

## Introducción

### El curso

- Profesores
- Temario
- Material de clase
- Herramientas
- Flujo de trabajo
- Evaluación

### Historia

- Inicios
- Influencias

### ¿Por qué C?

### ¿Para qué C?

- Proyectos en C

### Sumer Of Code

- GSOC
- Outreachy

# Introducción

## Tema 1

## Introducción

### El curso

Profesores  
Temario  
Material de clase  
Herramientas  
Flujo de trabajo  
Evaluación

### Historia

Inicios  
Influencias

### ¿Por qué C?

### ¿Para qué C?

Proyectos en C

### Sumer Of Code

GSOC  
Outreachy

- 1 El curso
  - Profesores
  - Temario
  - Material de clase
  - Herramientas
  - Flujo de trabajo
  - Evaluación
- 2 Historia de C
  - Inicios
  - Influencias
- 3 ¿Por qué C?
- 4 ¿Para qué C?
  - Proyectos en C
- 5 Sumer Of Code
  - Google Summer Of Code
  - Outreachy

## Introducción

### El curso

#### Profesores

Temario

Material de clase

Herramientas

Flujo de trabajo

Evaluación

### Historia

Inicios

Influencias

### ¿Por qué C?

### ¿Para qué C?

Proyectos en C

### Sumer Of Code

GSOC

Outreachy



Pablo Neira Ayuso



Carlos Falgueras García

## Introducción

### El curso

Profesores

**Temario**

Material de clase

Herramientas

Flujo de trabajo

Evaluación

### Historia

Inicios

Influencias

### ¿Por qué C?

### ¿Para qué C?

Proyectos en C

### Summer Of Code

GSOC

Outreachy

- Variables, tipos y punteros
- Herramientas y espacio de trabajo
- Arrays y estructuras
- C modular
- Memoria dinámica
- Objetos
- Argumentos del main
- Listas encadenadas
- Entrada y salida
- Punteros a funciones
- Herramientas de depuración
- Interfaces gráficas con GTK
- Sockets

## Introducción

### El curso

Profesores

Temario

**Material de clase**

Herramientas

Flujo de trabajo

Evaluación

### Historia

Inicios

Influencias

¿Por qué C?

¿Para qué C?

Proyectos en C

Sumer Of Code

GSOC

Outreachy

## Wiki

<http://1984.lsi.us.es/wiki-c>

## Lista de correo

[https:](https://listas.us.es/mailman/listinfo/programacion-c)

[//listas.us.es/mailman/listinfo/programacion-c](https://listas.us.es/mailman/listinfo/programacion-c)

## Introducción

### El curso

Profesores

Temario

Material de  
clase

### Herramientas

Flujo de  
trabajo

Evaluación

### Historia

Inicios

Influencias

### ¿Por qué C?

### ¿Para qué C?

Proyectos en  
C

### Sumer Of Code

GSOC

Outreachy

- Linux y terminal
- Editor de texto (*geany*)
- Compilador GCC
- *Make* para automatizar la compilación
- Repositorio de código (*git* y *GitHub*)

# Flujo de trabajo

## Introducción

### El curso

- Profesores
- Temario
- Material de clase
- Herramientas
- Flujo de trabajo**
- Evaluación

### Historia

- Inicios
- Influencias

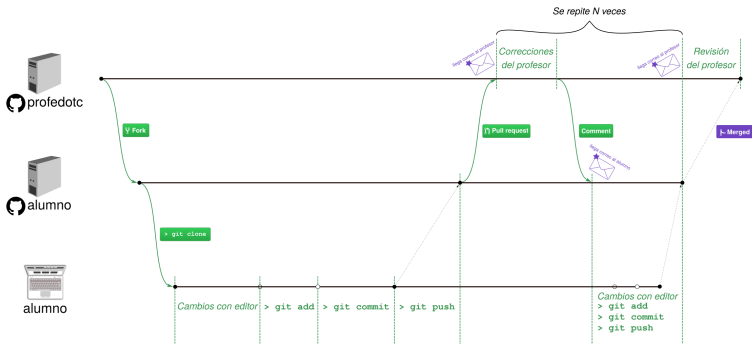
### ¿Por qué C?

### ¿Para qué C?

- Proyectos en C

### Sumer Of Code

- GSOC
- Outreachy



## Introducción

### El curso

Profesores

Temario

Material de clase

Herramientas

Flujo de trabajo

**Evaluación**

### Historia

Inicios

Influencias

¿Por qué C?

¿Para qué C?

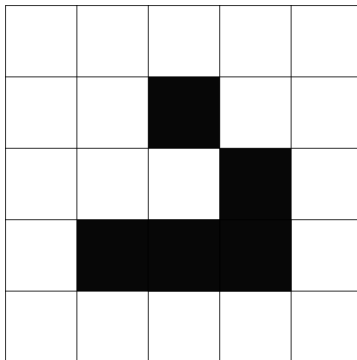
Proyectos en C

Summer Of Code

GSOC

Outreachy

## Juego de la vida de Conway



[https://en.wikipedia.org/wiki/Conway's\\_Game\\_of\\_Life](https://en.wikipedia.org/wiki/Conway's_Game_of_Life)





- Ken Thompson
- Dennis Ritchie
- Brian Kernighan
- Bell Labs (de AT&T)
- Ensamblador y B insuficientes → diseñan C
- C fue desarrollado por Dennis Ritchie entre 1969 y 1973
- Unix reescrito en C (1973)
- En 1973 *Brian Kernighan* y *Dennis Ritchie* publican **The C Programming Language (K&R)**, que por muchos años sirvió como especificación informal del lenguaje.
- Posteriormente se añaden más funcionales a C y se estandariza.



- **Ken Thompson**
- **Dennis Ritchie**
- **Brian Kernighan**
- Bell Labs (de AT&T)
- Ensamblador y B insuficientes → diseñan C
- C fue desarrollado por **Dennis Ritchie** entre 1969 y 1973
- **Unix** reescrito en C (1973)
- En 1973 *Brian Kernighan* y *Dennis Ritchie* publican **The C Programming Language (K&R)**, que por muchos años sirvió como especificación informal del lenguaje.
- Posteriormente se añaden más funcionales a C y se estandariza.



- **Ken Thompson**
- **Dennis Ritchie**
- **Brian Kernighan**
- **Bell Labs (de AT&T)**
  - Ensamblador y B insuficientes → diseñan C
  - C fue desarrollado por **Dennis Ritchie** entre 1969 y 1973
  - **Unix** reescrito en C (1973)
  - En 1973 *Brian Kernighan* y *Dennis Ritchie* publican **The C Programming Language (K&R)**, que por muchos años sirvió como especificación informal del lenguaje.
  - Posteriormente se añaden más funcionales a C y se estandariza.



- **Ken Thompson**
- **Dennis Ritchie**
- **Brian Kernighan**
- **Bell Labs (de AT&T)**
- **Ensamblador y B insuficientes → diseñan C**
- C fue desarrollado por **Dennis Ritchie** entre 1969 y 1973
- **Unix** reescrito en C (1973)
- En 1973 *Brian Kernighan* y *Dennis Ritchie* publican **The C Programming Language (K&R)**, que por muchos años sirvió como especificación informal del lenguaje.
- Posteriormente se añaden más funcionales a C y se estandariza.



- **Ken Thompson**
- **Dennis Ritchie**
- **Brian Kernighan**
- **Bell Labs (de AT&T)**
- **Ensamblador y B insuficientes → diseñan C**
- **C fue desarrollado por Dennis Ritchie entre 1969 y 1973**
- **Unix reescrito en C (1973)**
- En 1973 *Brian Kernighan y Dennis Ritchie* publican **The C Programming Language (K&R)**, que por muchos años sirvió como especificación informal del lenguaje.
- Posteriormente se añaden más funcionales a C y se estandariza.



- Ken Thompson
- Dennis Ritchie
- Brian Kernighan
- Bell Labs (de AT&T)
- Ensamblador y B insuficientes → diseñan C
- C fue desarrollado por **Dennis Ritchie** entre 1969 y **1973**
- **Unix** reescrito en C (1973)
  - En 1973 *Brian Kernighan* y *Dennis Ritchie* publican **The C Programming Language (K&R)**, que por muchos años sirvió como especificación informal del lenguaje.
- Posteriormente se añaden más funcionales a C y se estandariza.



- Ken Thompson
- Dennis Ritchie
- Brian Kernighan
- Bell Labs (de AT&T)
- Ensamblador y B insuficientes → diseñan C
- C fue desarrollado por **Dennis Ritchie** entre 1969 y 1973
- **Unix** reescrito en C (1973)
- En 1973 *Brian Kernighan* y *Dennis Ritchie* publican **The C Programming Language (K&R)**, que por muchos años sirvió como especificación informal del lenguaje.
- Posteriormente se añaden más funcionales a C y se estandariza.



## Introducción

### El curso

Profesores  
Temario  
Material de clase  
Herramientas  
Flujo de trabajo  
Evaluación

### Historia

Inicios  
Influencias

### ¿Por qué C?

### ¿Para qué C?

Proyectos en C

### Summer Of Code

GSOC  
Outreachy

- Ken Thompson
- Dennis Ritchie
- Brian Kernighan
- Bell Labs (de AT&T)
- Ensamblador y B insuficientes → diseñan C
- C fue desarrollado por **Dennis Ritchie** entre 1969 y 1973
- **Unix** reescrito en C (1973)
- En 1973 *Brian Kernighan* y *Dennis Ritchie* publican **The C Programming Language (K&R)**, que por muchos años sirvió como especificación informal del lenguaje.
- Posteriormente se añaden más funcionales a C y se estandariza.



