

INTRODUCCIÓN AL SOFTWARE

Desarrollo de un programa secuencial



Objetivos

- Familiarizarse con los procesos de compilación y ejecución de programas
- Practicar con la creación de un programa secuencial
- Realizar operaciones básicas de entrada/salida
- Uso de librerías externas

Organización

Parte 1: Compilación de un programa tipo *Hello World*

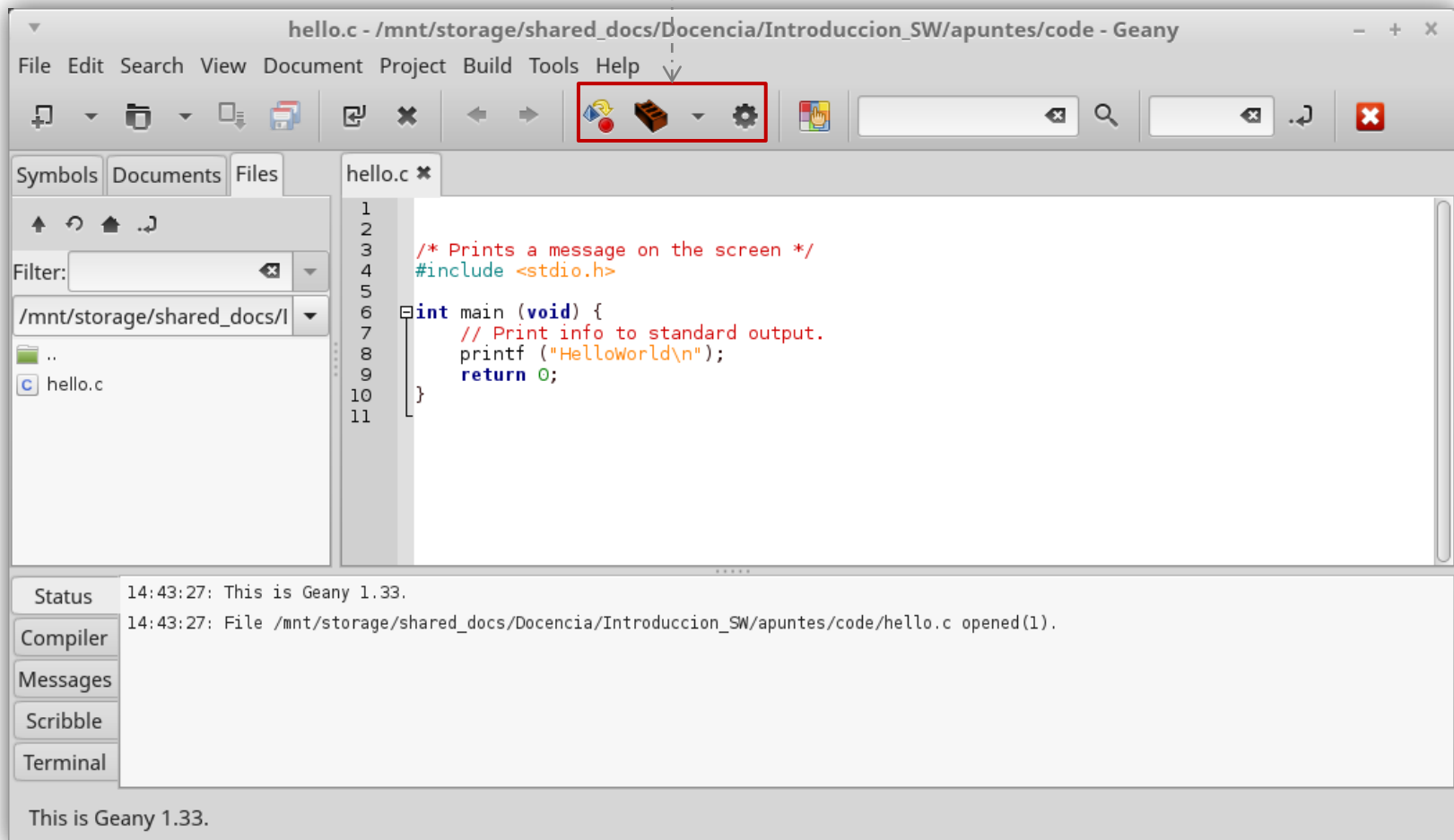
Parte 2: Compilación de un programa tipo *Hello World* utilizando una **librería gráfica**

