

Intro a C

Módulos

With ❤ by @vichoeq & @KnowYourselfs

#include

¿Qué hace `#include`?

Se reemplaza `#include <modulo.h>` con todo el contenido de `modulo.h`

Importación de módulos

`#include <global.h>:`

Módulo global que se encuentra instalado en el computador que compila el programa.

`#include "local.h":`

Módulo local que se encuentra en la misma carpeta que el archivo que lo importa. Acepta rutas relativas.

¿Qué contiene un módulo?



`modulo.h`

1. **Declaración** de **funciones** externas
2. **Definición** de **tipos** públicos
3. etc.



`modulo.c`

1. **Definición** de **funciones** externas
2. **Definición** de **funciones** internas
3. etc.

Importación - C v/s Python



```
#include "modulo.h"
```

```
#include "modulo.c"
```



```
from modulo import funcion, ...
```

```
from modulo import *
```

```
#include "struct.h"
```



struct.h

1. Definición del struct
2. Declaración de funciones del struct



struct.c

1. Definición de funciones externas
2. Definición de funciones internas

`#include "struct.h"`



list.h

```
struct list
{
    int value;
    struct list* next;
};

typedef struct list List;

List* list_init();
List* list_append(List* list);
List* list_at_index(List* list, int index);
void list_destroy(List* list);
```



list.c

```
#include "list.h"

static void aux_func(){ ... }

List* list_init(){ ... }

List* list_append(List* list){ ... }

List* list_at_index(List* list, int index){ ... }

void list_destroy(List* list){ ... }
```


static



Una **función static** sólo puede ser usada en el **archivo.c** que fue definida.

Es ideal para **funciones** internas de un módulo.

list.c

```
#include "list.h"

static void aux_func(){ ... }

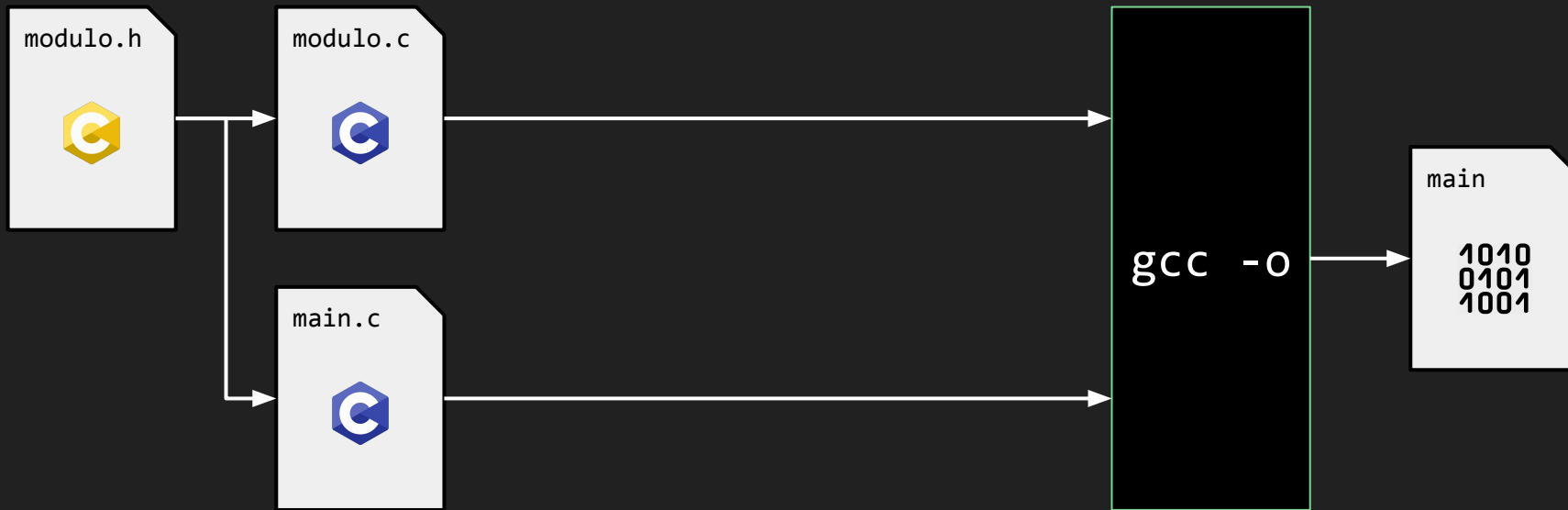
List* list_init(){ ... }

List* list_append(List* list){ ... }

List* list_at_index(List* list, int index){ ... }

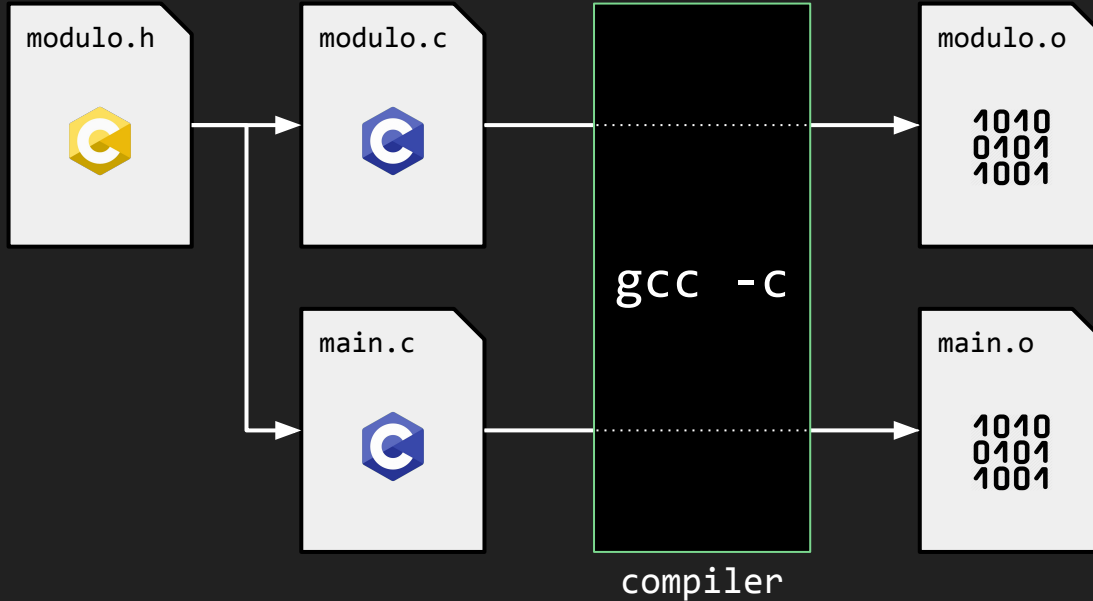
void list_destroy(List* list){ ... }
```