## Introducción

### El curso

- Profesores
- Temario
- Material de clase
- Herramientas
- Flujo de trabajo
- Evaluación

### Historia

- Inicios
- Influencias**

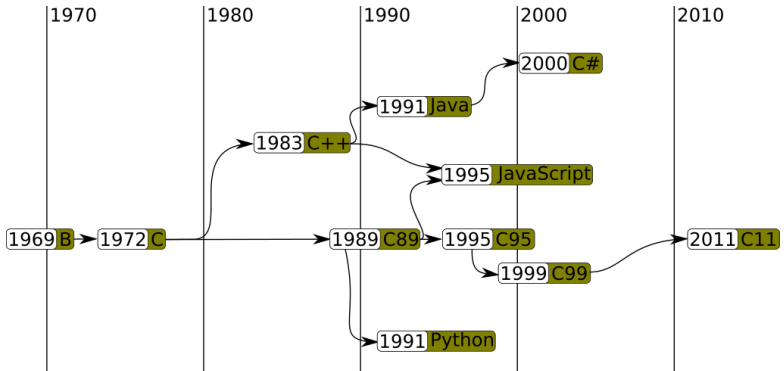
### ¿Por qué C?

### ¿Para qué C?

- Proyectos en C

### Sumer Of Code

- GSOC
- Outreachy



# ¿Por qué C?

## Introducción

### El curso

Profesores  
Temario  
Material de clase  
Herramientas  
Flujo de trabajo  
Evaluación

### Historia

Inicios  
Influencias

### ¿Por qué C?

### ¿Para qué C?

Proyectos en C

### Sumer Of Code

GSOC  
Outreachy

- Simpleza
- Características de bajo nivel
- Madurez
- Eficiencia
- Portabilidad
- Numerosas bibliotecas y herramientas

# ¿Por qué C?

## Introducción

### El curso

Profesores  
Temario  
Material de clase  
Herramientas  
Flujo de trabajo  
Evaluación

### Historia

Inicios  
Influencias

### ¿Por qué C?

### ¿Para qué C?

Proyectos en C

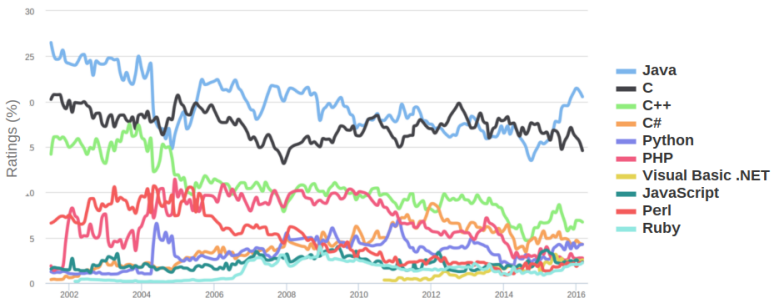
### Summer of Code

GSOC  
Outreachy

## • Popularidad

### TIOBE Programming Community Index

Source: [www.tiobe.com](http://www.tiobe.com)



# ¿Para qué C?

## Introducción

### El curso

Profesores  
Temario  
Material de clase  
Herramientas  
Flujo de trabajo  
Evaluación

### Historia

Inicios  
Influencias

### ¿Por qué C?

### ¿Para qué C?

Proyectos en C

### Sumer Of Code

GSOC  
Outreachy

- **Ciencia:**
  - Simulaciones
  - Operaciones con grandes cantidades de datos
- **Sistemas Empotrados:**
  - Sistemas Operativos en tiempo real
  - Electrodomésticos, ascensores, automovilismo ...
- **Robótica**
  - Drones
  - Robots humanoides
  - Coches autónomos
- **Medicina**
  - Prótesis robóticas
  - Equipamiento médico
- **Sistemas Operativos**

## Introducción

### El curso

Profesores

Temario

Material de clase

Herramientas

Flujo de trabajo

Evaluación

### Historia

Inicios

Influencias

### ¿Por qué C?

### ¿Para qué C?

**Proyectos en C**

Sumer Of Code

GSOC

Outreachy

- Unix, GNU/Linux, kernel de MacOS y kernel de Windows
- Firefox y muchos otros exploradores (gumbo)
- Apache
- Gnome (GTK)
- Rover Curiosity (2.5 millones de lineas)

# Google Summer Of Code

## Introducción

### El curso

Profesores  
Temario  
Material de clase  
Herramientas  
Flujo de trabajo  
Evaluación

### Historia

Inicios  
Influencias

### ¿Por qué C?

### ¿Para qué C?

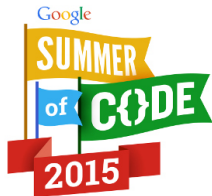
Proyectos en C

### Summer Of Code

**GSOC**  
Outreachy

- Beca de Google para estudiantes
- Trabajas **3 meses** en un proyecto de **software libre**
- Experiencia
- Dinero: 5500\$

<https://summerofcode.withgoogle.com/>



## Introducción

### El curso

Profesores

Temario

Material de clase

Herramientas

Flujo de trabajo

Evaluación

### Historia

Inicios

Influencias

### ¿Por qué C?

### ¿Para qué C?

Proyectos en C

### Summer Of Code

Code

GSOC

Outreachy

# OUTREACHY

- Beca de Gnome para:
  - mujeres
  - grupos discriminados o con poca representación en el mundo tecnológico
  - **Que no hayan participado antes ni en Outreachy ni en GSOC**
- Trabajas **3 meses** en un proyecto de **software libre**
- Experiencia
- Dinero: 5500\$

Workspace

Linux

Consola

Instalando  
herramientas

Hola Mundo

# Entorno de trabajo

## Tema 2



## Workspace

Linux

Consola

Instalando  
herramientas

Hola Mundo

**6** Linux

**7** Consola

**8** Instalando herramientas

**9** Hola Mundo

GNU/Linux es el sistema operativo que vamos a utilizar durante el curso.

- Ofrece muchísimas facilidades al programador
- Software libre y gratuito
- Repositorios con infinidad de herramientas a nuestra disposición



# Consola

Workspace

## CTRL + ALT + T

Linux

Consola

Instalando  
herramientas

Hola Mundo

```

s/udata.c | /udata.h
2 EXPORT_SYMBOL(nftnl_udata_attr_value);
21
22 struct nftnl_udata *nftnl_udata_attr_next(const struct nftnl_udata *attr)
23 {
24     return (struct nftnl_udata *)attr->value[attr->len];
25 }
26 EXPORT_SYMBOL(nftnl_udata_attr_next);
27
28 int nftnl_udata_parse(const struct nftnl_udata_buf *buf, nftnl_udata_cb_t cb,
29                     void *data)
30 {
31     int ret = 0;
32     const struct nftnl_udata *attr;
33
34     nftnl_udata_for_each(buf, attr) {
35         ret = cb(attr, data);
36         if (ret <= 0)
37             return ret;
38     }
39
40     return ret;
41 }
42
134 EXPORT_SYMBOL(nftnl_udata_parse);=
src/udata.c                                c 100% 134:1  NORMAL | #0 -# 0 | squash | include/udata.h | nftnl_udata | cpp | utf-8[unix] | 60% | 24:1
buffers

ls
arch          drivers  Kbuild   mm          patches     System.map
block         firmware Kconfig  modules.builtin  README      tools
certs         fs       kernel   modules.order  REPORTING-0065  user
COPYING      include  kva-conf-3.12  Module.symvers  samples      virt
CREDITS      init     lib       net           scripts      valinux
crypto       instk.sh MAINTAINERS  nft-105995-gbccc90f  security     valinux.o
Documentation ipc       kexecfile  RT-907429c     sound

git log -1
commit 3816c7947bd0ebd3a1c34097a5df675eeaf190
Author: Carlos Falgaras Garcia <carlosgf@riseup.net>
Date:   Mon Dec 14 12:38:46 2015 +0100

netfilter: nf_tables_api: Add new attributes into nft_set to store user data.

User data is stored at after 'nft_set_ops' private data into 'data[]'
flexible array. The field 'udata' points to user data and 'udlen' stores
its length.

Add new flag NFTA_SET_USERDATA.

```

Comandos básicos:

`ls`: Lista directorios

`cd <dir>`: Cambia a directorio

`mkdir <dir>`: Crea directorio

`touch <file>`: Crea archivo vacío\*

`rm <file>`: Borra archivo

`rm -r <dir>`: Borra directorio y lo que hay dentro

# Instalando herramientas

Workspace

Linux

Consola

Instalando  
herramientas

Hola Mundo

Comandos para instalar:

`sudo <comando>`: Ejecuta un comando con permisos de administrador

`apt-get install <programa>`: Instala un programa del repositorio

Programas a instalar (`sudo apt-get install <programa>`):

- **gcc**: Compilador
- **make**: Automatización de tareas
- **git**: Gestor de versiones
- **geany**: Editor de texto gráfico

# Hola Mundo

Workspace

Abrir *geany* y guardar el siguiente archivo:

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Hola mundo!\n");
6
7     return 0;
8 }
```

Compilación y ejecución:

- Hacer *cd* hasta el directorio dónde se encuentra *helloworld.c*
- `gcc <mi_prog.c> -o <mi_exe>`
- `./mi_exe`

## Variables

¿Qué es?

Tipos  
básicos

Tipos de  
tamaño fijo

# Variables y tipos

## Tema 3

## Variables

¿Qué es?

Tipos  
básicos

Tipos de  
tamaño fijo

**10** ¿Qué es una variable?

**11** Tipos básicos

**12** Tipos de tamaño fijo



# ¿Qué es una variable?

## Variables

### ¿Qué es?

### Tipos básicos

### Tipos de tamaño fijo

- Variables como **zona de memoria** reservada de **tamaño específico**
- Los tipos:
  - Definen el tamaño
  - Dan una idea del uso que se le van a dar a los datos guardados

# ¿Qué es una variable?

## VARIABLES

### ¿Qué es?

### Tipos básicos

### Tipos de tamaño fijo

`int` contador;

reservo 4 bytes

voy a contar  
numeros enteros (grandes)

# ¿Qué es una variable?

## Variables

¿Qué es?

Tipos  
básicos

Tipos de  
tamaño fijo

`char c;`

reservo 1 byte

voy a guardar  
un caracter

# ¿Qué es una variable?

## Variables

### ¿Qué es?

### Tipos básicos

### Tipos de tamaño fijo

`char` contador;

reservo 1 byte

voy a contar  
numeros enteros (pequeños)

## Tipos:

- char ("%c")
- int ("%i") ó ("%d")
- float ("%f")
- double ("%f")
- bool

## Modificadores:

- signed ("%hh□")
- unsigned ("%u")
- short ("%h□")
- long ("%l□")
- long long ("%ll□")

Más info sobre formato de printf: <http://www.cplusplus.com/reference/cstdio/printf>

# Tipos de tamaño fijo

## Variables

¿Qué es?

Tipos básicos

Tipos de tamaño fijo

```
#include <stdint.h>
```

```
[u]int_<size>_t
```

- int8\_t
- int16\_t
- int32\_t
- int64\_t
- uint8\_t
- uint16\_t
- uint32\_t
- uint64\_t

## Arrays

Descripción

Ejemplo

Cadenas

Array multi-  
dimensional

# Arrays y tipos

## Tema 4

## Arrays

Descripción

Ejemplo

Cadenas

Array multi-  
dimensional

**13** Descripción

**14** Ejemplo

**15** Cadenas

**16** Array multidimensional



## Arrays

```
int array[5] = {1, 2, 3, 4, 5};
```

- Reserva de memoria **continua** de forma **estática**
- Usos:
  - Vector de elementos
  - Matrices (multidimensionales)
  - Cadenas de texto
  - Espacio de memoria (buffer)

# Ejemplo

## Arrays

Descripción

Ejemplo

Cadenas

Array multi-  
dimensional

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int i;
6     int vector1[10];
7     int vector2[] = {9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
8
9     for (i = 0; i < 10; i++)
10         vector1[i] = vector2[i];
11
12     for (i = 0; i < 10; i++)
13         printf("%d ", vector1[i]);
14
15     return 0;
16 }
```

# Cadenas

## Arrays

### Descripción

### Ejemplo

### Cadenas

### Array multi-dimensional

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int i;
6     char hola[] = {'h', 'o', 'l', 'a', '\0'};
7     char mundo[] = "mundo";
8
9     printf("%s %s\n", hola, mundo);
10
11     for (i = 0; i < 4; i++)
12         printf("%c ", hola[i]);
13
14     for (i = 0; i < 5; i++)
15         printf("%c ", mundo[i]);
16
17     return 0;
18 }
```

# Array multidimensional

## Arrays

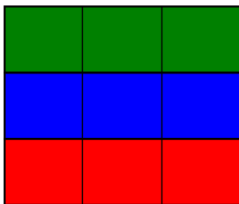
Descripción

Ejemplo

Cadenas

Array multi-  
dimensional

```
int array[3][3] = {{11, 12, 13}, {21, 22, 23}, {31, 32, 33}};
```



GOL

[Introducción](#)

[Ejemplo](#)

[Enlaces de  
interés](#)

# Juego de la vida

## Tema 5

## GOL

Introducción

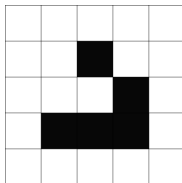
Ejemplo

Enlaces de  
interés

**17** Introducción

**18** Ejemplo

**19** Enlaces de interés



- juego de 0 jugadores
- Rejilla de células cuadradas como universo bidimensional ortogonal (infinito o no)
- Cada célula tiene dos estados (muerta o viva) e interactúa con sus 8 vecinas según unas reglas
- La regla más común es:
  - **Nacimiento:** Una célula muerta con exactamente 3 vecinas vivas estará viva en la siguiente iteración
  - **Supervivencia:** Una célula viva con 2 o 3 vecinas vivas seguirá viva en la siguiente iteración, de lo contrario morirá.

# Ejemplo

GOL

[Introducción](#)

[Ejemplo](#)

[Enlaces de interés](#)

	1	2	3	2	1	
	1	1	2	1	1	
	1	2	3	2	1	



# Ejemplo

GOL

[Introducción](#)

[Ejemplo](#)

[Enlaces de interés](#)

		1	1	1		
		2	1	2		
		3	2	3		
		2	1	2		
		1	1	1		

# Ejemplo

## GOL

[Introducción](#)

**[Ejemplo](#)**

[Enlaces de interés](#)

	1	2	3	2	1	
	1	1	2	1	1	
	1	2	3	2	1	

## GOL

Introducción

Ejemplo

Enlaces de  
interés

- Más información:  
[es.wikipedia.org/wiki/Juego\\_de\\_la\\_vida](https://es.wikipedia.org/wiki/Juego_de_la_vida)
- Simulador (muy bueno):  
[golly.sourceforge.net/](http://golly.sourceforge.net/)
- Simulador web:  
[pmav.eu/stuff/javascript-game-of-life-v3.1.1/](http://pmav.eu/stuff/javascript-game-of-life-v3.1.1/)

## Punteros

¿Qué son?

Ejemplo

Sintaxis

Arrays y  
punteros

Ejemplo

Recorriendo  
arrays

Jugando con  
Punteros

# Punteros

## Tema 6

## Punteros

¿Qué son?

Ejemplo

Sintaxis

Arrays y  
punteros

Ejemplo

Recorriendo  
arrays

Jugando con  
Punteros

**20** ¿Qué es un puntero?

- Ejemplo
- Sintaxis

**21** Arrays y punteros

■ Ejemplo

- Formas de recorrer un array

**22** Jugando con Punteros

# ¿Qué es un puntero?

## Punteros

¿Qué son?

Ejemplo  
Sintaxis

Arrays y  
punteros

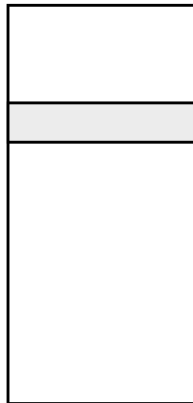
Ejemplo  
Recorriendo  
arrays

Jugando con  
Punteros

- Son **variables normales y corrientes**
- Pensadas para guardar una **dirección de memoria**
- El tipo del puntero hace referencia al tipo de dato **al que apunta**

\*mi\_puntero

## Memoria



# ¿Qué es un puntero?

## Punteros

¿Qué son?

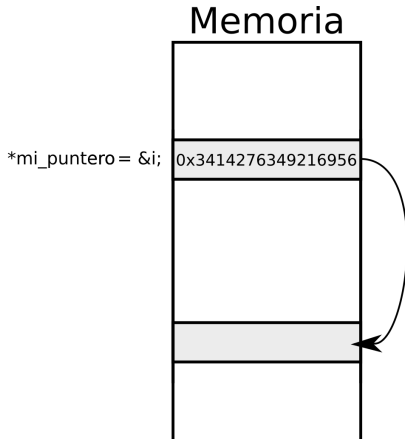
Ejemplo  
Sintaxis

Arrays y  
punteros

Ejemplo  
Recorriendo  
arrays

Jugando con  
Punteros

- Son **variables normales** y corrientes
- Pensadas para guardar una **dirección de memoria**
- El tipo del puntero hace referencia al tipo de dato al que apunta



# ¿Qué es un puntero?

## Punteros

### ¿Qué son?

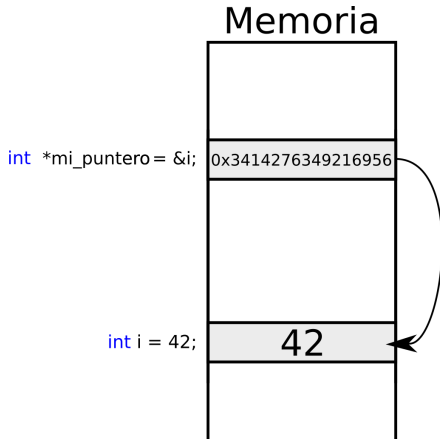
Ejemplo  
Sintaxis

### Arrays y punteros

Ejemplo  
Recorriendo arrays

### Jugando con Punteros

- Son **variables normales** y corrientes
- Pensadas para guardar una **dirección de memoria**
- El tipo del puntero hace referencia al tipo de dato **al que apunta**





# ¿Qué es un puntero?

## Punteros

¿Qué son?

Ejemplo

Sintaxis

Arrays y

punteros

Ejemplo

Recorriendo

arrays

Jugando con

Punteros

```
int *ptr;
```

al sumar/restar se hace de 4 en 4 bytes

al desreferenciar obtengo un char

guardo una dirección de memoria

# Ejemplo

## Punteros

¿Qué son?

**Ejemplo**

Sintaxis

Arrays y  
punteros

**Ejemplo**

Recorriendo  
arrays

Jugando con  
Punteros

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int i = 42;
6     int *pi;
7
8     pi = &i;
9     printf("dir = %p\n", pi);
10    printf("val = %d\n", *pi);
11
12    *pi = 24;
13    printf("val = %d\n", i);
14
15    return 0;
16 }
```

## Punteros

¿Qué son?