Parte 3: Generar un programa secuencial que dibuje una figura

Parte 4: Desplazamiento de la figura (opcional)

El entorno de desarrollo Geany

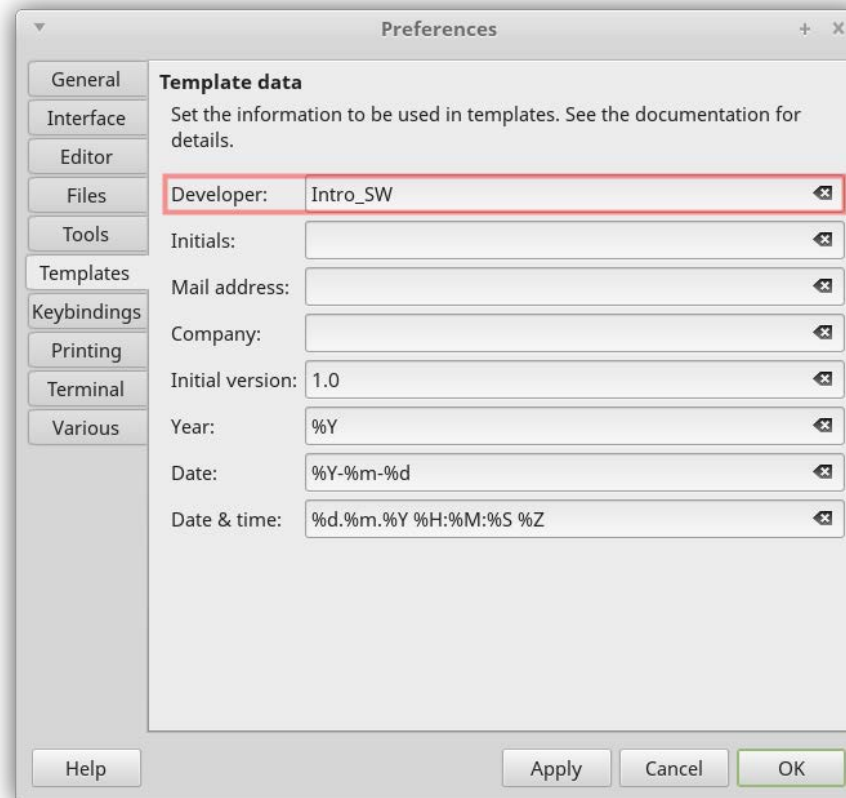
iconos de compilación



Configuración del entorno de desarrollo (1/4)

