
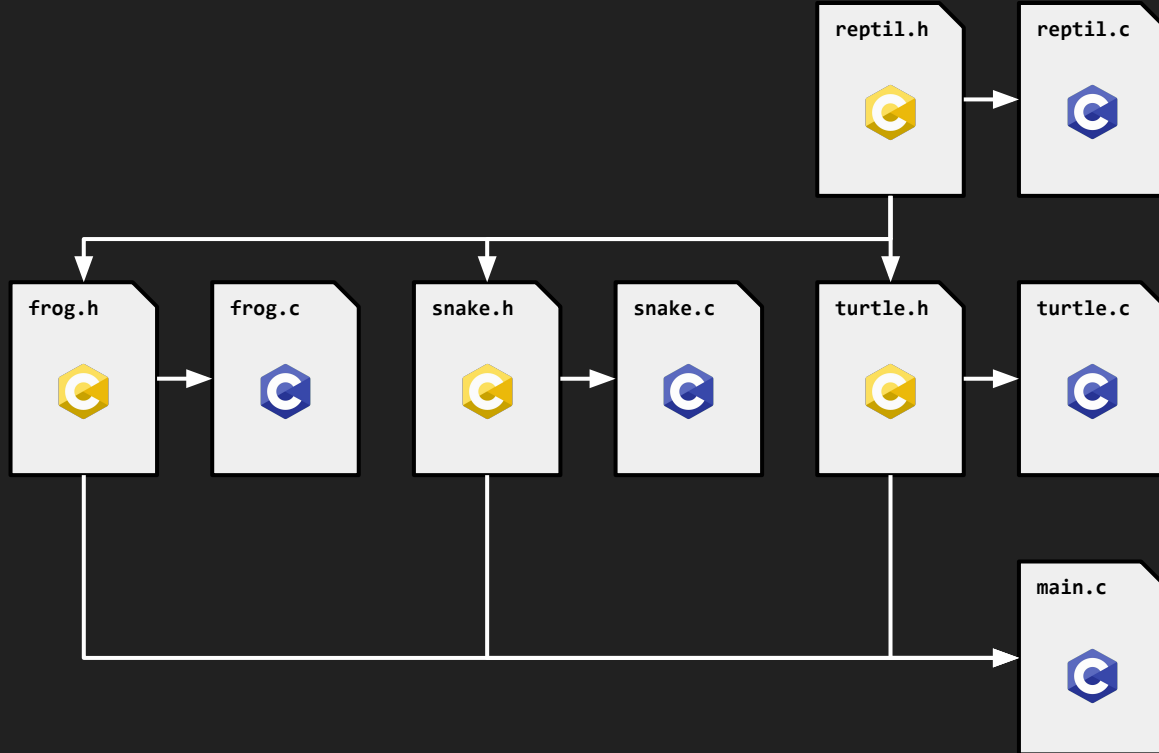


# Intro a C

## Make

With  by @vichoeq & @KnowYourselfs

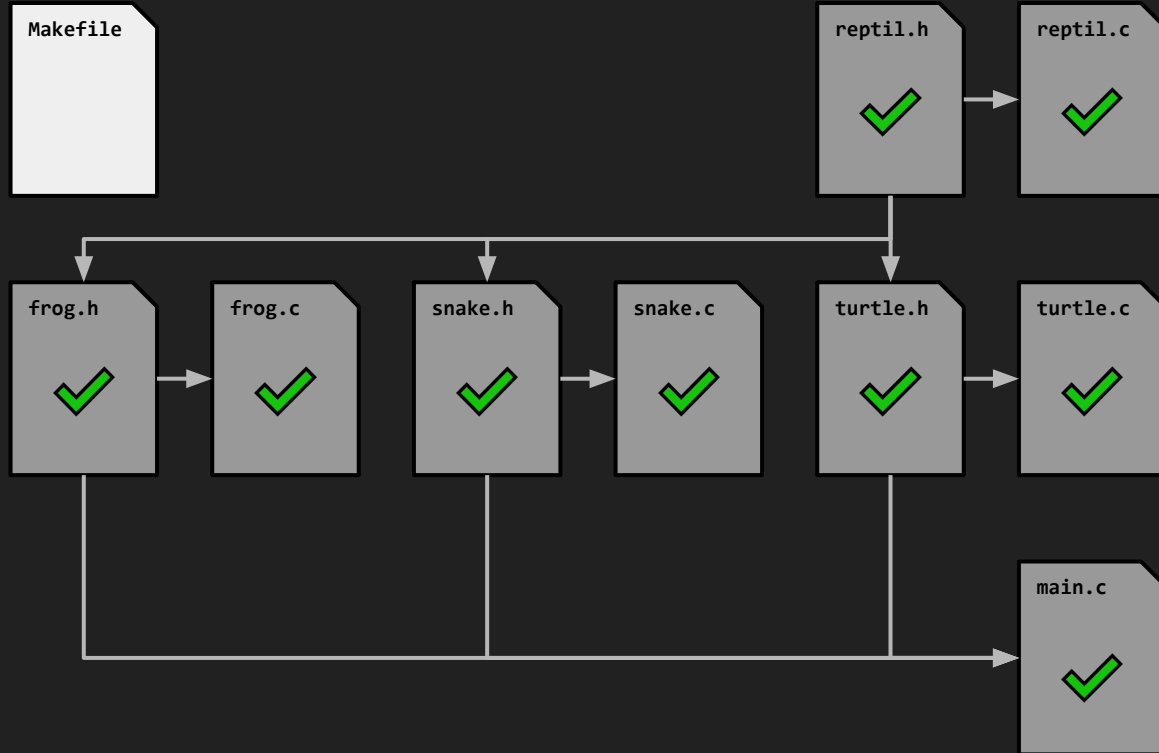
# Compilando programa con módulos



```
$ gcc reptil.c -c -o reptil.o
$ gcc frog.c -c -o frog.o
$ gcc snake.c -c -o snake.o
$ gcc turtle.c -c -o turtle.o
$ gcc main.c -c -o main.o
$ gcc reptil.o frog.o snake.o
  turtle.o main.o -o main
$ ./main
```