Ejemplo

**Sintaxis**

Arrays y

punteros

Ejemplo

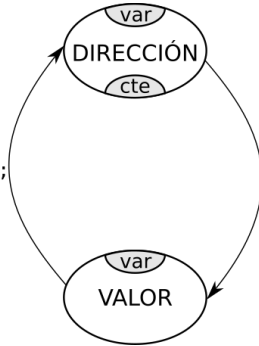
Recorriendo

arrays

Jugando con

Punteros

```
int *pi = &i;
```



```
int i = *pi;
```

```
int i = *(int *) (0x23845672);
```

# Arrays y punteros

## Punteros

¿Qué son?

Ejemplo

Sintaxis

Arrays y  
punteros

Ejemplo

Recorriendo  
arrays

Jugando con  
Punteros

- **Arrays:**

- Son prácticamente punteros constantes (no se puede modificar la dirección a la que apunta)
- Apuntan al primer elemento del array
- Mediante el tipo y el índice se obtiene la dirección del elemento deseado

- **Punteros:**

- Soportan las operaciones de suma y resta de enteros
- Al sumar un entero y un puntero estamos sumando a la dirección de memoria ese entero por el tamaño del tipo del puntero
- Se pueden indexar como un array

# Arrays y punteros

## Punteros

¿Qué son?

Ejemplo

Sintaxis

Arrays y punteros

Ejemplo

Recorriendo arrays

Jugando con Punteros

- **Arrays:**
  - Son prácticamente punteros constantes (no se puede modificar la dirección a la que apunta)
  - Apuntan al primer elemento del array
  - Mediante el tipo y el índice se obtiene la dirección del elemento deseado
- **Punteros:**
  - Soportan las operaciones de suma y resta de enteros
  - Al sumar un entero y un puntero estamos sumando a la dirección de memoria ese entero por el tamaño del tipo del puntero
  - Se pueden indexar como un array

# Arrays y punteros

## Punteros

¿Qué son?

Ejemplo

Sintaxis

Arrays y  
punteros

Ejemplo

Recorriendo  
arrays

Jugando con  
Punteros

- **Arrays:**
  - Son prácticamente punteros constantes (no se puede modificar la dirección a la que apunta)
  - Apuntan al primer elemento del array
  - Mediante el tipo y el índice se obtiene la dirección del elemento deseado
- **Punteros:**
  - Soportan las operaciones de suma y resta de enteros
  - Al sumar un entero y un puntero estamos sumando a la dirección de memoria ese entero por el tamaño del tipo del puntero
  - Se pueden indexar como un array

# Arrays y punteros

## Punteros

¿Qué son?

Ejemplo

Sintaxis

Arrays y  
punteros

Ejemplo

Recorriendo  
arrays

Jugando con  
Punteros

- **Arrays:**
  - Son prácticamente punteros constantes (no se puede modificar la dirección a la que apunta)
  - Apuntan al primer elemento del array
  - Mediante el tipo y el índice se obtiene la dirección del elemento deseado
- **Punteros:**
  - Soportan las operaciones de suma y resta de enteros
  - Al sumar un entero y un puntero estamos sumando a la dirección de memoria ese entero por el tamaño del tipo del puntero
  - Se pueden indexar como un array

# Arrays y punteros

## Punteros

¿Qué son?

Ejemplo

Sintaxis

Arrays y  
punteros

Ejemplo

Recorriendo  
arrays

Jugando con  
Punteros

- **Arrays:**
  - Son prácticamente punteros constantes (no se puede modificar la dirección a la que apunta)
  - Apuntan al primer elemento del array
  - Mediante el tipo y el índice se obtiene la dirección del elemento deseado
- **Punteros:**
  - Soportan las operaciones de suma y resta de enteros
  - Al sumar un entero y un puntero estamos sumando a la dirección de memoria ese entero por el tamaño del tipo del puntero
  - Se pueden indexar como un array



# Ejemplo

## Punteros

¿Qué son?

Ejemplo

Sintaxis

Arrays y

punteros

**Ejemplo**

Recorriendo

arrays

Jugando con

Punteros

## Ejemplo:

```
1 int array[3] = {1, 2, 3};
2 int *p = array;
3 int i;
4
5 /* Todas las direcciones iguales */
6 printf("%p\n%p\n%p\n", array, p, &array[0]);
7
8 p[2] = 22;
9 p += 1;
10 *p = 33;
11 *(p - 1) = 11;
12
13 /* Que imprimira? */
14 for (i = 0; i < 3; i++)
15     printf("%d\n", array[i]);
```

# Formas de recorrer un array

## Punteros

### ¿Qué son?

#### Ejemplo Sintaxis

### Arrays y punteros

#### Ejemplo Recorriendo arrays

### Jugando con Punteros

```
1 int main() {
2     int i;
3     p = array;
4
5     // forma 1: Contador e incremento de puntero
6     for (i = 0; i < 5; i++)
7         printf("%d\t", *p++);
8
9     // forma 2: Incremento de puntero y comparacion de
10         direcciones
11     for (p = array; p <= &array[4]; p++)
12         printf("%d\t", *p);
13
14     // forma 3: Contador y puntero como array
15     for (i = 0, p = array; i < 5; i++)
16         printf("%d\t", p[i]);
17
18     return 0;
19 }
```

# Jugando con Punteros

## Punteros

¿Qué son?

Ejemplo  
Sintaxis

Arrays y  
punteros

Ejemplo  
Recorriendo  
arrays

Jugando con  
Punteros

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int magic = 0x00796177;
6     printf("\nmagic = %0X\n", magic);
7     printf("magic = \"%s\"\n", (char *)&magic);
8
9     return 0;
10 }
```

## Funciones

### Funciones

Parámetros

Paso por  
copia

Paso por  
referencia

# Funciones

## Tema 7

## Funciones

### Funciones

#### Parámetros

Paso por copia

Paso por referencia

- 23 Funciones
  - Paso de parámetros
    - Paso por copia
    - Paso por referencia

# Funciones

## Funciones

### Funciones

#### Parámetros

Paso por copia

Paso por referencia

En C las funciones:

- Retornan un solo valor o nada (*void*)
- De cero a N parámetros
- Cada parámetro es de un tipo específico
- Todos los parámetros se pasan **por copia**
- Tienen una **declaración** y una **definición**
- Una función ha de estar declarada antes de ser llamada
- Una función no tiene por que estar definida a la hora de ser llamada

```
#include <stdio.h>

/* Declaracion */
int f(int a, int b);

int main()
{
    /* Llamada */
    printf("%d\n", f(2, 3));

    return 0;
}

/* Definicion */
int f(int a, int b)
{
    return a + b;
}
```

# Paso por copia

## Funciones

### Funciones

#### Parámetros

#### Paso por copia

#### Paso por referencia

- Siempre se copia el parámetro (variable o constante) que se le pasa a la función al llamarla
- Dentro de la función se trabaja con la copia
- Las variables originales no se ven afectadas

```
1 #include <stdio.h>
2
3 void f(int a)
4 {
5     a = 33;
6 }
7
8 int main()
9 {
10     int a = 3;
11     f(a);
12     printf("%d\n", a);
13
14     return 0;
15 }
```

# Paso por referencia

## Funciones

### Funciones

#### Parámetros

#### Paso por copia

#### Paso por referencia

- Para poder modificar las variables originales dentro de una función, esta ha de trabajar con la **referencia** a la variable, no con la original
- Esto se consigue con **punteros**