- Rellenar plantilla de datos personales

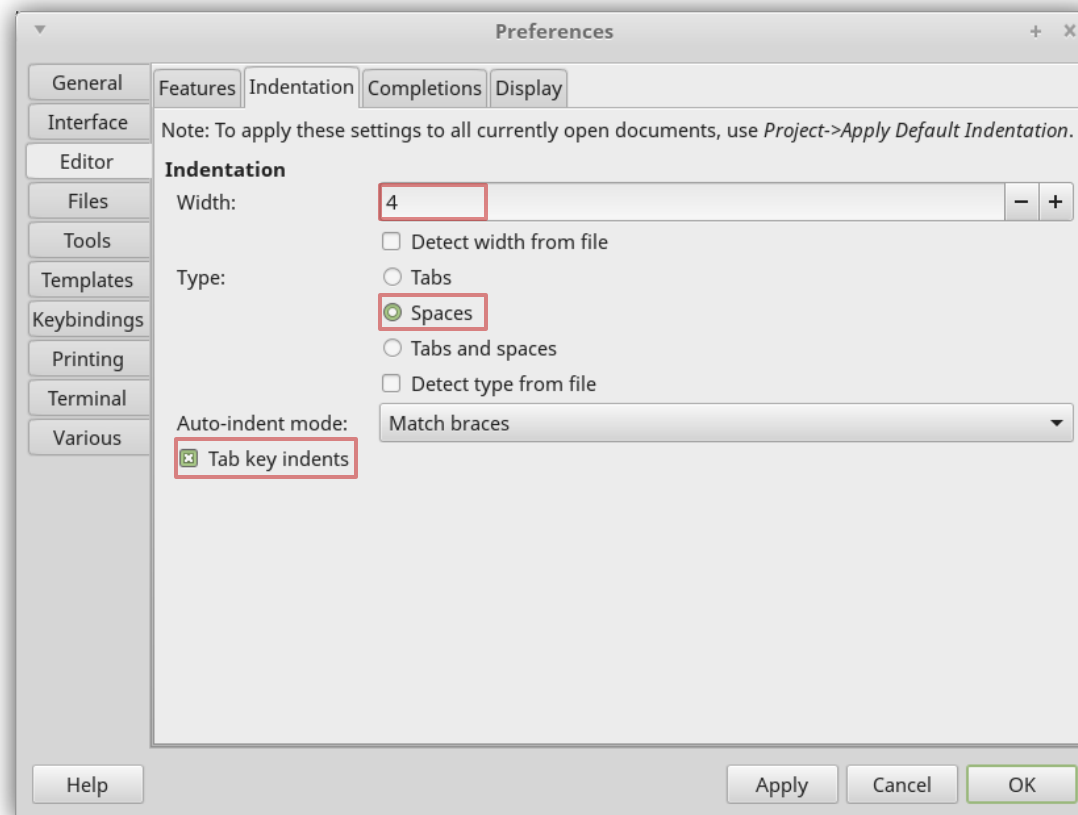
Edit->Preferences->Templates



Configuración del entorno de desarrollo (2/4)

- Configurar las opciones de sangrado del código

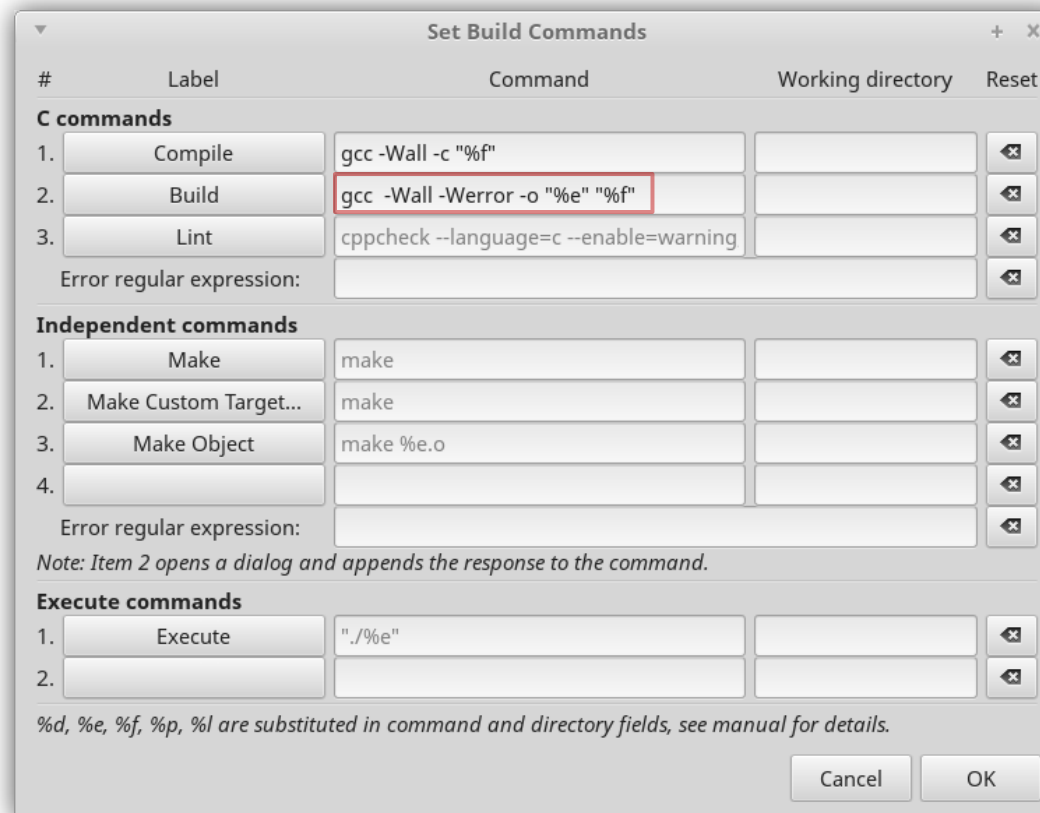
Edit->Preferences->Editor->Indentation



Configuración del entorno de desarrollo (3/4)