# Compilando programa con módulos



```
$ make main  
$ ./main
```



# GNU Make

La idea de `Make` es identificar qué debe ser re-compilado luego del cambio de un archivo, para no compilar de más.

Para esto, en el archivo `Makefile` se definen las dependencias entre los archivos del programa en el formato de `reglas`.

# Makefile

# Reglas

Las **reglas** de **Make** se definen de la siguiente manera:

```
regla: condicion_1 condicion_2 ... condicion_n  
      comando
```

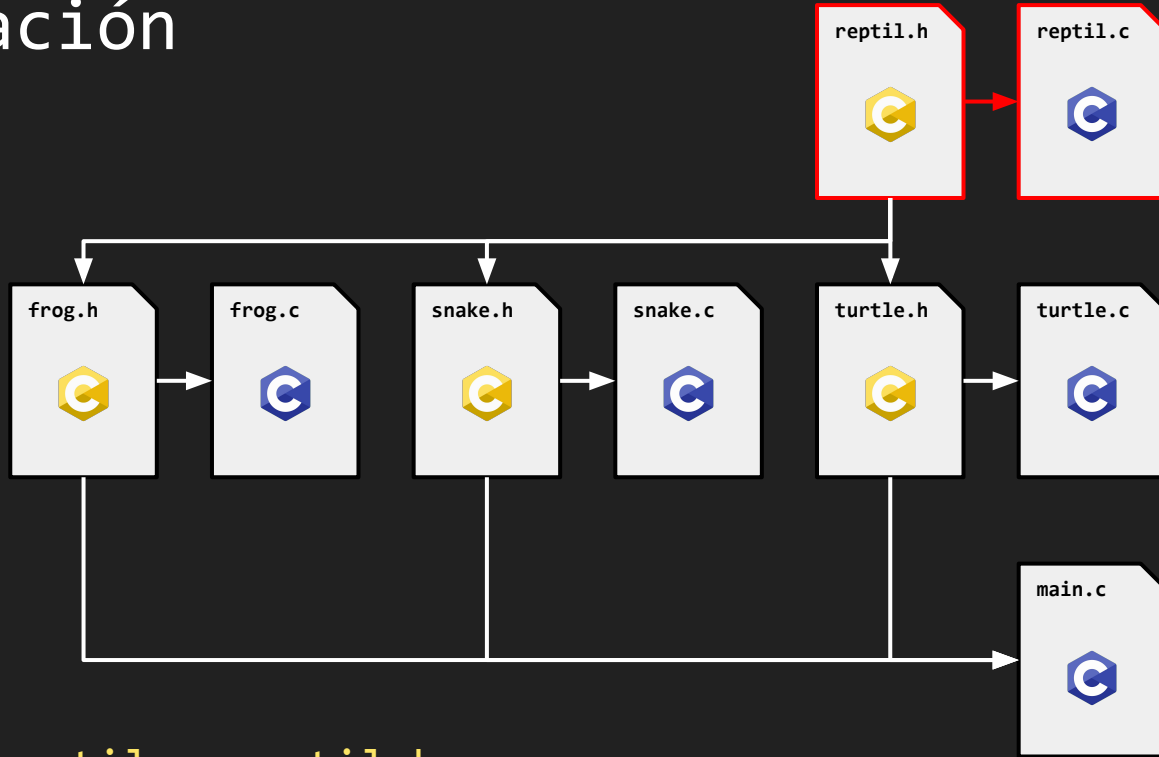
Al invocar una **regla** se ejecuta su **comando**, pero sólo si se cumple alguna de sus **condiciones**

# Condiciones

Una `condicion` se puede cumplir de dos maneras:

- Si `condicion` es el nombre de una `regla`, se invoca. Si esta ejecuta su comando, entonces se considera como cumplida.
- Si `condicion` es el nombre de un `archivo` existente, y este ha sido modificado desde la última vez que se ejecutó la `regla`.