```
1 #include <stdio.h>
2
3 void f(int *a)
4 {
5     *a = 33;
6 }
7
8 int main()
9 {
10     int a = 3;
11     f(&a);
12     printf("%d\n", a);
13
14     return 0;
15 }
```



## GIT

### Git

Características

Distribuido

VS

Centralizado

Funcionamiento

Cuenta en

GitHub

Plan gratuito

Confirmar

correo

Fork de mi

repositorio

Buscar mi

repositorio

Fork

Clonar mi

repositorio

Workflow

Cambios

git push

Pull request

Comprobación

Descripción

Solicitado

Revisiones

# GIT

## Tema 8

## GIT

### Git

Características  
Distribuido  
VS  
Centralizado  
Funcionamiento

### Cuenta en GitHub

Plan gratuito  
Confirmar  
correo

### Fork de mi repositorio

Buscar mi  
repositorio  
Fork  
Clonar mi  
repositorio

### Workflow

Cambios  
git push  
Pull request  
Comprobación  
Descripción  
Solicitado  
Revisiones

## 24 Git

- Características
- Distribuido VS Centralizado
- Funcionamiento

## 25 Cuenta en GitHub

- Plan gratuito
- Confirmar correo

## 26 Fork de mi repositorio

- Buscar mi repositorio
- Fork
- Clonar mi repositorio

## 27 Flujo de trabajo

- Crea y revisa tus cambios
- Sube los cambios a tu repositorio
- Crea un nuevo pull request
  - Comprobar los cambios
  - Descripción

## GIT

### Git

Características

Distribuido

VS

Centralizado

Funcionamiento

Cuenta en

GitHub

Plan gratuito

Confirmar

correo

Fork de mi

repositorio

Buscar mi

repositorio

Fork

Clonar mi

repositorio

Workflow

Cambios

git push

Pull request

Comprobación

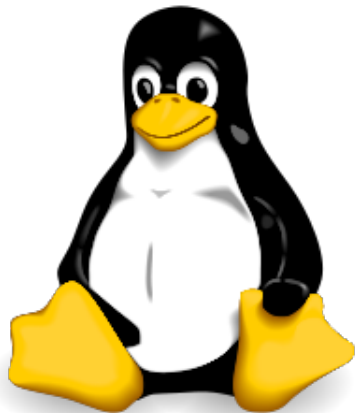
Descripción

Solicitado

Revisiones



# git



## GIT

### Git

#### Características

Distribuido  
VS  
Centralizado  
Funcionamiento

#### Cuenta en GitHub

Plan gratuito  
Confirmar  
correo

#### Fork de mi repositorio

Buscar mi  
repositorio  
Fork  
Clonar mi  
repositorio

#### Workflow

Cambios  
git push  
Pull request  
Comprobación  
Descripción  
Solicitado  
Revisiones

- Historial de versiones
- Visualización de cambios
- Revertir cambios
- Trabajo en equipo de forma concurrente
- Integridad de los archivos
- Sistema distribuido

# Distribuido VS Centralizado

## GIT

### Git

Características

**Distribuido**  
**VS**  
**Centralizado**

Funcionamiento

Cuenta en  
GitHub

Plan gratuito  
Confirmar  
correo

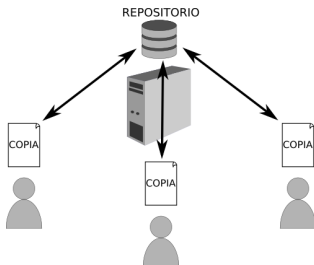
Fork de mi  
repositorio

Buscar mi  
repositorio  
Fork  
Clonar mi  
repositorio

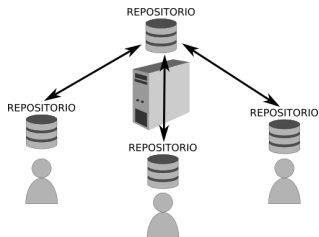
Workflow

Cambios  
git push  
Pull request  
Comprobación  
Solicitado  
Revisiones

## Centralizado



## Distribuido



## GIT

### Git

Características

Distribuido

VS

Centralizado

**Funcionamiento**

Cuenta en

GitHub

Plan gratuito

Confirmar

correo

Fork de mi

repositorio

Buscar mi

repositorio

Fork

Clonar mi

repositorio

Workflow

Cambios

git push

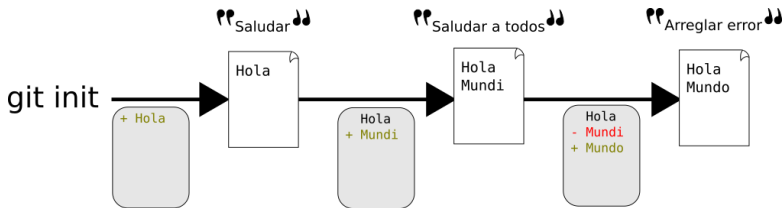
Pull request

Comprobación

Descripción

Solicitado

Revisiones



- Instantáneas del estado del repo
- Un comentario por cada instantánea
- Solo se guardan las diferencias
- Máquina de el tiempo

# Cuenta en GitHub

## GIT

[github.com](https://github.com)

Elegimos un nombre de usuario, una contraseña e introducimos nuestro correo

---



The screenshot shows the GitHub sign-up form with the following fields and values:

- Username: 4lice ✓
- Email: alice@mailinator.org ✓
- Password: [masked] ✓

Use at least one letter, one numeral, and seven characters.

[Sign up for GitHub](#)

By clicking "Sign up for GitHub", you agree to our [terms of service](#) and [privacy policy](#). We'll occasionally send you account related emails.

A red arrow points to the "Sign up for GitHub" button.

## Git

Características

Distribuido

VS

Centralizado

Funcionamiento

## Cuenta en GitHub

Plan gratuito

Confirmar correo

## Fork de mi repositorio

Buscar mi repositorio

Fork

Clonar mi repositorio

## Workflow

Cambios

git push

Pull request

Comprobación

Descripción

Solicitado

Revisiones

Nos aseguramos de que el plan gratuito está seleccionado y hacemos click en “*Finish sign up*”

## Welcome to GitHub

You've taken your first step into a larger world, @4lice.

**Completed**  
Set up a personal account
  **Step 2:**  
Choose your plan
  **Step 3:**  
Go to your dashboard

### Choose your personal plan

Plan	Cost <small>(view in EUR)</small>	Private repositories	
Large	\$50/month	50	<input type="button" value="Choose"/>
Medium	\$22/month	20	<input type="button" value="Choose"/>
Small	\$12/month	10	<input type="button" value="Choose"/>
Micro	\$7/month	5	<input type="button" value="Choose"/>
Free	\$0/month	0	<input checked="" type="button" value="Chosen"/>

### Each plan includes:

**Unlimited** collaborators  
**Unlimited** public repositories

- ✓ Free setup
- ✓ HTTPS Protection
- ✓ Email support
- ✓ Wikis, Issues, Pages, & more

Charges to your account will be made in US Dollars. Converted prices are provided as a convenience and are only an estimate based on current exchange rates. Local prices will change as the exchange rate fluctuates.

Don't worry, you can cancel or upgrade at any time.

**Help me set up an organization next**

Organizations are separate from personal accounts and are best suited for businesses who need to manage permissions for many employees.  
[Learn more about organizations.](#)



Debemos confirmar la dirección de correo

---

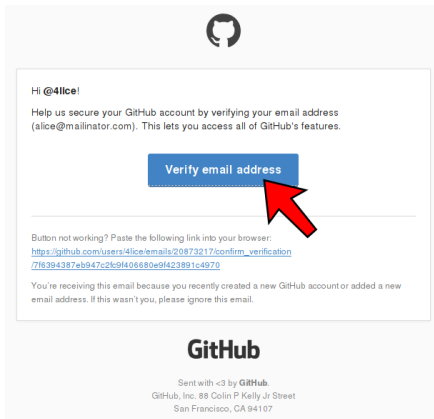


## Please verify your email address

Before you can contribute on GitHub, we need you to verify your email address.  
An email containing verification instructions was sent to **alice@mailinator.com**.

Didn't get the email? [Resend verification email](#) or [change your email settings](#).

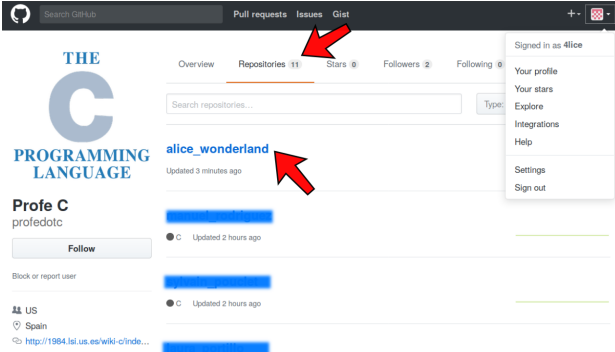
Buscamos el correo de confirmación en nuestro buzón y hacemos click en “*Verify email address*”



# Buscar mi repositorio

GIT

Entramos en la cuenta del profesor ([github.com/profedotc](https://github.com/profedotc)), y en la pestaña "Repositories" buscamos el repositorio que tenga nuestro nombre y apellido



The screenshot shows the GitHub profile page for user 'profedotc'. The 'Repositories' tab is selected and highlighted with a red arrow. Below the search bar, the repository 'alice\_wonderland' is listed, also highlighted with a red arrow. The repository is updated 3 minutes ago. The user's profile information shows they are from the US and Spain, with a link to a website.

Git

Características  
Distribuido  
VS  
Centralizado  
Funcionamiento

Cuenta en  
GitHub

Plan gratuito  
Confirmar  
correo

Fork de mi  
repositorio

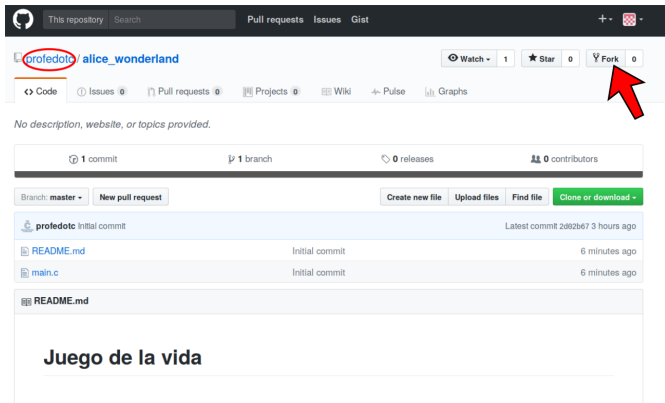
Buscar mi  
repositorio

Fork  
Clonar mi  
repositorio

Workflow

Cambios  
git push  
Pull request  
Comprobación  
Descripción  
Solicitado  
Revisiones

Hacemos clic en “Fork” para crear una copia del repositorio en nuestra cuenta

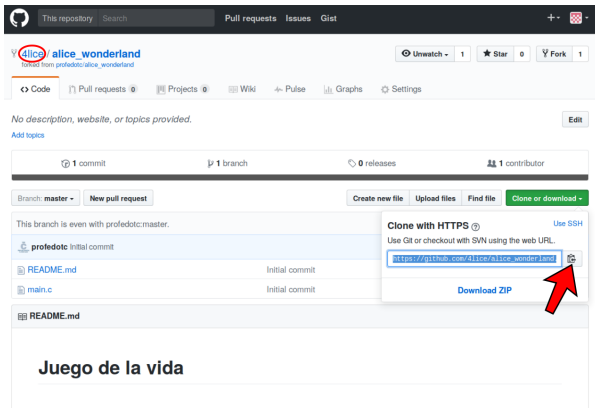


The screenshot shows the GitHub interface for the repository 'profedotc/alice\_wonderland'. The repository name is circled in red. A red arrow points to the 'Fork' button, which is located next to the 'Watch' and 'Star' buttons. Below the repository name, there are tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Pulse', and 'Graphs'. The repository has 1 commit, 1 branch, 0 releases, and 0 contributors. The 'Clone or download' button is highlighted in green. The repository contains files 'README.md' and 'main.c', both with initial commits. The README content is 'Juego de la vida'.

# Clonar mi repositorio

GIT

En el menú “*Clone or download*” podemos encontrar la URL necesaria para clonar nuestro repositorio



The screenshot shows the GitHub interface for the repository 'alice\_wonderland' by user 'profedotc'. The 'Clone or download' button is highlighted in green, and its dropdown menu is open. The menu shows 'Clone with HTTPS' as the selected option, with the URL 'https://github.com/alice/alice\_wonderland' displayed in a blue box. A red arrow points to the copy icon next to the URL. Other options in the menu include 'Use SSH' and 'Download ZIP'. The repository details show 1 commit, 1 branch, 0 releases, and 1 contributor. The main content area shows a file tree with 'README.md' and 'main.c', and a preview of the README content titled 'Juego de la vida'.

Git

Características

Distribuido

VS

Centralizado

Funcionamiento

Cuenta en  
GitHub

Plan gratuito

Confirmar

correo

Fork de mi  
repositorio

Buscar mi  
repositorio

Fork

Clonar mi  
repositorio

Workflow

Cambios

git push

Pull request

Comprobación

Descripción

Solicitado

Revisiones

# Clonar mi repositorio

## GIT

### Git

- Características
- Distribuido VS Centralizado
- Funcionamiento

### Cuenta en GitHub

- Plan gratuito
- Confirmar correo

### Fork de mi repositorio

- Buscar mi repositorio
- Fork
- Clonar mi repositorio**

### Workflow

- Cambios
- git push
- Pull request
- Comprobación
- Descripción
- Solicitado
- Revisiones

Para clonar nuestro repositorio abrimos un terminal, navegamos hasta la carpeta dónde lo queramos clonar y ejecutamos el siguiente comando de git:

```
> git clone https://github.com/profedotc/  
alice_wonderland.git
```

# Flujo de trabajo

## GIT

### Git

#### Características

Distribuido

VS

Centralizado

Funcionamiento

### Cuenta en GitHub

Plan gratuito

Confirmar correo

### Fork de mi repositorio

Buscar mi repositorio

Fork

Clonar mi repositorio

### Workflow

Cambios

git push

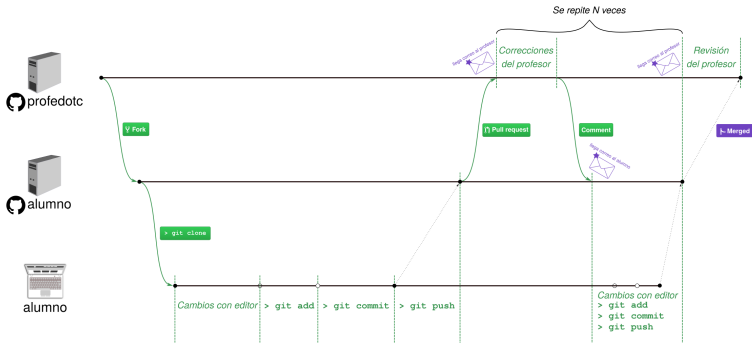
Pull request

Comprobación

Descripción

Solicitado

Revisiones



# Crea y revisa tus cambios

## GIT

```
> git diff
```

```

1 diff --git a/main.c b/main.c
2 index 7aa2631..3544b1d 100644
3 --- a/main.c
4 +++ b/main.c
5 @@ -2,10 +2,11 @@
6 #include <stdlib.h>
7 #include <stdbool.h>
8
9 --// TODO: Crea dos macros con el tamaño horizontal y vertical del
10     mundo
11 +#define W_SIZE_X 10
12 +#define W_SIZE_Y 10
13 void world_init(/* Recibo un mundo */);
14 -void world_print(/* Recibo un mundo */);
15 +void world_print(bool w[W_SIZE_X][W_SIZE_Y]);
16 void world_step(/* Recibo dos mundos */);
17 int world_count_neighbors(/* Recibo un mundo y unas coordenadas */)
18 ;
19 bool world_get_cell(/* Recibo un mundo y unas coordenadas */);

```

### Git

Características  
Distribuido  
VS  
Centralizado  
Funcionamiento

### Cuenta en GitHub

Plan gratuito  
Confirmar correo

### Fork de mi repositorio

Buscar mi  
repositorio  
Fork  
Clonar mi  
repositorio

### Workflow

**Cambios**  
git push  
Comprobación  
Descripción  
Solicitado  
Revisiones



# Crea y revisa tus cambios

## GIT

### Git

Características

Distribuido

VS

Centralizado

Funcionamiento

Cuenta en

GitHub

Plan gratuito

Confirmar

correo

Fork de mi

repositorio

Buscar mi

repositorio

Fork

Clonar mi

repositorio

Workflow

**Cambios**

git push

Pull request

Comprobación

Descripción

Solicitado

Revisiones

```
1 @@ -14,12 +15,13 @@ void world_copy(/* Recibo dos mundos */);
2 int main()
3 {
4     int i = 0;
5     - // TODO: Declara dos mundos
6     + bool world_a[W_SIZE_X][W_SIZE_Y];
7     + bool world_b[W_SIZE_X][W_SIZE_Y];
8
9     // TODO: inicializa el mundo
10    do {
11        printf("\033cIteration %d\n", i++);
12    - // TODO: Imprime el mundo
13    + world_print(world_a);
14        // TODO: Itera
15    } while (getchar() != 'q');
```

# Crea y revisa tus cambios

## GIT

```

1 @@ -37,21 +39,15 @@ void world_init(/* Recibo un mundo */)
2     */
3 }
4
5 -void world_print(/* Recibo un mundo */)
6 +void world_print(bool w[W_SIZE_X][W_SIZE_Y])
7 {
8 - // TODO: Imprimir el mundo por consola. Sugerencia:
9 - /*
10 - *      . # . . . . .
11 - *      . . # . . . . .
12 - *      # # # . . . . .
13 - *      . . . . .
14 - *      . . . . .
15 - *      . . . . .
16 - *      . . . . .
17 - *      . . . . .
18 - *      . . . . .
19 - *      . . . . .
20 - */
21 + for (int i = 0; i < W_SIZE_X; i++) {
22 +     for (int j = 0; j < W_SIZE_Y; j++) {
23 +         printf("%s", w[i][j] ? "#" : ".");
24 +     }
25 +     printf("\n");
26 + }
27 + printf("\n");
28 }
29
30 void world_step(/* Recibo dos mundos */)

```

## Git

Características

Distribuido  
VS  
Centralizado

Funcionamiento

Cuenta en  
GitHub

Plan gratuito  
Confirmar  
correo

Fork de mi  
repositorio

Buscar mi  
repositorio  
Fork

Clonar mi  
repositorio

Workflow

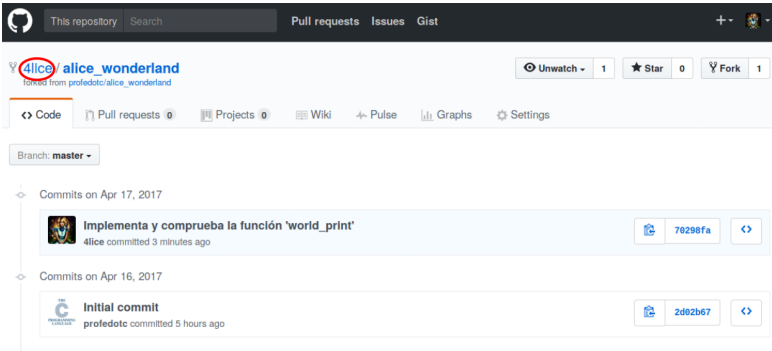
**Cambios**

git push  
Pull request  
Comprobación  
Descripción  
Solicitado  
Revisiones

# Sube los cambios a tu repositorio

## GIT

Subimos los cambios con `git push` y observamos que aparecen en GitHub



The screenshot shows a GitHub repository page for '4lice/alice\_wonderland'. The repository is forked from 'profedotc/alice\_wonderland'. The page displays the commit history for the 'master' branch, showing two commits:

- Initial commit** by profedotc, committed 5 hours ago. Commit hash: 2d62b67.
- Implementa y comprueba la función 'world\_print'** by 4lice, committed 3 minutes ago. Commit hash: 70298fa.

## Git

Características

Distribuido

VS

Centralizado

Funcionamiento

Cuenta en  
GitHub

Plan gratuito

Confirmar  
correo

Fork de mi  
repositorio

Buscar mi  
repositorio

Fork

Clonar mi  
repositorio

Workflow

Cambios

**git push**

Pull request

Comprobación

Descripción

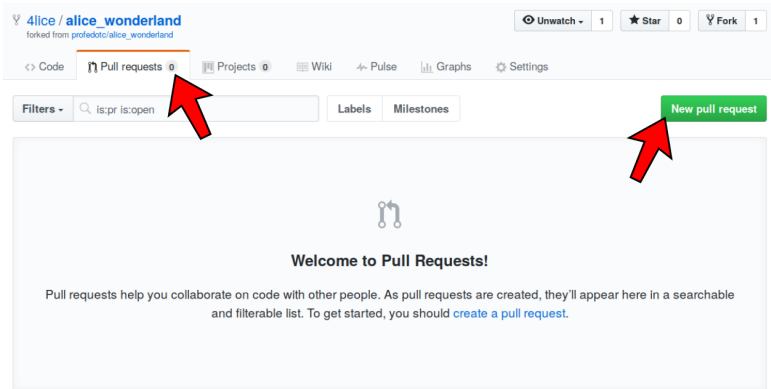
Solicitado

Revisiones

# Crea un nuevo pull request

## GIT

En la pestaña “*Pull requests*” pulsamos “*New pull request*” para crear un nuevo Pull Request



4lice / **alice\_wonderland**  
forked from profesdote/alice\_wonderland

Unwatch 1 Star 0 Fork 1

Code Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Filters - Search: is:pr is:open Labels Milestones

**New pull request**

**Welcome to Pull Requests!**

Pull requests help you collaborate on code with other people. As pull requests are created, they'll appear here in a searchable and filterable list. To get started, you should [create a pull request](#).

## Git

Características  
Distribuido  
VS  
Centralizado  
Funcionamiento

## Cuenta en GitHub

Plan gratuito  
Confirmar correo

## Fork de mi repositorio

Buscar mi repositorio  
Fork  
Clonar mi repositorio

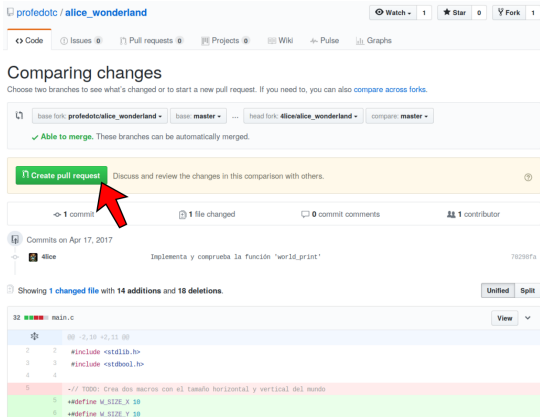
## Workflow

Cambios  
git push  
**Pull request**  
Comprobación  
Descripción  
Solicitado  
Revisiones

# Comprobar los cambios

## GIT

Volvemos a comprobar los diffs y pulsamos “*Create pull requests*”



profedotc / alice\_wonderland

Code Issues Pull requests Projects Wiki Pulse Graphs

### Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).

base fork: profedotc/alice\_wonderland base: master head fork: 4lice/alice\_wonderland compare: master

✓ Able to merge. These branches can be automatically merged.

**Create pull request** Discuss and review the changes in this comparison with others.

1 commit 1 file changed 0 commit comments 1 contributor

Commits on Apr 17, 2017

4lice Implementa y comprueba la función 'world\_print'

Showing 1 changed file with 14 additions and 18 deletions.