Compilación con módulos



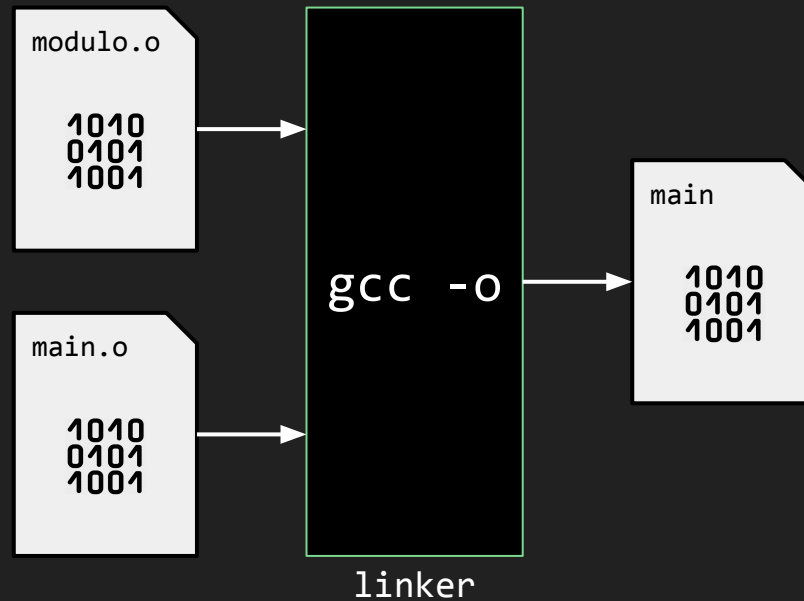
```
$ gcc modulo.c main.c -o main
```

Compilación con módulos



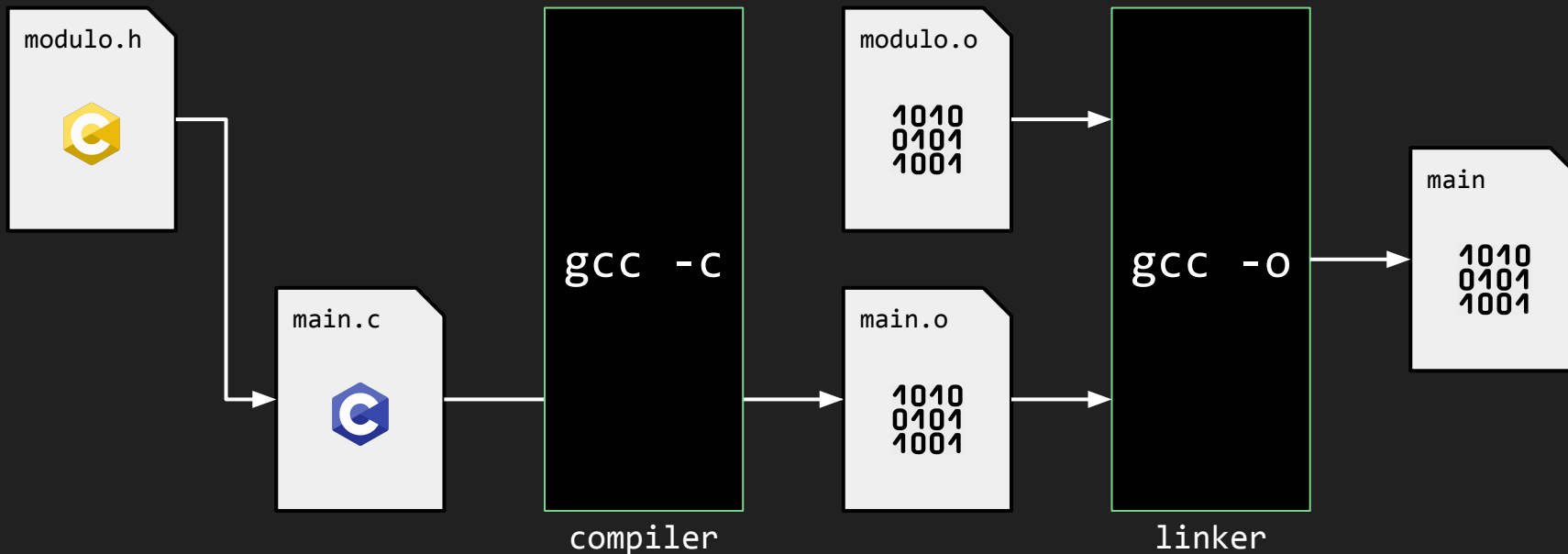
```
$ gcc modulo.c -c -o modulo.o  
$ gcc main.c -c -o main.o
```