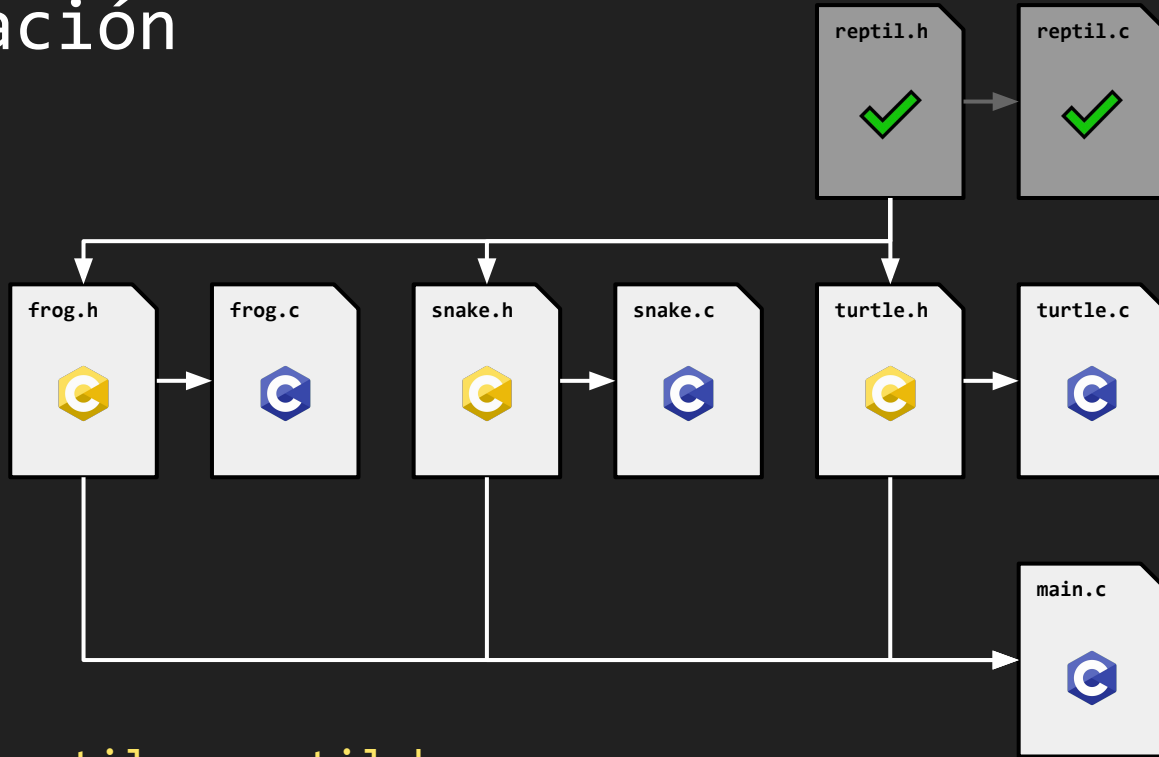
# Compilación



```
reptil.o: reptil.c reptil.h  
gcc reptil.c -c -o reptil.o
```