```

32 main.c
@@ -2,18 +2,11 @@
 2  #include <stdlib.h>
 3  #include <stdbool.h>
 4
 5  // TODO: Crea dos macros con el tamaño horizontal y vertical del mundo
 6  #define W_SIZE_X 10
 7  #define W_SIZE_Y 10
  
```

## Git

Características  
Distribuido  
VS  
Centralizado  
Funcionamiento

## Cuenta en GitHub

Plan gratuito  
Confirmar  
correo

## Fork de mi repositorio

Buscar mi  
repositorio  
Fork  
Clonar mi  
repositorio

## Workflow

Cambios  
git push  
Pull request

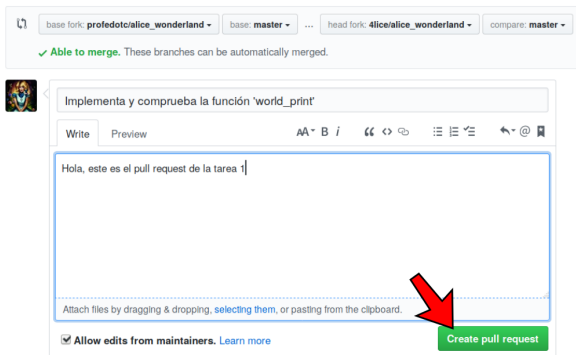
## Comprobación

Descripción  
Solicitado  
Revisiones

Escribimos un pequeño texto indicando la tarea que se entrega y los cambios realizados

## Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).



base fork: profedotc/alice\_wonderland - base: master - ... head fork: 4lice/alice\_wonderland - compare: master -

✓ Able to merge. These branches can be automatically merged.

Implementa y comprueba la función 'world\_print'

Write Preview AA B i « > ↻ ☰ ☷ ☹ ↶ @ 📧

Hola, este es el pull request de la tarea 1

Attach files by dragging & dropping, [selecting them](#), or pasting from the clipboard.

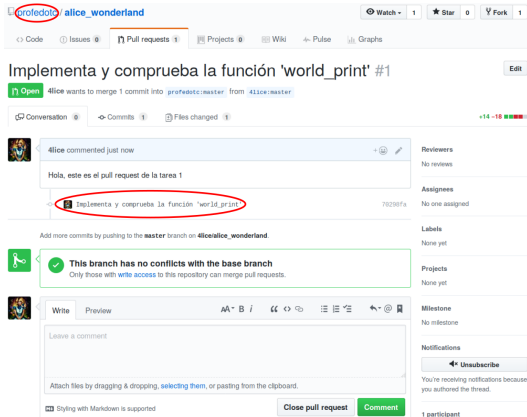
Allow edits from maintainers. [Learn more](#)

Create pull request

# Solicitud terminada

GIT

Si todo ha ido bien, deberíamos ver una pantalla parecida a la siguiente:



The screenshot shows a GitHub pull request interface for the repository 'profedotc/alice\_wonderland'. The pull request title is 'Implementa y comprueba la función 'world\_print' #1'. The pull request is from the '4lice:master' branch to the 'profedotc:master' branch. A comment from '4lice' is visible, stating 'Hola, este es el pull request de la tarea 1' and 'Implementa y comprueba la función 'world\_print''. A green checkmark indicates 'This branch has no conflicts with the base branch'. The interface includes navigation tabs for Code, Issues, Pull requests, Projects, Wiki, Pulse, and Graphs. The right sidebar shows sections for Reviewers, Assignees, Labels, Projects, Milestone, and Notifications.

Git

Características  
Distribuido  
VS  
Centralizado  
Funcionamiento

Cuenta en  
GitHub

Plan gratuito  
Confirmar  
correo

Fork de mi  
repositorio

Buscar mi  
repositorio  
Fork  
Clonar mi  
repositorio

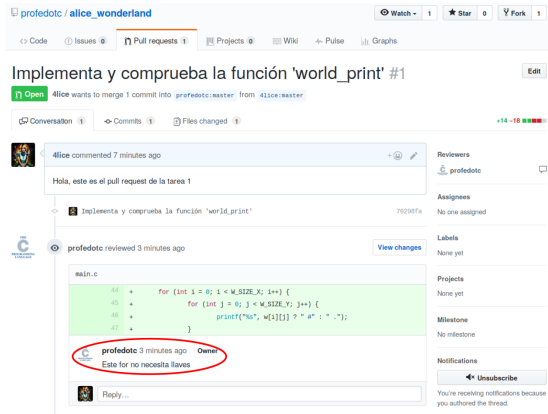
Workflow

Cambios  
git push  
Pull request  
Comprobación  
Descripción  
Solicitado  
Revisiones

# Esperar las revisiones del profesor

**GIT**

Cuando el profesor te haga correcciones, te llegará un correo y te aparecerán en la página del pull request



The screenshot shows a GitHub Pull Request interface. At the top, it says "profedotc / alice\_wonderland" with "Watch 1", "Star 0", and "Fork 1" buttons. Below that are tabs for "Code", "Issues", "Pull requests", "Projects", "Wiki", "Pulse", and "Graphs". The main title of the pull request is "Implementa y comprueba la función 'world\_print' #1". A green "Open" button is visible, along with the text "alice wants to merge 1 commit into profedotc:master from Alice:master".

The conversation section shows a comment from "alice" 7 minutes ago: "Hola, este es el pull request de la tarea 1". Below it is a commit titled "Implementa y comprueba la función 'world\_print'" by "79290fa".

A review from "profedotc" 3 minutes ago is shown. The code snippet is:
 

```

main.c
44 +   for (int i = 0; i < M.SIZE_X; i++) {
45 +       for (int j = 0; j < M.SIZE_Y; j++) {
46 +           printf("%c", w[i][j] ? "a" : ".");
47 +       }
    