- Opciones de compilación

Build->Set Build Commands



The dialog box titled "Set Build Commands" contains three sections: "C commands", "Independent commands", and "Execute commands". Each section has a table with columns for #, Label, Command, Working directory, and a Reset button. The "C commands" section has three rows: Compile (gcc -Wall -c "%f"), Build (gcc -Wall -Werror -o "%e" "%f"), and Lint (cppcheck --language=c --enable=warning). The "Independent commands" section has four rows: Make (make), Make Custom Target... (make), Make Object (make %e.o), and an empty row. The "Execute commands" section has two rows: Execute ("./%e") and an empty row. A note at the bottom states: "Note: Item 2 opens a dialog and appends the response to the command." A footer note says: "%d, %e, %f, %p, %l are substituted in command and directory fields, see manual for details." Buttons for Cancel and OK are at the bottom right.

#	Label	Command	Working directory	Reset
C commands				
1.	Compile	gcc -Wall -c "%f"		
2.	Build	gcc -Wall -Werror -o "%e" "%f"		
3.	Lint	cppcheck --language=c --enable=warning		
Error regular expression:				
Independent commands				
1.	Make	make		
2.	Make Custom Target...	make		
3.	Make Object	make %e.o		
4.				
Error regular expression:				
Execute commands				
1.	Execute	"./%e"		
2.				

Note: Item 2 opens a dialog and appends the response to the command.

%d, %e, %f, %p, %l are substituted in command and directory fields, see manual for details.

Cancel OK

Uso del entorno de desarrollo

- Nuevo fichero C a través de la plantilla
File->New file (with template)->main.c
- Autocompletado básico pulsando *control+espacio*
 - autocompleta los identificadores definidos en el fichero actual
 - autocompleta los identificadores definidos en el resto de ficheros abiertos
 - p.ej., al abrir un fichero de cabecera el IDE nos mostrará todas sus funciones
- Iconos para compilar, construir y ejecutar desde el IDE
- Terminal integrado

Herramientas de apoyo

- Al escribir código fuente suele ser habitual seguir una **guía de estilo**
 - consulta los apuntes disponibles en el anexo del tema 1.2
- Nos apoyaremos en la herramienta **is_style** para cumplir con **algunas** de las recomendaciones de estilo de la asignatura
 - código de colores:
 - verde (añadir), rojo (eliminar), amarillo (tabuladores)

```
[marte@.../style]$ is_style hello_tabs.c
```

```
#include <stdio.h>
int main (void) {

    int a = 5;
int b;

    printf ("%d", a + b)
    printf ("HelloWorld\n");}
```

```
#include <stdio.h>
int main (void) {

    int a = 5;
int b;

    printf ("%d", a + b)
    printf ("HelloWorld\n");
}
```

And consider adding more comments!

Parte 1: Programa tipo Hello World

- Copia el programa *Hello World* de los apuntes en el fichero *hola.c*
 - compila y ejecuta el programa desde el *IDE*
 - compila y ejecuta el programa desde el *terminal*
 - podemos usar el terminal integrado en geany para introducir el siguiente comando

```
gcc -Wall -Werror -o hola hola.c
```

- el flag *-Wall* indica que se muestren todos los *warnings* de compilación
 - el flag *-o* indica el nombre del fichero ejecutable a generar
 - comprueba el estilo con la herramienta *is_style*
 - recordad hacerlo siempre antes de cada entrega

```
is_style hola.c
```

Parte 2: Programa tipo Hello World con librería gráfica

- Descarga el fichero *[hola_grafico.c](#)*

```
#include "dibujo.h"

int main (void) {

    // Creamos el dibujo
    crea_ventana ("Hola gráfico con la librería dibujo", 400, 400);

    // Creamos un texto en el dibujo
    crea_texto (10, 50, "Hola mundo", "negro", 40);
    // Pintamos el dibujo y esperamos a que se cierre la ventana
    pinta_y_espera();

    // Destruimos el dibujo
    destruye_ventana();

    return 0;
}
```

Parte 2: Programa tipo Hello World con librería gráfica

- El programa previo muestra un texto en una ventana utilizando la librería gráfica **libdibujo**
- Identifica las funciones de la librería que se utilizan en el programa, así como la funcionalidad que proporcionan cada una de ellas
 - para ello, consulta la documentación de la librería y/o el anexo de la práctica
- Compila y ejecuta el programa
 - para ello, sigue las instrucciones proporcionadas en la siguiente transparencia
- Una vez que compruebes que el programa funciona correctamente, añade las **variables** que consideres necesarias para mejorar la **legibilidad** del programa

Parte 2: Programa tipo Hello World con librería gráfica

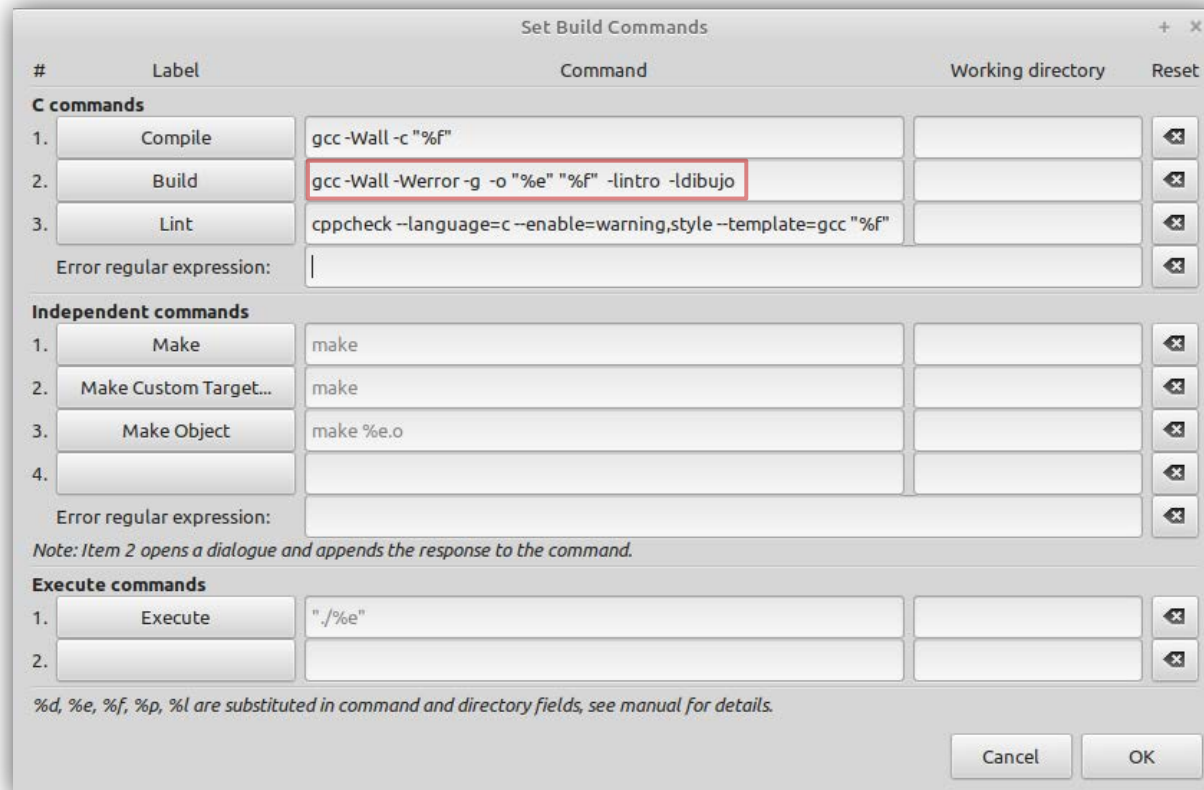
- Compilar y ejecutar el programa
 - desde el **terminal**
 - introducir el siguiente comando

```
gcc -Wall -Werror -o hola_grafico hola_grafico.c -ldibujo
```
 - la librería *dibujo* no forma parte de la librería estándar de C, por lo que debemos indicar al compilador que queremos utilizarla mediante el flag **-l** y el **nombre** de la librería
 - desde el **IDE**
 - recuerda añadir las librerías a las opciones de compilación en el lugar que se muestra a continuación

Configuración del entorno de desarrollo (4/4)

- Opciones de compilación

Build->Set Build Commands



The dialog box titled "Set Build Commands" contains a table with columns: #, Label, Command, Working directory, and Reset. It is divided into three sections: C commands, Independent commands, and Execute commands.

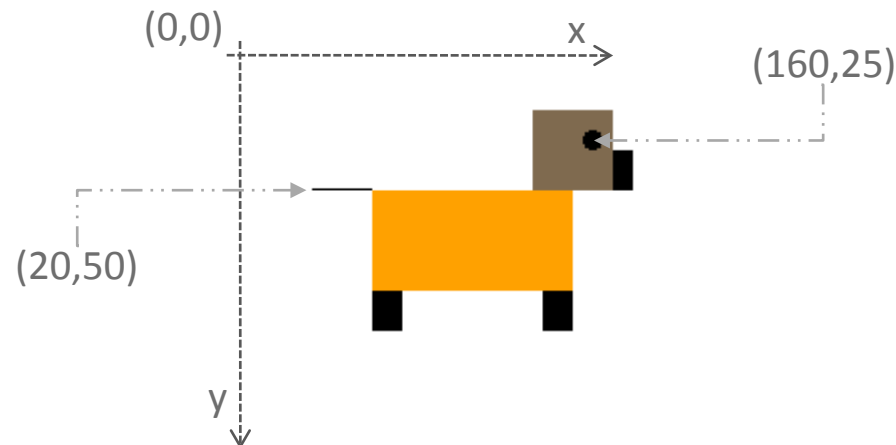
#	Label	Command	Working directory	Reset
C commands				
1.	Compile	gcc -Wall -c "%F"		
2.	Build	gcc -Wall -Werror -g -o "%e" "%F" -lintro -ldibujo		
3.	Lint	cppcheck --language=c --enable=warning,style --template=gcc "%F"		
Error regular expression:				
Independent commands				
1.	Make	make		
2.	Make Custom Target...	make		
3.	Make Object	make %e.o		
4.				
Error regular expression:				
<i>Note: Item 2 opens a dialogue and appends the response to the command.</i>				
Execute commands				
1.	Execute	"/%e"		
2.				

%d, %e, %f, %p, %l are substituted in command and directory fields, see manual for details.

Cancel OK

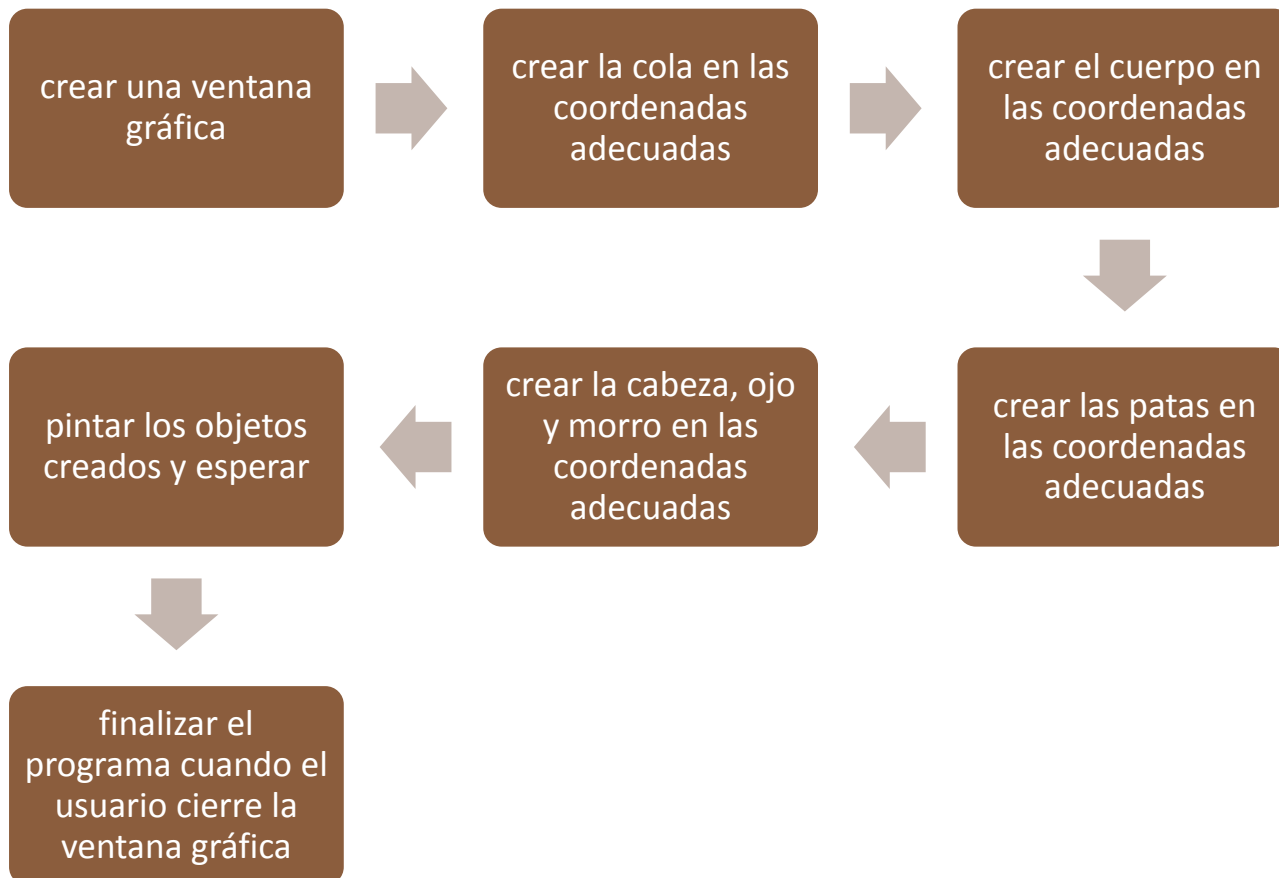
Parte 3: Programa secuencial con librería gráfica

- Dibuja la figura de un perrete utilizando la librería gráfica **libdibujo**
 - la figura presenta las siguientes medidas (unidades en pixeles)
 - cola: línea con largo = 30
 - cuerpo: rectángulo con largo = 100 , alto = 50
 - patas: rectángulo con largo = 15, alto = 20
 - cabeza: rectángulo con largo = 40, alto = 40
 - morro: rectángulo con largo = 10, alto = 20
 - ojo: círculo con radio = 5



Parte 3: Programa secuencial con librería gráfica

- Etapas del algoritmo a desarrollar

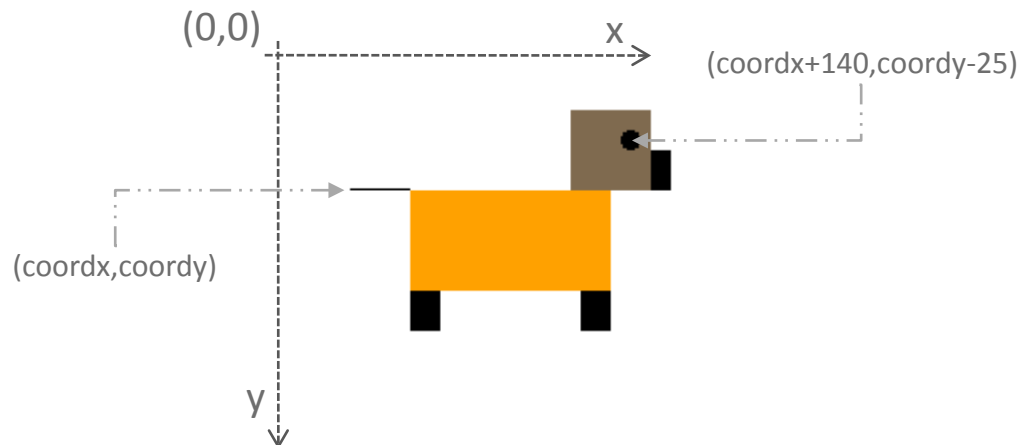


Parte 3: Programa secuencial con librería gráfica

- Descarga el fichero *perrete.c* que contiene una implementación parcial del programa
 - completa el código para que implemente cada una de las etapas del algoritmo propuesto
 - sigue las instrucciones dadas por los comentarios “*TODO*”
 - consulta el **anexo de la práctica** para ver las funciones que necesitamos para dibujar *círculos, rectángulos y líneas rectas*
 - consulta la documentación de la librería en caso necesario
 - recuerda que el tipo de dato *string* está incluido en la librería de la asignatura *libintro*
 - compila y ejecuta el programa

Parte 4: Desplazamiento de la figura (opcional)

- Crea un programa *perrete_despl.c* a partir del programa anterior que permita dibujar la figura en otro lugar de la ventana gráfica con el *mínimo* número de cambios en el código fuente
 - podemos declarar dos variables que representen el punto de origen del dibujo
 - las coordenadas de los objetos se expresarían en función de esas variables
 - comprobar que el código implementado es correcto dibujando el inicio de la cola del perrete en la *coordenada (150, 150)* de la ventana gráfica



Entrega

- Incluir los siguientes ficheros (sin comprimir):
 - el código fuente (ficheros con extensión .c) de los tres programas generados
 - hola.c
 - hola_grafico.c
 - perrete.c
 - si procede, el código fuente de la parte opcional
 - perrete_despl.c



Anexo: Uso de la librería gráfica

- La librería gráfica *dibujo* define un conjunto de strings o cadenas de caracteres para expresar los **colores** más habituales
 - ejemplos

"gris claro"	"naranja"	"lima"	"transparente"
"gris"	"rosa"	"verde oscuro"	"violeta"
"gris oscuro"	"rojo"	"azul celeste"	"purpura"
"amarillo"	"marron"	"azul"	"blanco"
"dorado"	"verde"	"azul oscuro"	"negro"

Anexo: Uso de la librería gráfica

- Crear la ventana gráfica e inicializar los recursos
 - mediante la función *crea_ventana*
 - toma como parámetros el título de la ventana, la anchura y la altura (en píxeles)
 - esta función debe invocarse antes que cualquier otra función de la librería
 - ejemplo: crear una ventana gráfica cuadrada de 300 píxeles de lado con el título “Ejemplo”

```
crea_ventana (“Ejemplo”, 300, 300);
```

- Cerrar la ventana gráfica y liberar los recursos
 - mediante la función *destruye_ventana*
 - no recibe ningún parámetro de entrada
 - ejemplo:

```
destruye_ventana();
```

Anexo: Uso de la librería gráfica

- Dibujar una línea: mediante la función *crea_linea*
 - toma como parámetros las coordenadas (x,y) del punto inicial (en píxeles), las coordenadas (x,y) del punto final (en píxeles) y el color
 - ejemplo: pintar una línea marrón entre los puntos (50, 100) y (100, 150)
`crea_linea (50, 100, 100, 150, "marron");`
- Dibujar un círculo: mediante la función *crea_circulo*
 - toma como parámetros las coordenadas (x,y) del centro (en píxeles), el radio y el color
 - ejemplo: pintar un círculo dorado de radio 20 y centrado en las coordenadas (50,60)
`crea_circulo (50, 60, 20, "dorado");`
- Dibujar un rectángulo: mediante la función *crea_rectangulo*
 - toma como parámetros las coordenadas (xmin,ymin) de la esquina superior izquierda (en píxeles), las coordenadas (xmax,ymax) de la esquina inferior derecha (en píxeles) y el color
 - ejemplo: pintar un rectángulo naranja entre los puntos (50, 100) y (150, 150)
`crea_rectangulo (50, 100, 150, 150, "naranja");`

Anexo: Uso de la librería gráfica

- Dibujar un texto
 - mediante la función *crea_texto*
 - toma como parámetros las coordenadas (x,y) del inicio del texto (en píxeles), el texto, el color y el tamaño de la fuente
 - ejemplo: escribir “Hola” en marrón a partir del punto (0, 0) con un tamaño de fuente de 24

`crea_texto (0, 0, “Hola”, “marron”, 24);`

- Pintar el dibujo y esperar a que el usuario cierre la ventana
 - mediante la función *pinta_y_espera*
 - no recibe ningún parámetro de entrada
 - debe invocarse después de todas las operaciones de dibujo
 - ejemplo:

`pinta_y_espera();`