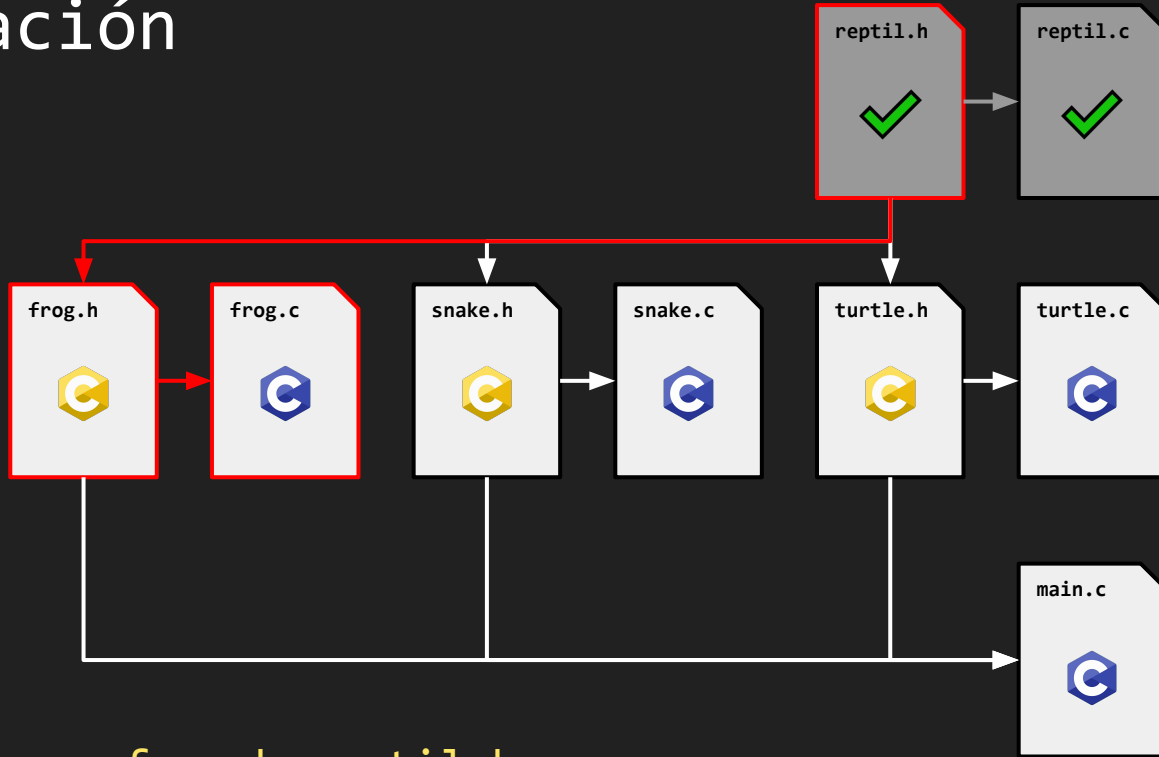


# Compilación



```
reptil.o: reptil.c reptil.h  
gcc reptil.c -c -o reptil.o
```

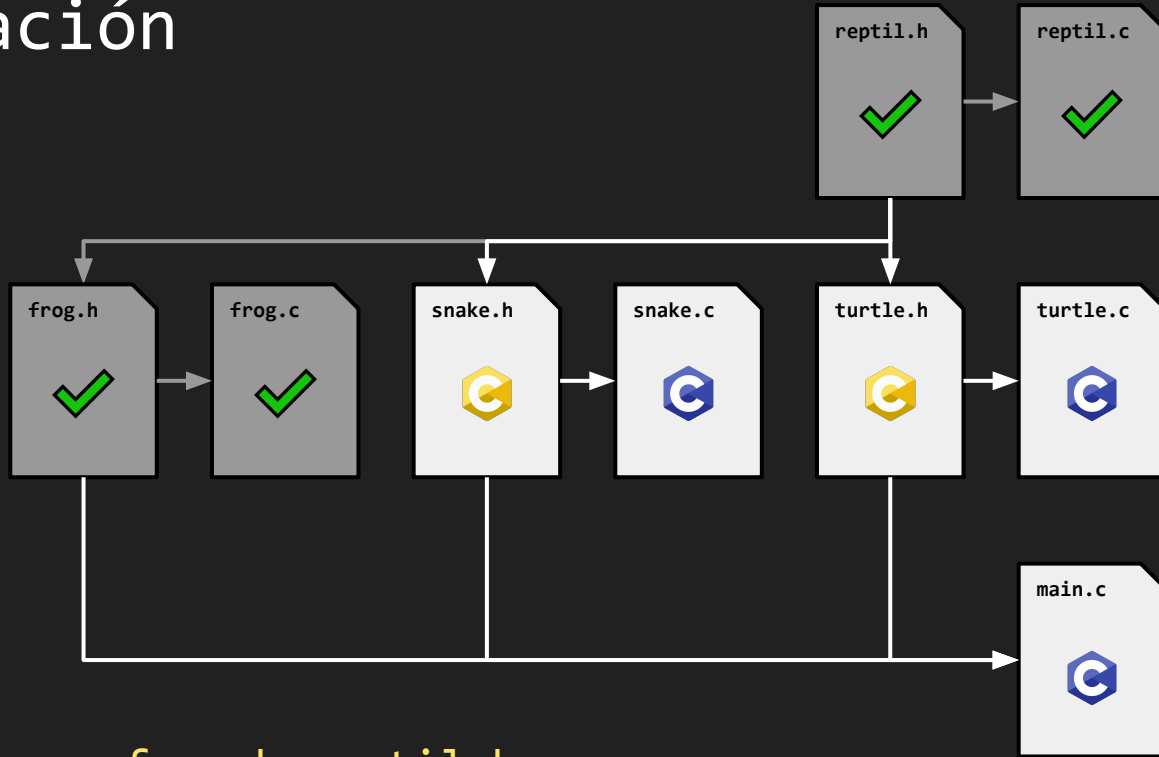
# Compilación



```
frog.o: frog.c frog.h reptil.h
```

```
gcc frog.c -c -o frog.o
```

