EJERCICIOS DE PROGRAMACION MODULAR (17 oct- 6 nov)

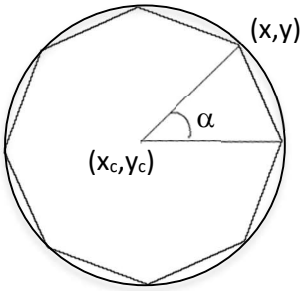
Problema 1: Dibujando polígonos

Escribe un programa en C que dibuje un polígono regular empleando la librería gráfica gfx. El programa solicitará las coordenadas (x_c, y_c) del centro del polígono y su radio r, así como el número de lados a dibujar. Una vez comprobado que no se excede de la ventana gráfica, el programa dibujará el polígono centrado en la posición (x_c,y_c) .

Para hallar los vértices del polígono puedes utilizar las siguientes ecuaciones:

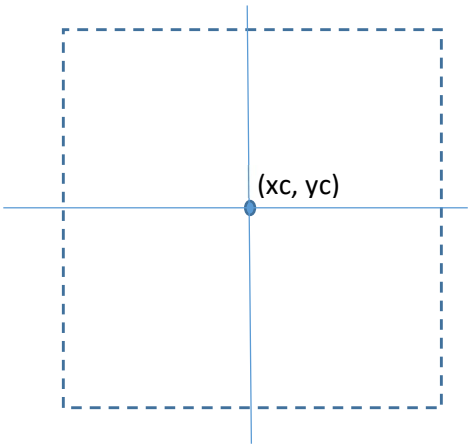
$$x = x_c + r \cdot \cos(\alpha)$$
$$y = y_c + r \cdot \sin(\alpha)$$

donde α es el ángulo correspondiente al vértice, como se observa en la figura.