Compilación con módulos



```
$ gcc modulo.o main.o -o main
```

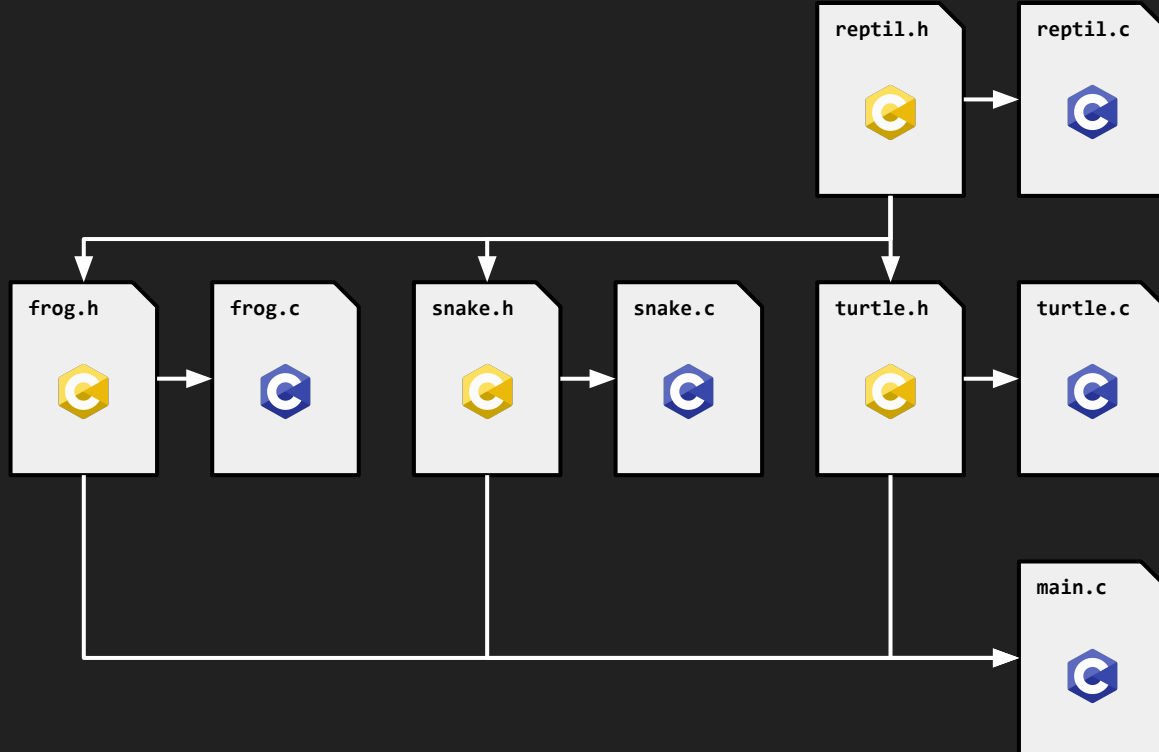
Compilación con módulos



```
$ gcc main.c -c -o main.o  
$ gcc modulo.o main.o -o main
```

What could go wrong?

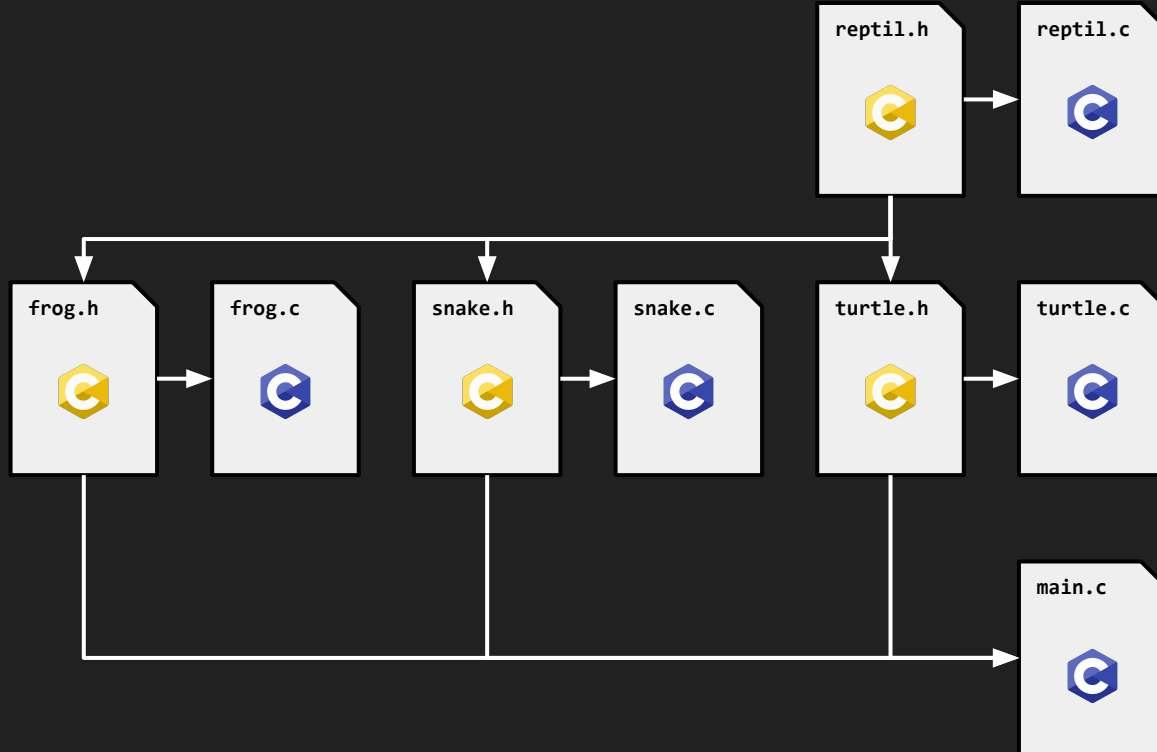
Cadenas de `#include`



¿Cuántas veces se incluye
`reptil.h`?

?

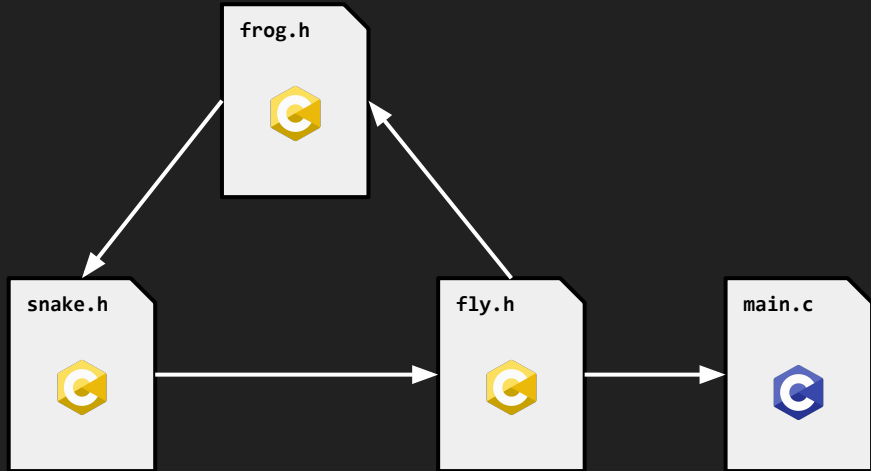
Cadenas de `#include`



¿Cuántas veces se incluye
`reptil.h`?

¡10! 🤯

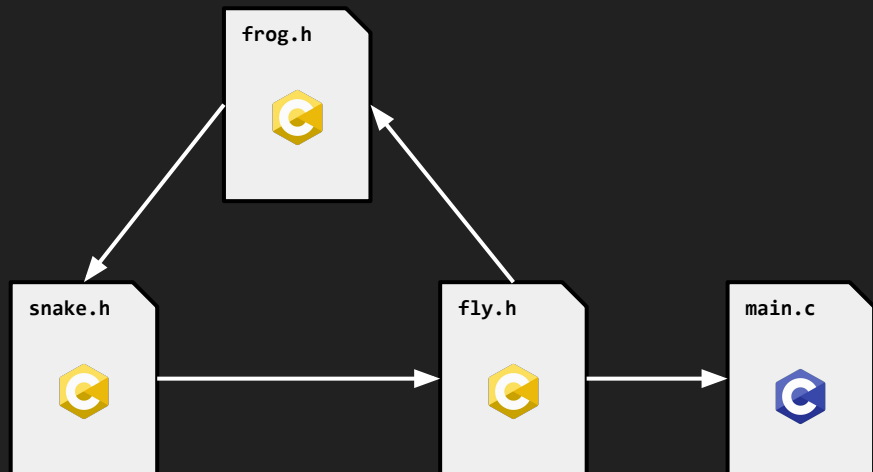
Ciclos de #include



¿Cuántas veces se incluye `snake.h`?

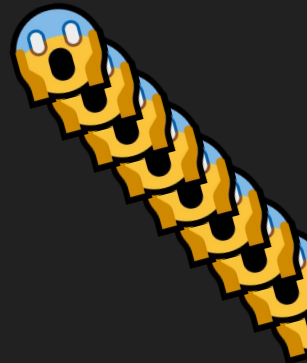
?

Ciclos de #include



¿Cuántas veces se incluye `snake.h`?

∞



#pragma once



reptil.h

```
#include <global.h>
#include "local.h"

typedef struct persona
{
    ...
} Persona;

void foo(int bar, char** boo);
```



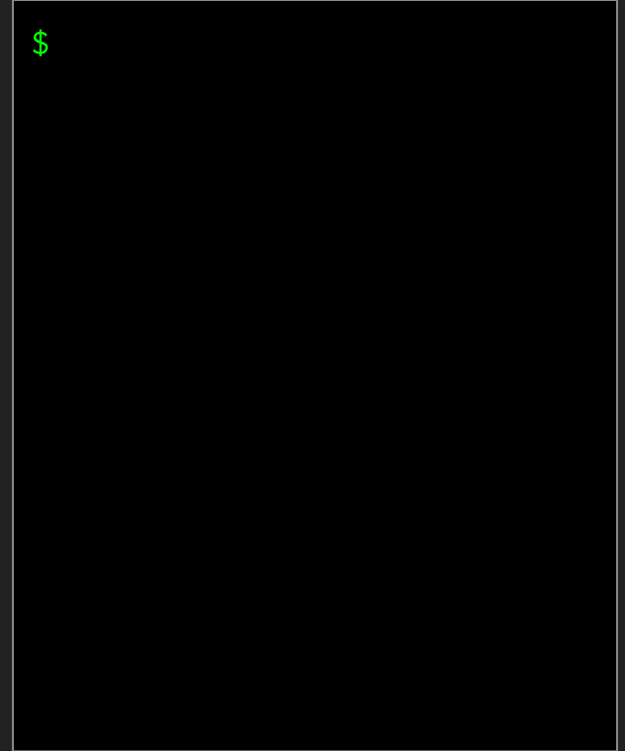
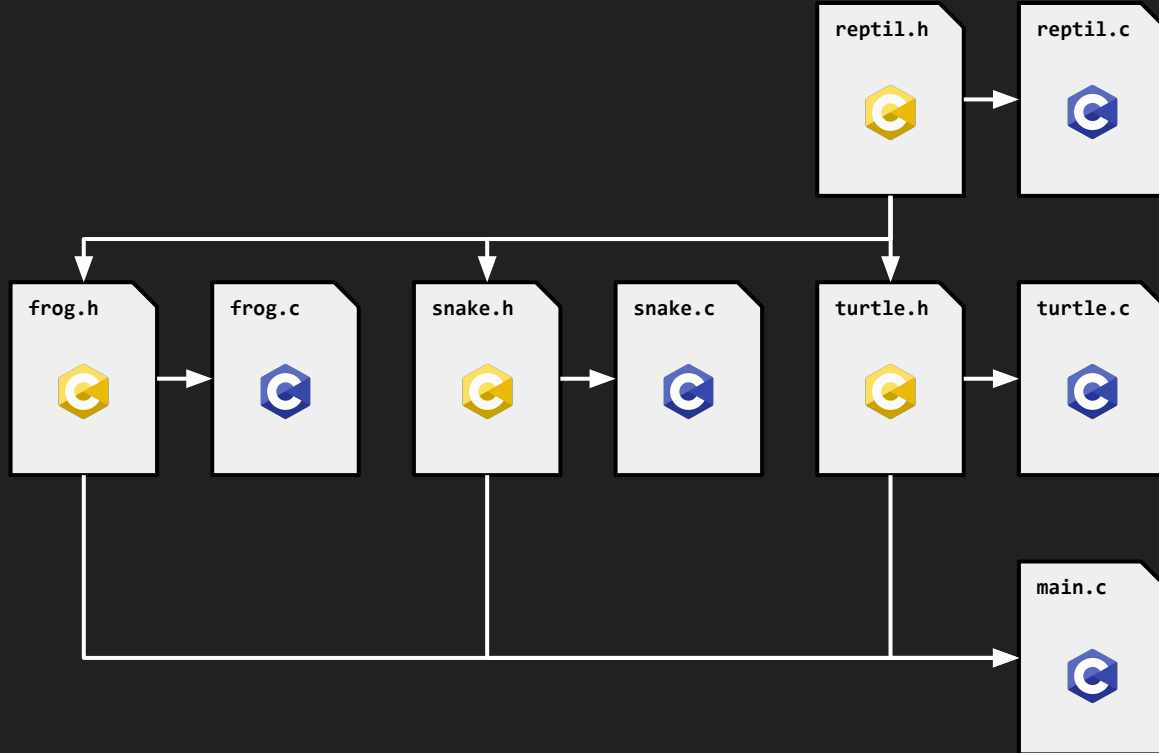
reptil.h

```
#pragma once
#include <global.h>
#include "local.h"

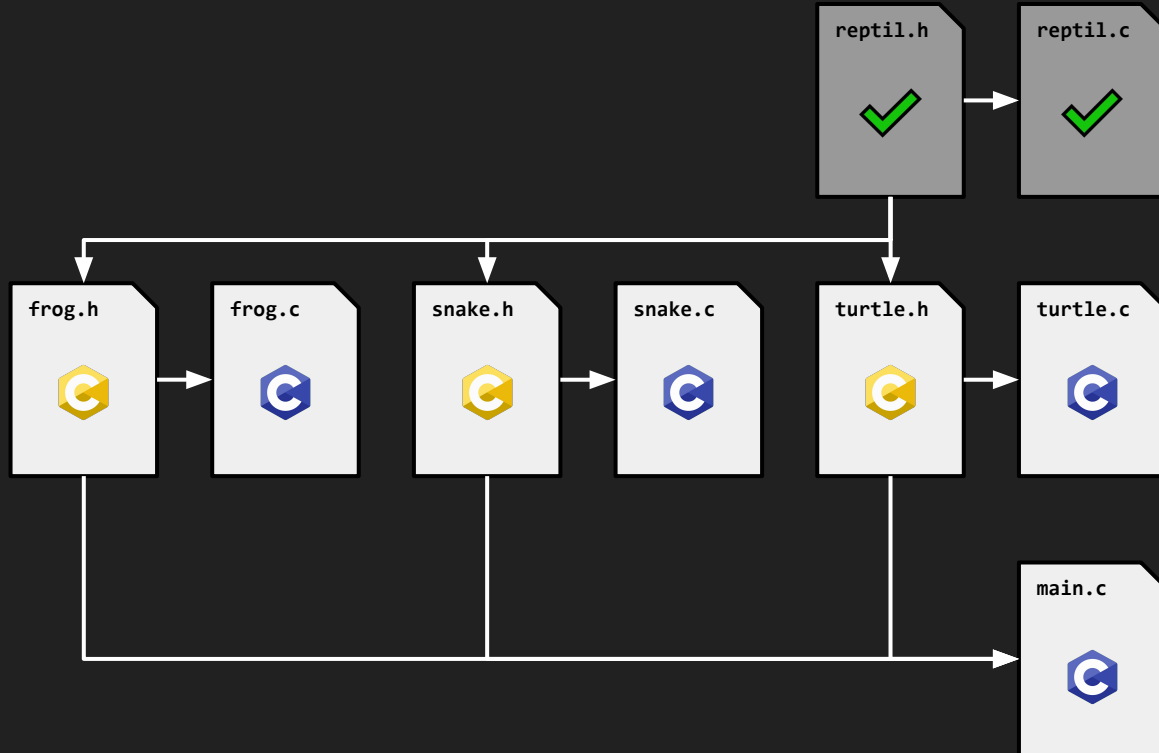
typedef struct persona
{
    ...
} Persona;

void foo(int bar, char** boo);
```

Compilando programa con módulos

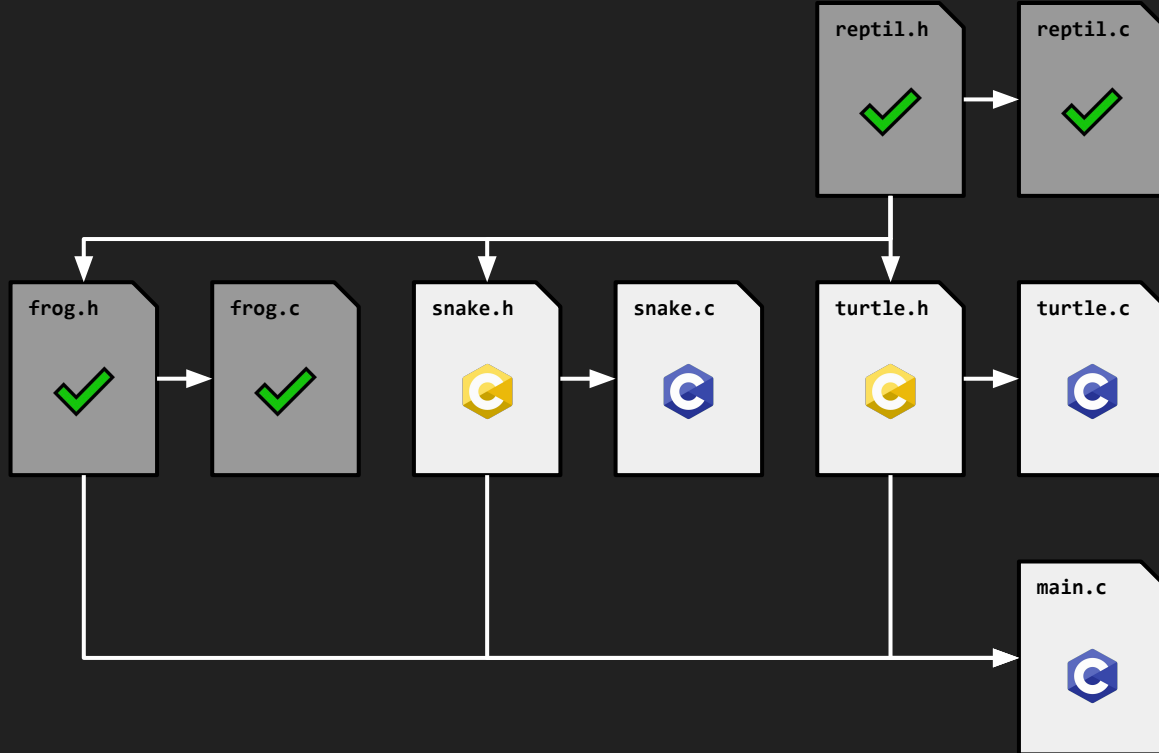


Compilando programa con módulos



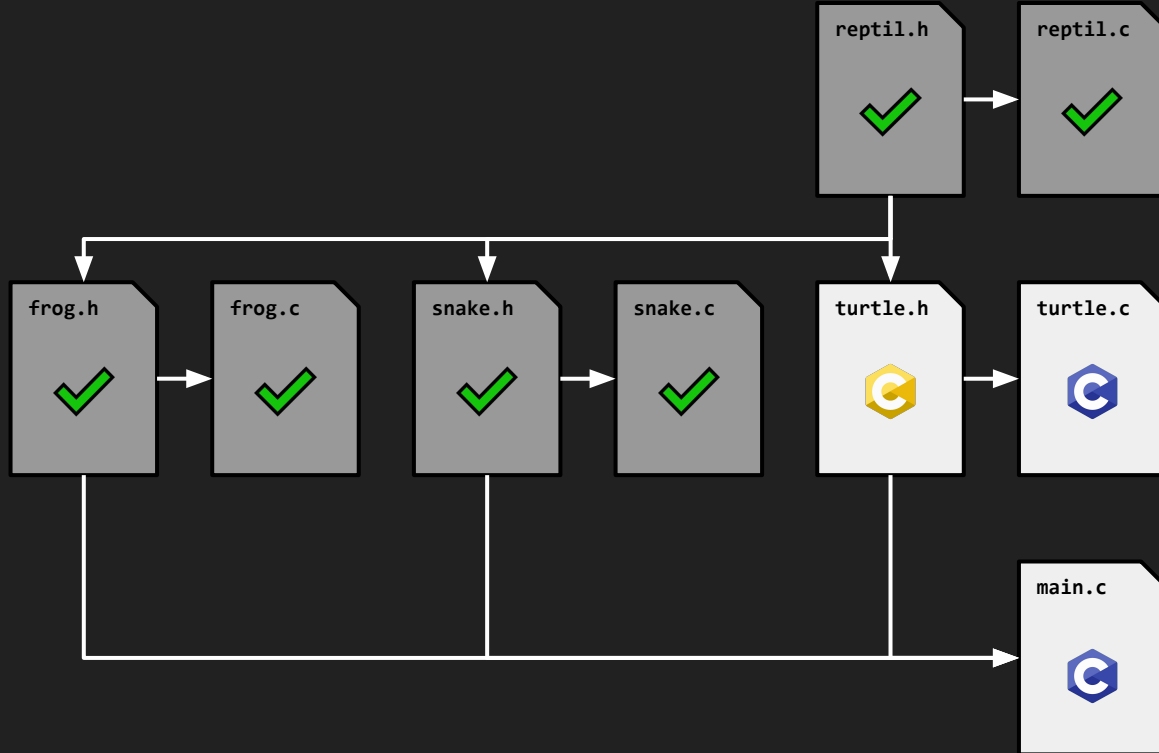
```
$ gcc reptil.c -c -o reptil.o
```

Compilando programa con módulos



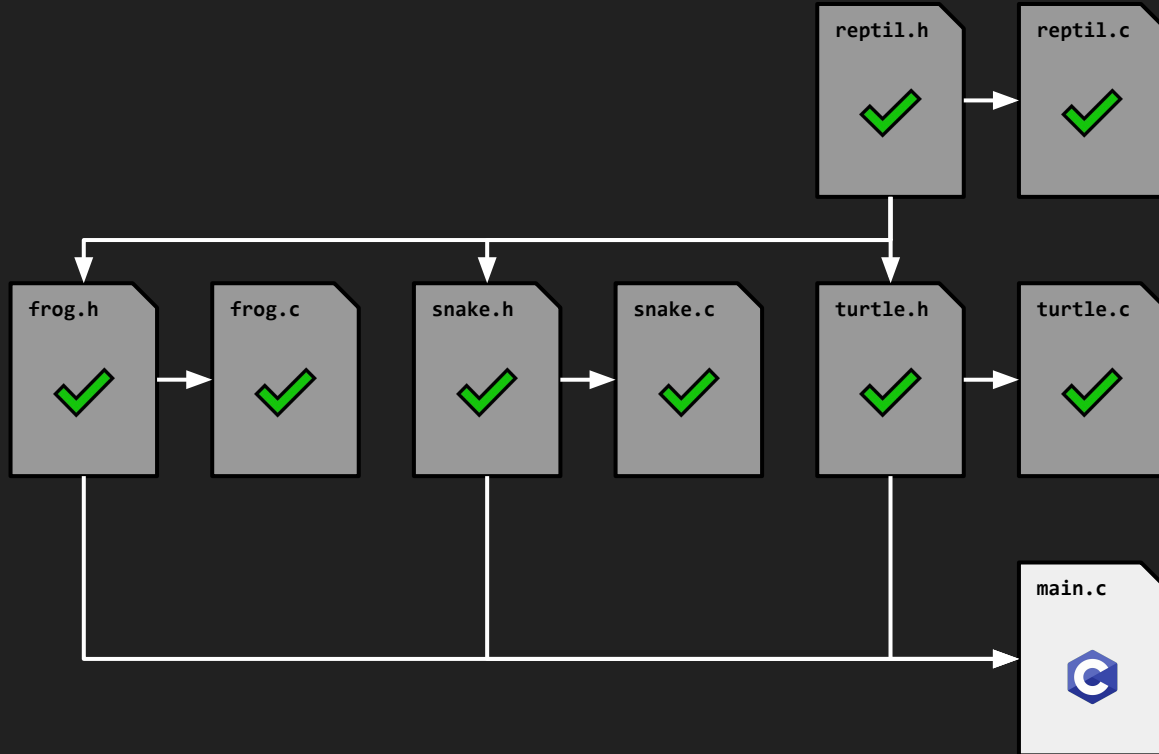
```
$ gcc reptil.c -c -o reptil.o  
$ gcc frog.c -c -o frog.o
```

Compilando programa con módulos



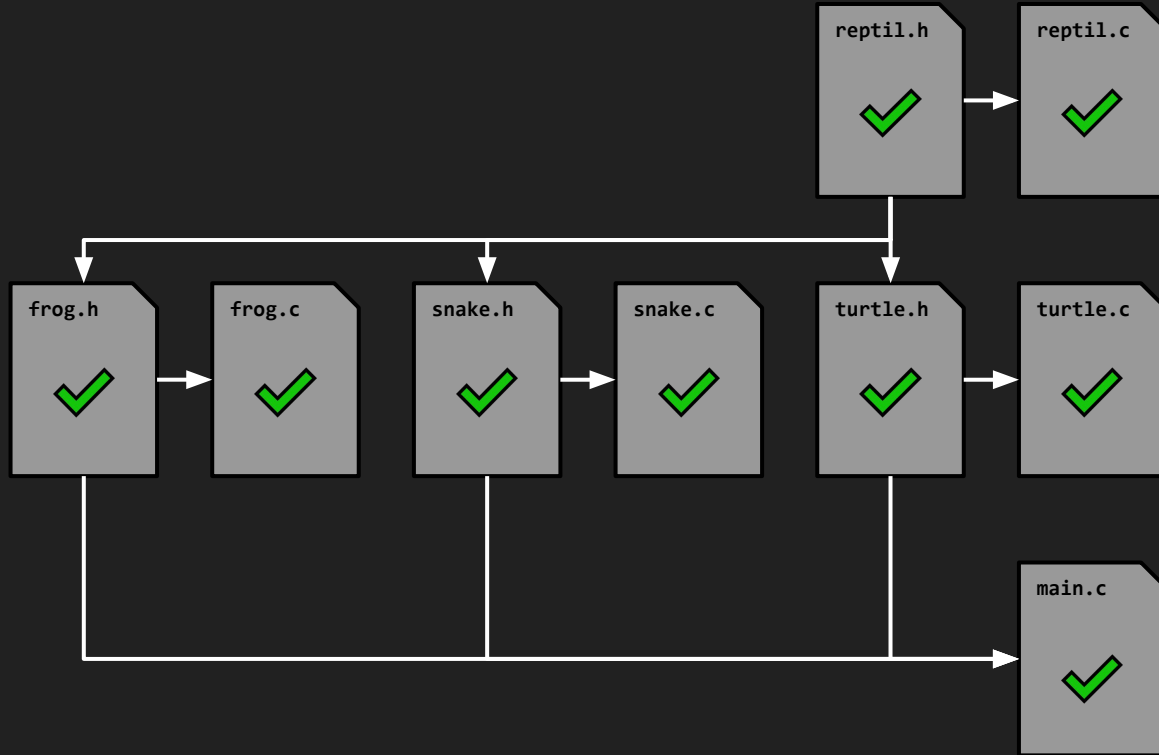
```
$ gcc reptil.c -c -o reptil.o  
$ gcc frog.c -c -o frog.o  
$ gcc snake.c -c -o snake.o
```

Compilando programa con módulos



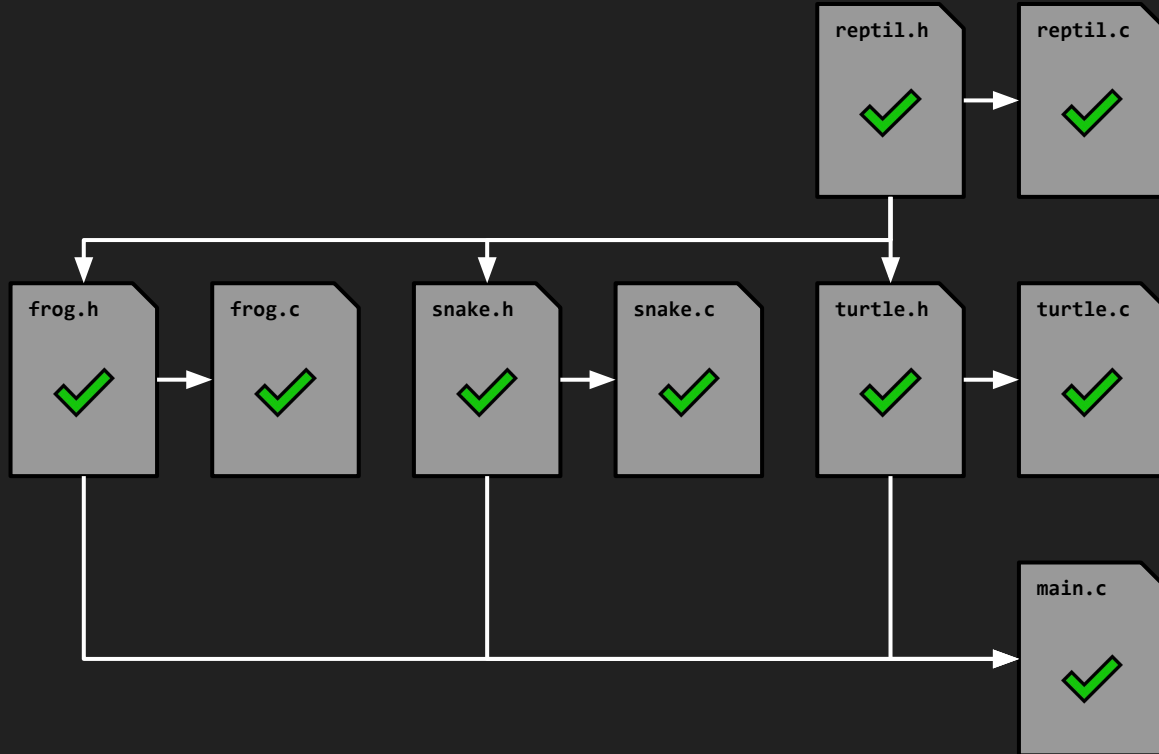
```
$ gcc reptil.c -c -o reptil.o  
$ gcc frog.c -c -o frog.o  
$ gcc snake.c -c -o snake.o  
$ gcc turtle.c -c -o turtle.o
```


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
```

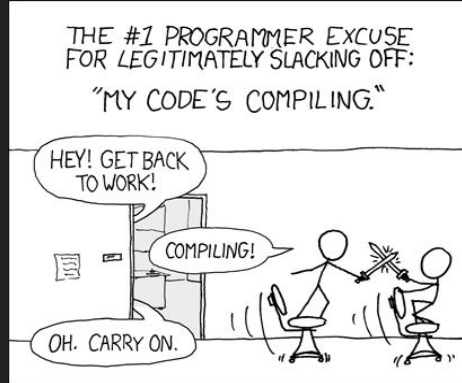
Compilando programa con módulos



```
$ 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
```



¡Muchas Gracias!



With ♥ by @vichoeq & @KnowYourselves