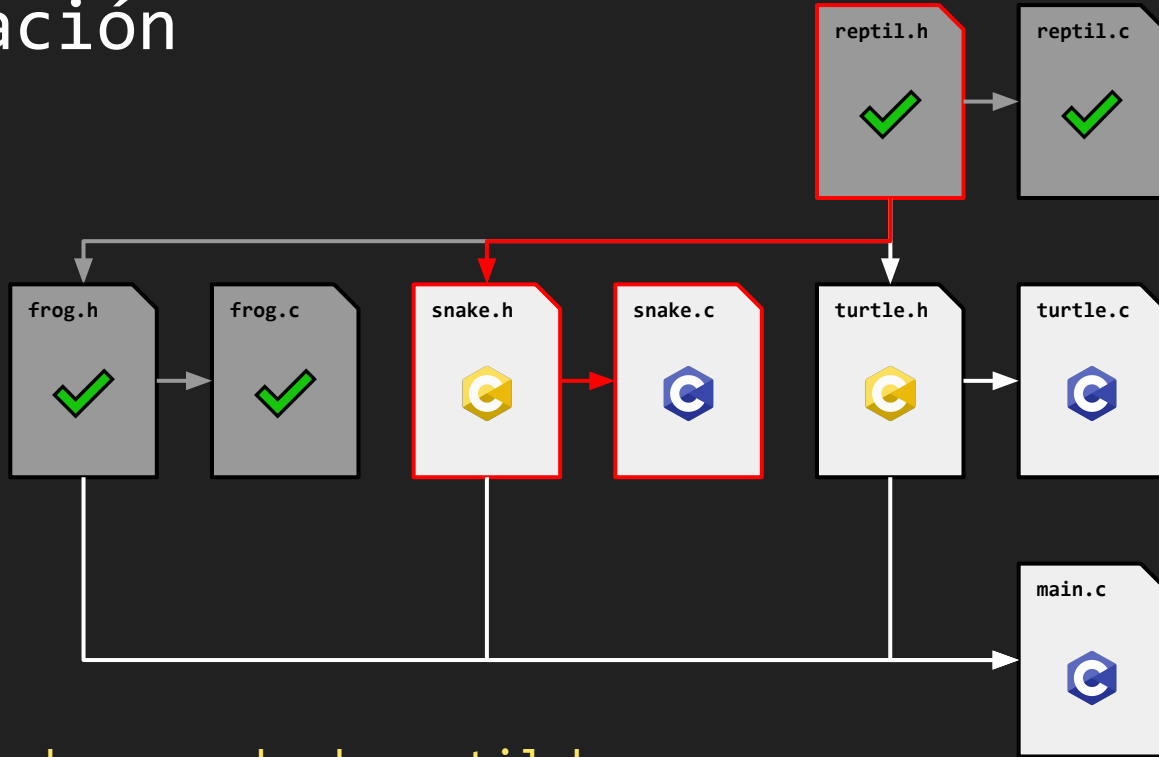
# Compilación



`frog.o: frog.c frog.h reptil.h`

`gcc frog.c -c -o frog.o`

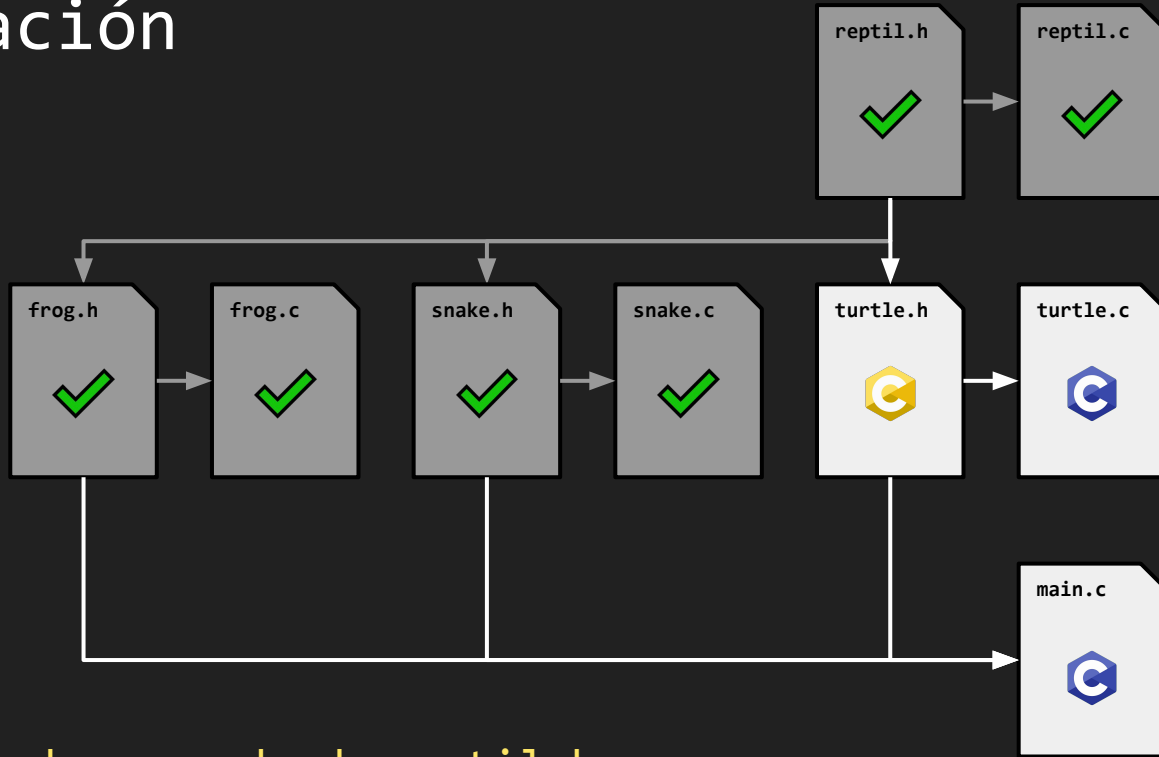
# Compilación



`snake.o: snake.c snake.h reptil.h`

`gcc snake.c -c -o snake.o`

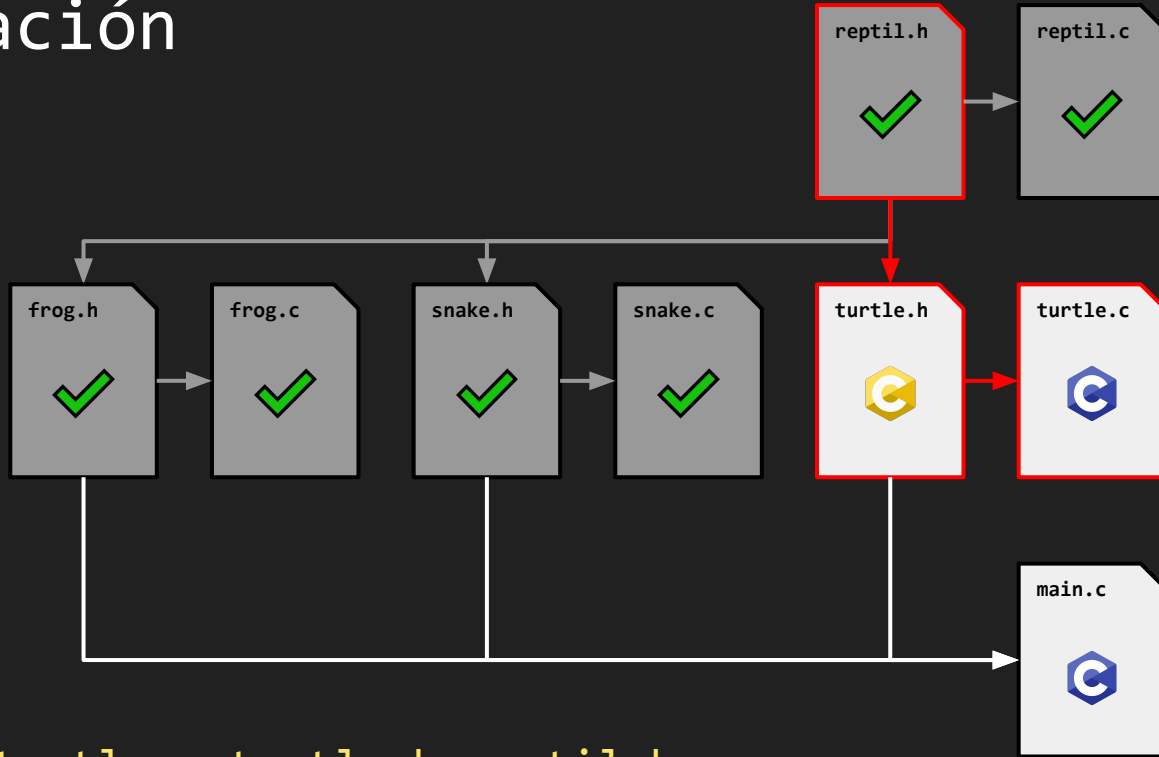
# Compilación



snake.o: snake.c snake.h reptil.h

gcc snake.c -c -o snake.o

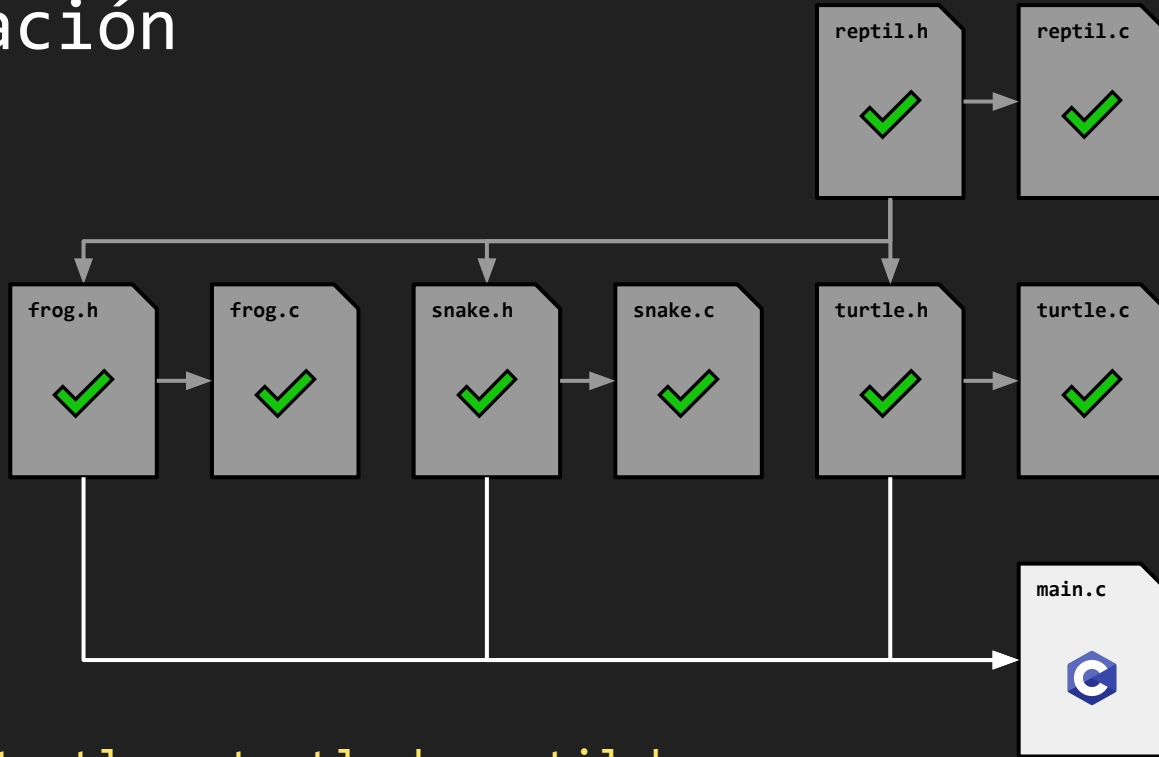
# Compilación



```