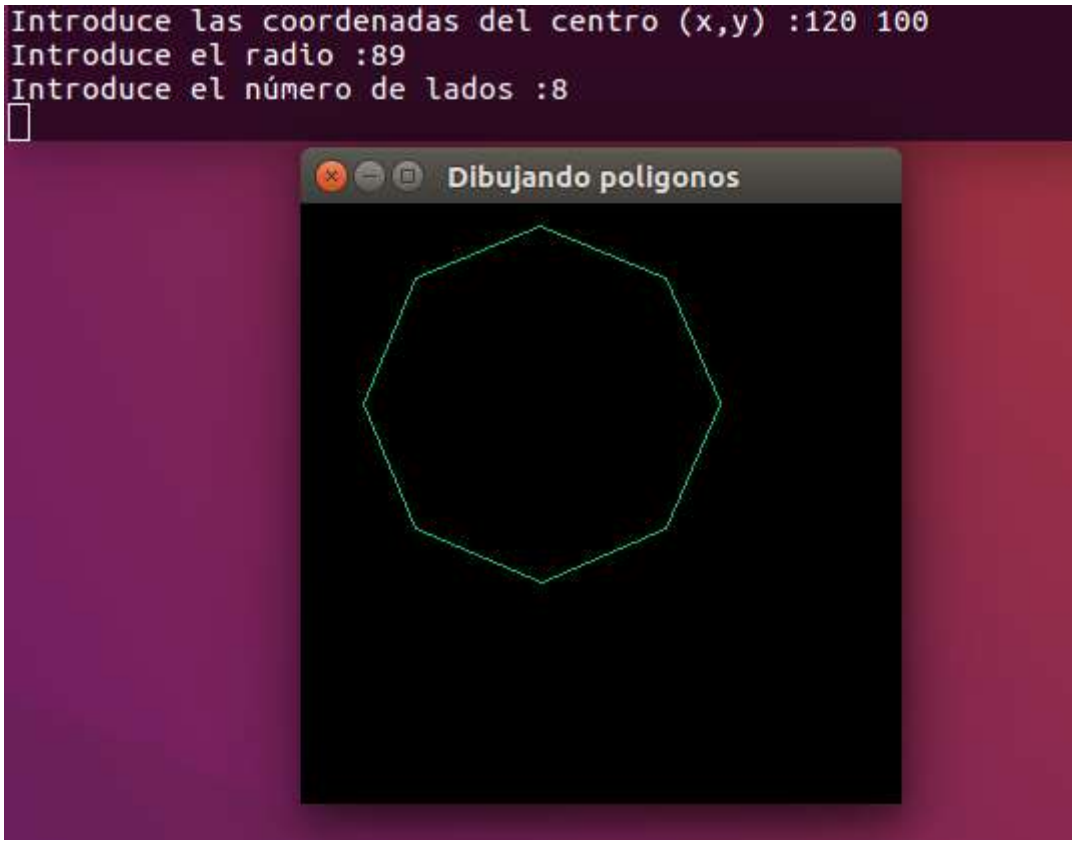


El programa debe tener como mínimo los siguientes módulos:

- `validaCoordenadas`: pide las coordenadas del centro y comprueba si algún vértice del polígono se sale de la zona de trazado. El programa debe dibujar el polígono sólo si todos los vértices se encuentran en la zona de trazado. La comprobación se hará de manera sencilla, simplemente mirando si un cuadrado centrado en (x_c, y_c) quedaría dentro de la ventana de dibujo.

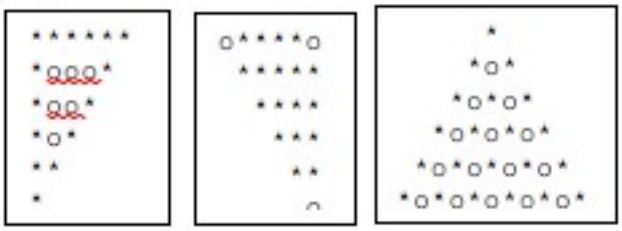


- `validaLados`: función que se encarga de pedir el número de lados del polígono y validar que se encuentra entre 4 y 15. Este módulo debe obligar al usuario a introducir un número correcto.

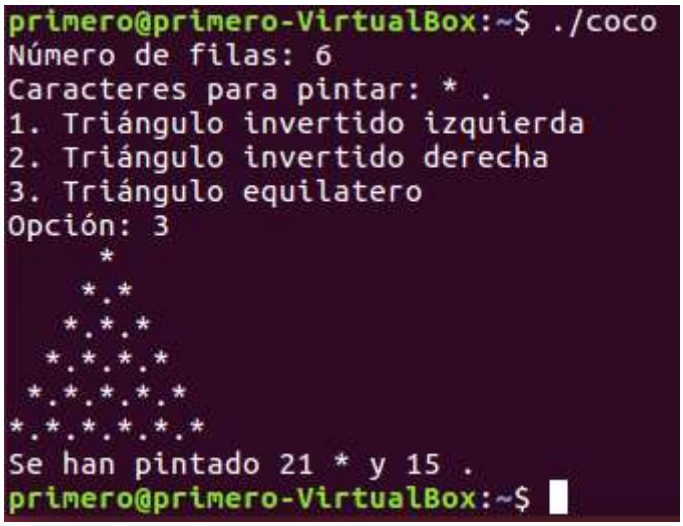


Problema 2: Dibujando triángulos

Implementa un programa que visualice en pantalla un menú que permita elegir entre las siguientes figuras para dibujar (en modo texto)21 preguntando al usuario el número de filas n que deben tener las figuras y los dos caracteres que debe usar para dibujar (En las figuras n = 6 y los caracteres son * y o). El menú y cada tipo de figura se hará en un módulo distinto. Además de dibujar la figura, el módulo debe devolver al main la cantidad de caracteres que se han pintado de cada tipo. El main mostrará estás cantidades en pantalla:



Ejemplo de ejecución



Problema 3: BlackJack

Implementa una versión simplificada del juego de cartas BlackJack, también llamado 21. En este juego se van repartiendo cartas de una en una y el objetivo del juego es llegar lo más cerca posible de 21 puntos sin pasarse. Habitualmente participa la banca y uno o más jugadores. La mecánica del juego será la siguiente:

- Para simplificar, en nuestra versión del juego solo participa un jugador y la banca.
- La banca no va a ir obteniendo cartas como el jugador, simplemente se le asignará una puntuación aleatoria entre 1 y 21.
- Al jugador se le irán "repartiendo" cartas de 1 en 1. Para esto, se generará un valor al azar entre 1 y 12. (Para simplificar, no tendremos en cuenta el palo de la carta).
- Tras "repartirle" carta al jugador se le dirá qué carta ha salido, cuántos puntos lleva, y, si no se ha pasado de 21, se le preguntará si quiere otra carta (tendrá que introducir una 's' o una 'n'). Si no introduce ninguna de las dos, se le volverá a preguntar.
- Cuando el jugador diga que no quiere más cartas, se mostrará el resultado final. Ganará el jugador si tiene más puntos que la banca y no se ha pasado de 21 (fijaos en que la banca no puede pasarse tal y como funciona nuestra versión del juego)

El programa deberá tener al menos los siguientes módulos:

- reparteCarta: debe devolver el número de la carta "repartida" (o sea, generada al azar). Tendréis que generar un entero al azar entre 1 y 12. No es misión de este módulo imprimir en pantalla qué carta ha salido, esto es trabajo del main o de otro módulo.
- calculaTotal: recibe como parámetro los puntos actuales del jugador y el número de la carta que ha salido. Devuelve el nuevo

total, sumando los puntos actuales más el valor de la nueva carta teniendo en cuenta que:

- El as (1) vale 11 puntos.
- Las figuras (cartas mayores que 9) valen 10.
- El resto de cartas tienen su valor habitual.
- No es misión de este módulo comprobar si nos hemos pasado o no de 21, simplemente debe calcular el resultado (sea el que sea).

Ejemplos de ejecución:

```
primero@primero-VirtualBox:~/P1$ ./coco
Te ha salido un 11
Por ahora tienes 10. ¿Quieres carta? (s/n)?s
Te ha salido un 2
Por ahora tienes 12. ¿Quieres carta? (s/n)?s
Te ha salido un 4
Por ahora tienes 16. ¿Quieres carta? (s/n)?s
Te ha salido un 8
Por ahora tienes 24.
La banca tenía 13
Te has pasado, ha ganado la banca
primero@primero-VirtualBox:~/P1$ ./coco
Te ha salido un 12
Por ahora tienes 10. ¿Quieres carta? (s/n)?s
Te ha salido un 4
Por ahora tienes 14. ¿Quieres carta? (s/n)?s
Te ha salido un 7
Por ahora tienes 21.
La banca tenía 20
Has ganado
```

Problema 4: Disparando a una nave

Implementa un programa utilizando la librería gfx que muestre una nave en la parte superior de la ventana y un cañón en la parte inferior. El programa debe visualizar una bala que se dispara desde el cañón hacia arriba. Si la bala impacta en la nave mostrará un mensaje indicando "Nave alcanzada", es caso contrario mostrará "Nave no alcanzada".

Consideraciones de implementación:

- La ventana tiene un tamaño de 300x300
- Nave:
  - Cuadrado denso de color azul de 20x20
  - Posición: y= 20, x se genera de manera aleatoria dentro del ancho de la ventana
- Cañón:
  - Cuadrado hueco de color rosa de 20x10

    - Posición: y= 270, x se genera de manera aleatoria dentro del ancho de la ventana
- Bala:
    - Cuadrado denso de color verde de 5x2
    - Posición inicial:
        - x: la del cañón +4 para que esté centrada
        - y: la del cañón
- Se necesita un bucle que haga avanzar la bala desde la posición del cañón hacia arriba
- Para conseguir un retardo temporal y que se vea cómo va avanzando la bala se puede usar la función `usleep()` usando como parámetro los microsegundos, p.e `usleep(50000)`. Hay que incluir la librería <`unistd.h`>
- Emplear `gfx_flush()` para volcar en la ventana gráfica los comandos de dibujo
- Módulos:
    - `genPos`: genera una posición x aleatoria dentro del rango de la ventana
    - `dibNave`: dibuja la nave
    - `dibArma`: dibuja el cañón
    - `dibBala`: dibuja la baja

La imagen muestra distintas secuencias del programa: