MAKEFILE

// Primero vamos a crear las ‘variables’.

// Las creamos con el nombre que le queramos asignar, en mayúsculas, seguido del signo ‘=’.

// Detrás del ‘=’ escribiremos todos los valores que queramos asociar a dicha ‘variable’

// Cuando creemos reglas y escribamos el nombre de la variable, será como escribir todos los valores asociados

**NAME = libftprintf.a**

// Estamos creando una variable para poner nombre al archivo ejecutable que vamos a generar.

// NAME es el nombre de la variable. Puedes poner el nombre que quieras (por ejemplo EXEC)

// Detrás del = vamos a poner el nombre del archivo ejecutable que queremos crear.

**SRCS = archivo1.c archivo2.c archivo3.c \**

**archivo4.c archivo5.c**

// En esta variable especificamos los nombres de los .c que se enlazarán para hacer el ejecutable.

// Los ponemos separados por espacios y con ‘\’ al final de la línea si necesitamos líneas adicionales.

// Como siempre, puedes ponerle el nombre que quieras a la variable, pero mejor si es significativo.

**OBJS = ${SRCS:.c=.o}**

// Para usar una variable escribes ‘$’ seguido de () o {} y dentro pones el nombre de la variable.

// En este caso estamos diciendo que en la variable OBJS vamos a guardar los archivos .o que tomará de la variable SRCS donde están los archivos .c

**CC = gcc**

// En esta variable vamos a guardar el compilador que vamos a utilizar. En este caso ‘gcc’.

**CFLAGS = -c -Wall -Wextra -Werror -g**

// Guardamos los flags con los que vamos a compilar

// Puedes añadir ‘-g’ si quieres utilizar un depurador de código

// Además de los flags obligados de 42 (all, extra, error) estoy compilando con -g (para poder depurar)

**RM = rm -f**

// Borra los archivos. Por defecto viene con el flag -f (que creo que significa que lo haga sí o sí)

// Ahora vamos a crear las reglas

// Se escriben de la siguiente manera

nombre: dependencias

(tab) instrucciones

// Primero el nombre de la regla. Después ‘:’

// Después de los dos puntos se escriben las dependencias, que son los elementos necesarios para que se puedan ejecutar las instrucciones. Por ejemplo: para crear los .o necesita tener los .c

// Y finalmente las instrucciones en el orden que hay que hacerlas.

**all: $(NAME)**

// La regla ‘all’ es una regla estándar que indica qué regla se va a ejecutar si no le dices otra cosa

// Si escribes ‘make’ a secas, ejecutará la primera regla. Si pones ‘make’ y el nombre de una regla, ejecutará esa regla. Si pones ‘make’ a secas y hay una regla ‘all’, ejecutará esa regla.

// Depende de la variable NAME que es nuestro archivo ejecutable

**$(NAME): $(SRCS)**

**$(CC) $(CFLAGS) $(SRCS)**

**ar -rc $(NAME) $(OBJS)**

**ranlib $(NAME)**

// La primera línea tiene el nombre y las dependencias. El nombre será el del archivo ejecutable que queremos crear, que está en la variable NAME. Y detrás de los dos puntos lo que vamos a necesitar los .c, que están en la variable SRCS.

// En la segunda línea empiezan las instrucciones. Se escriben después de un TABULADOR.

// La segunda línea es como escribir: ‘gcc’ ‘-c -Wall -Wextra -Werror -g’ ‘todos los .c’ en terminal. Primero ponemos el compilador (que lo tenemos en la variable CC), luego los flags con los que queremos compilar (CFLAGS), y luego el listado de archivos .c (SRCS).

// La tercera línea es donde se crea el archivo ejecutable, adjuntando todos los archivos objeto (.o). Pero para que el enlazador pueda usar ese ejecutable necesita un índice.

// En la cuarta línea creamos ese índice (tabla de símbolos)

// En teoría se puede quitar la última línea poniendo en la anterior ar -rc -s que hace lo mismo

// Para más info sobre ‘ar’ y ‘ranlib’:

https://stackoverflow.com/questions/47910759/what-is-the-difference-between-ranlib-ar-and-ld-for-making-libraries

**clean:**

**$(RM) $(OBJS)**

// Usa la variable RM (que contiene las órdenes rm -f) para borrar los archivos OBJS (es decir, los .o)

**fclean: clean**

**$(RM) $(NAME)**

// Si llamas a la regla fclean, primero va a llamar a la regla ‘clean’ (porque la tiene como dependencia), y luego va a borrar también el archivo ejecutable y las demás cosas que creó

**re: fclean all**

// Creo que si lo ejecutas escribiendo ‘make re’ en terminal, primero te borra todo (hace ‘fclean’) y luego crea todo otra vez llamando a la regla ‘all’.

**.PHONY: all clean fclean re**

// Indica que estas reglas no están asociadas a archivos concretos. Creo que sirve para prevenir si se escriben archivos con los nombres de esas reglas