin,ymin,xmax,ymax;

void configuraventana(){
      xmin =  TAM_VENTANA_X / 3.0;
      ymin =  TAM_VENTANA_Y / 3.0;

      xmax =   2*TAM_VENTANA_X / 3.0;
      ymax =   2*TAM_VENTANA_Y / 3.0;
}
```

Y la dibujaremos construyendo una función para ello.

Ahora implementaremos el algoritmo de Cohen – Sutherland, que tendrá el siguiente prototipo:

```
/*********************************************************************/
/* ALGORITMO DE RECORTE DE COHEN SUTHERLAND ***************************/
/* SI DEVUELVE TRUE, ES RECORTADO *************************************/
/* SI DEVUELVE FALSE, NO LO ES ****************************************/
/* EN LOS PARAMETROS POR REFERENCIA ***********************************/
/* SE ENCUENTRAN LOS NUEVOS PUNTOS DEL SEGMENTO ***********************/
/*********************************************************************/

int Cohen_Suth(float x0,float y0,float x1,float y1,float *xdef0,float *ydef0,float
*xdef1,float *ydef1);
```

donde `x0,y0,x1,y1` son los extremos del segmento a recortar
y `xdef0,ydef0,xdef1, ydef1` son los extremos recortados. En caso de aceptación o rechazo trivial, éstas variables deberán cargarse con los extremos originales (`x0,y0,x1,y1`).

Cada segmento de la figura que hemos construído en el ejercicio 2 debe pasarse a este algoritmo de recorte.
Para ello modificaremos la función `void LineTo()` de la siguiente manera:

```
void lineTo(float x, float y){

      float xdef0,ydef0, xdef1, ydef1;
      int resul;

resul =
Cohen_Suth(posicion[0],posicion[1],x,y,&xdef0,&ydef0,&xdef1,&ydef1);
      if(resul == TRUE){

    glBegin(GL_LINES);
      glColor3f(0.0,0.0,0.0);
      glVertex2f(posicion[0],posicion[1]);
    glVertex2f(xdef0,ydef0);
      glColor3f(1.0,1.0,0.0);
      glVertex2f(xdef0,ydef0);
    glVertex2f(xdef1,ydef1);
      glColor3f(0.0,0.0,0.0);
      glVertex2f(xdef1,ydef1);
      glVertex2f(x,y);
      glEnd();
      }
      else{
      glColor3f(0.0,0.0,0.0);
    glBegin(GL_LINES);
    glVertex2f(posicion[0],posicion[1]);
    glVertex2f(x,y);
    glEnd();
      }
   glFlush();
   posicion[0] = x;
   posicion[1] = y;
}
```

Esto pintará de amarillo y negro los tramos recortados y de negro únicamente los tramos trivialmente aceptados o rechazados y los que se rechazaron al final en su totalidad. Fíjate que esto depende de que des en las circunstancias adecuadas dentro del algoritmo de C-H el valor TRUE a la variable que devuelve dicho algoritmo.

La manera de funcionar el programa es sencilla: Inicialmente aparecerá pintado en un color el rectángulo de recorte, que esta centrado alrededor del centro de la ventana que es donde se inicia el dibujo.
Con cada click del ratón se irá construyendo progresivamente la espiral desde el interior de la ventana de recorte, y debemos observar como cuando ésta llega al borde de la ventana comienza a cambiar los tramos del segmento recortado a otro color (por ejemplo verde), hasta que la espiral se sale completamente del rectángulo.