turtle.o: turtle.c turtle.h reptil.h
```

```
gcc turtle.c -c -o turtle.o
```

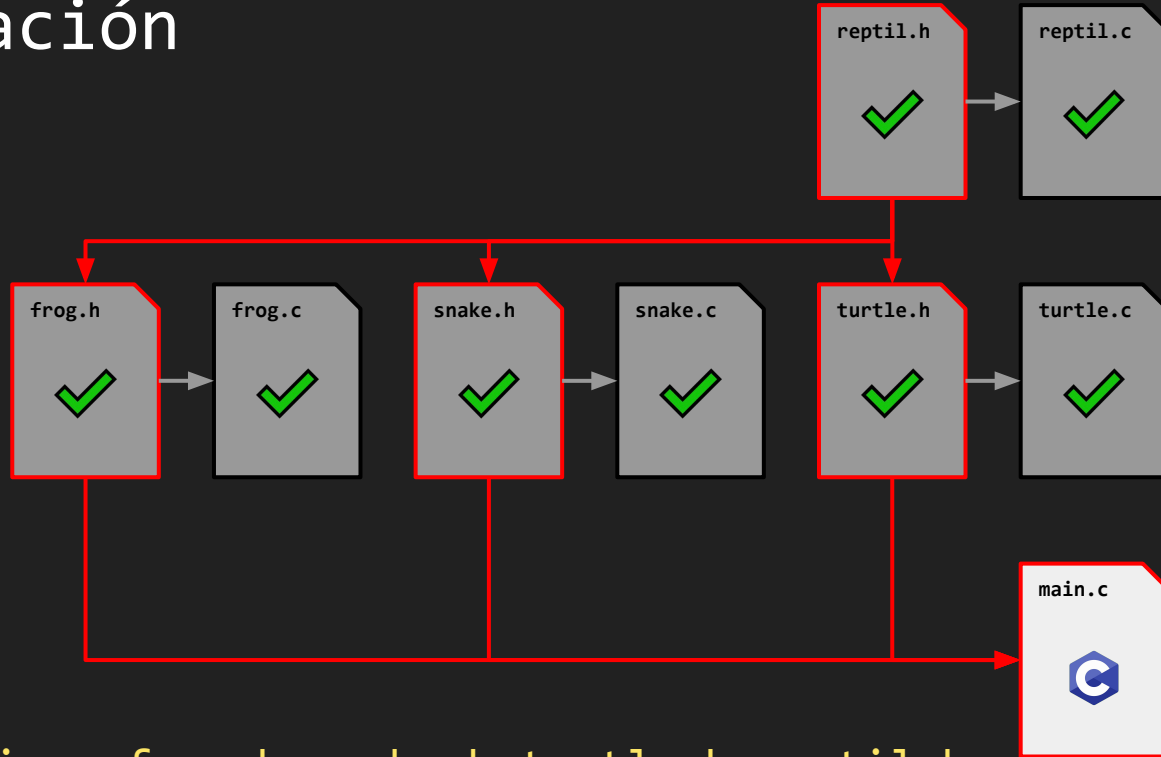
# Compilación



```
turtle.o: turtle.c turtle.h reptil.h
```

```
gcc turtle.c -c -o turtle.o
```

# Compilación

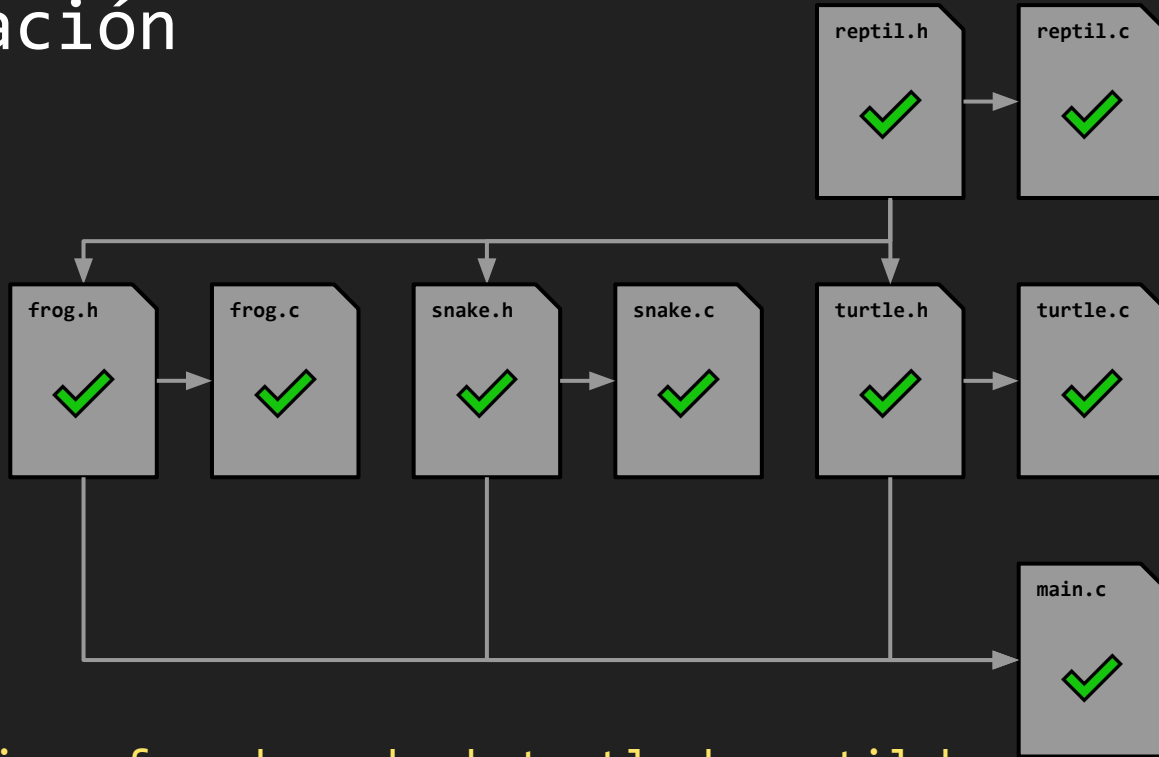


`main.o: main.c frog.h snake.h turtle.h reptil.h`

`gcc main.c -c -o main.o`



# Compilación



`main.o: main.c frog.h snake.h turtle.h reptil.h`

`gcc main.c -c -o main.o`

# Compilando el programa entero

```
main: main.o reptil.o frog.o snake.o turtle.o
```

```
gcc main.o reptil.o frog.o snake.o turtle.o -o main
```

```
$ make main
```

```
./main
```



# Makefile Final

## Makefile

```
main: main.o reptil.o frog.o snake.o turtle.o
    gcc main.o reptil.o frog.o snake.o turtle.o -o main

main.o: main.c frog.h snake.h turtle.h reptil.h
    gcc main.c -c -o main.o

reptil.o: reptil.c reptil.h
    gcc reptil.c -c -o reptil.o

frog.o: frog.c frog.h reptil.h
    gcc frog.c -c -o frog.o

snake.o: snake.c snake.h reptil.h
    gcc snake.c -c -o snake.o

turtle.o: turtle.c turtle.h reptil.h
    gcc turtle.c -c -o turtle.o
```

# Compilando el programa entero

`make` a secas invoca la primera regla del archivo

```
$ make
```

```
./main
```



# ¡Muchas Gracias!



With ♥ by @vichoeq & @KnowYourselves