```

 The review comment says "Este for no necesita llaves" (This for loop doesn't need braces) and is circled in red. The reviewer's name "profedotc" and the time "3 minutos ago" are also circled in red.

On the right side, there are sections for "Reviewers" (listing "profedotc"), "Assignees" (None assigned), "Labels" (None yet), "Projects" (None yet), "Milestone" (None milestone), and "Notifications" (Unsubscribe button).

Git

Características  
Distribuido  
VS  
Centralizado  
Funcionamiento

Cuenta en  
GitHub

Plan gratuito  
Confirmar  
correo

Fork de mi  
repositorio

Buscar mi  
repositorio  
Fork  
Clonar mi  
repositorio

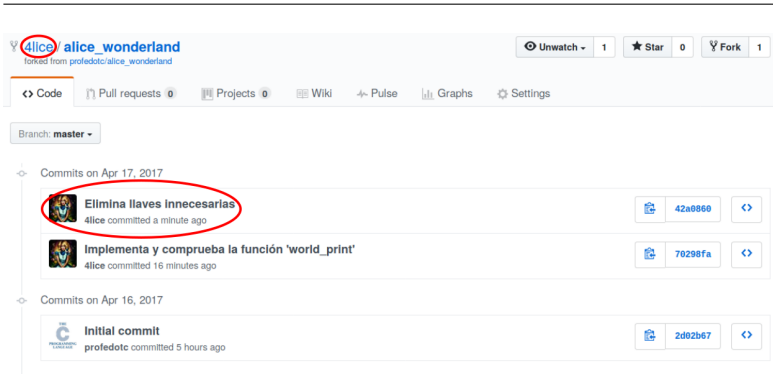
Workflow

Cambios  
git push  
Pull request  
Comprobación  
Descripción  
Solicitado  
Revisiones



## GIT

Crea un nuevo commit (o varios) para solucionar las correcciones que te hayan pedido



The screenshot shows the GitHub interface for the repository '4lice/alice\_wonderland'. The repository name '4lice' is circled in red. The page displays the commit history for the 'master' branch, with the following commits listed:

- Elimina llaves innecesarias** (4lice committed a minute ago) - This commit is circled in red. It has a commit hash of 42a8860.
- Implementa y comprueba la función 'world\_print'** (4lice committed 16 minutes ago) - It has a commit hash of 70298fa.
- Initial commit** (profedotc committed 5 hours ago) - It has a commit hash of 2d02b67.

Git

Características  
Distribuido  
VS  
Centralizado  
Funcionamiento

Cuenta en  
GitHub

Plan gratuito  
Confirmar  
correo

Fork de mi  
repositorio

Buscar mi  
repositorio  
Fork

Clonar mi  
repositorio

Workflow

Cambios  
git push  
Pull request  
Comprobación  
Descripción  
Solicitado  
Revisiones

En la página del pull request deben aparecer los nuevos commits automáticamente



The screenshot shows a GitHub pull request interface. At the top, the repository name is 'profedotc / alice\_wonderland'. Below that, there are tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Pulse', and 'Graphs'. The main title of the pull request is 'Implementa y comprueba la función 'world\_print' #1'. A green 'Open' button is visible, along with the text 'Alice wants to merge 2 commits into profedotc:master from 4lice:master'. Below the title, there are statistics: 'Conversation 1', 'Commits 2', and 'Files changed 1'. The main content area shows a list of commits. The first commit is by '4lice' with the message 'Implementa y comprueba la función 'world\_print'' and commit hash '76298fa'. Below it, there is a review by 'profedotc' from 7 minutes ago. Under the review, a file named 'main.c' is listed with a 'Show outdated' button. The second commit in the list is by '4lice' with the message 'Elimina llaves innecesarias' and commit hash '42a8860'. This second commit is circled in red in the original image.

Add more commits by pushing to the **master** branch on **4lice/alice\_wonderland**.

- Git
- Características
- Distribuido VS Centralizado
- Funcionamiento
- Cuenta en GitHub
- Plan gratuito
- Confirmar correo
- Fork de mi repositorio
- Buscar mi repositorio
- Fork
- Clonar mi repositorio
- Workflow
- Cambios
- git push
- Pull request
- Comprobación
- Descripción
- Solicitado
- Revisiones

# Espera a nuevas revisiones o a que sea aceptado

## GIT

### Git

Características  
Distribuido  
VS  
Centralizado  
Funcionamiento

### Cuenta en GitHub

Plan gratuito  
Confirmar  
correo

### Fork de mi repositorio

Buscar mi  
repositorio  
Fork  
Clonar mi  
repositorio

### Workflow

Cambios  
git push  
Pull request  
Comprobación  
Descripción  
Solicitado  
Revisiones

profedotc / alice\_wonderland

Unwatch 1 Star 0 Fork 1

Code Issues 0 Pull requests 1 Projects 0 Wiki Pulse Graphs Settings

## Implementa y comprueba la función 'world\_print' #1

Merged profedotc merged 2 commits into profedotc:master from 41ice:master 29 seconds ago

Conversation 2 Commits 2 Files changed 1 +13 -18

41ice commented 16 minutos ago First-time contributor

Hola, este es el pull request de la tarea 1

- Implementa y comprueba la función 'world\_print' 76298fa

profedotc reviewed 13 minutos ago View changes

main.c Show outdated

New changes since you last viewed View changes

- Elimina llaves innecesarias 42ab600

profedotc approved these changes a minute ago View changes

¡Genial! Hago merge, pasa a la siguiente tarea.

profedotc merged commit edcc0d9 into profedotc:master 29 seconds ago Revert

Reviewers: profedotc ✓

Assignees: No one—assign yourself

Labels: None yet

Projects: None yet

Milestone: No milestone

Notifications: Unsubscribe

You're receiving notifications because you modified the open/close state.

2 participants

## GIT

### Git

Características  
Distribuido  
VS  
Centralizado  
Funcionamiento

### Cuenta en GitHub

Plan gratuito  
Confirmar correo

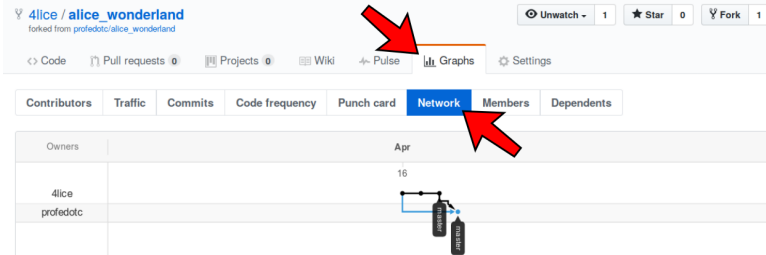
### Fork de mi repositorio

Buscar mi  
repositorio  
Fork  
Clonar mi  
repositorio

### Workflow

Cambios  
git push  
Pull request  
Comprobación  
Descripción  
Solicitado  
Revisiones

Puedes ver tu pull request gráficamente en la sección “*Network*” de la pestaña “*Graphs*”



4lice / **alice\_wonderland**  
forked from profedotc/alice\_wonderland

Unwatch 1 Star 0 Fork 1

Code Pull requests 0 Projects 0 Wiki Pulse **Graphs** Settings

Contributors Traffic Commits Code frequency Punch card **Network** Members Dependents

Owners

Apr

16

4lice

profedotc

network

master

## Estructuras

### Struct

- Alineación y tamaño
- Anidamiento
- Anónimas
- Arrays y punteros

### Union

- Ejemplo 1
- Ejemplo 2

### Campos de bits

- Ejemplo

### Enumerados

- Macros
- Ejemplos
- Enum
- Ejemplo

### Inicialización

# Estructuras de datos

## Tema 9

## Estructuras

### Struct

- Alineación y tamaño
- Anidamiento
- Anónimas
- Arrays y punteros

### Union

- Ejemplo 1
- Ejemplo 2

### Campos de bits

- Ejemplo

### Enumerados

- Macros
- Ejemplos
- Enum
- Ejemplo

### Inicialización

## 28 Struct

- Alineación y tamaño
- Anidamiento
- Estructuras anónimas
- Arrays y punteros

## 29 Union

- Ejemplo 1
- Ejemplo 2

## 30 Campos de bits

- Ejemplo

## 31 Enumerados

- Macros: El preprocesador de C
  - Ejemplos
- Enum
  - Ejemplo

## 32 Designated initializers

# Struct

## Estructuras

Lista de variables agrupadas físicamente en un mismo bloque de memoria.

### Struct

Alineación y tamaño  
Anidamiento  
Anónimas  
Arrays y punteros

### Union

Ejemplo 1  
Ejemplo 2

### Campos de bits

Ejemplo

### Enumerados

Macros  
Ejemplos  
Enum  
Ejemplo

### Inicialización

```
1 struct person {  
2     char name[256];  
3     char surname[256];  
4     unsigned char age;  
5     unsigned int phone;  
6 };  
7  
8 int main()  
9 {  
10    struct person p = {"Man", "Bat", 35, 69813244};  
11  
12    printf("%s%s\n", p.surname, p.name);  
13  
14    return 0;  
15 }
```

# Alineación y tamaño

## Estructuras

### Struct

#### Alineación y tamaño

#### Anidamiento

#### Anónimas

#### Arrays y punteros

### Union

#### Ejemplo 1

#### Ejemplo 2

### Campos de bits

#### Ejemplo

### Enumerados

#### Macros

#### Ejemplos

#### Enum

#### Ejemplo

### Inicialización

```
struct ejemplo
{
    uint8_t  v1; /* 1 */
    uint32_t v2; /* 4 */
    uint32_t v3; /* 4 */
};
```

`sizeof(struct ejemplo);` ¿5 bytes?



# Alineación y tamaño

## Estructuras

### Struct

#### Alineación y tamaño

#### Anidamiento

#### Anónimas

#### Arrays y punteros

### Union

#### Ejemplo 1

#### Ejemplo 2

### Campos de bits

#### bits

#### Ejemplo

### Enumerados

#### Macros

#### Ejemplos

#### Enum

#### Ejemplo

### Inicialización

```
struct ejemplo
{
    uint8_t v1; /* 1 */
    uint32_t v2; /* 4 */
    uint32_t v3; /* 4 */
};
```



`sizeof(struct ejemplo);` ¿5 bytes? → 8 bytes

## Estructuras

### Struct

Alineación y tamaño

**Anidamiento**

Anónimas

Arrays y punteros

### Union

Ejemplo 1

Ejemplo 2

### Campos de bits

Ejemplo

### Enumerados

Macros

Ejemplos

Enum

Ejemplo

### Inicialización

```
1 struct A {
2     int a;
3     struct B {
4         int b;
5     } stb;
6 };
7
8 int main()
9 {
10     struct A a = {1, {2}};
11
12     printf("a = {%d, {%d}}\n", a.a, a.stb.b);
13
14     return 0;
15 }
```

# Estructuras anónimas

## Estructuras

### Struct

Alineación y tamaño

Anidamiento

**Anónimas**

Arrays y punteros

### Union

Ejemplo 1

Ejemplo 2

### Campos de bits

Ejemplo

### Enumerados

Macros

Ejemplos

Enum

Ejemplo

### Inicialización

```
1 struct A {
2     int a;
3     struct {
4         int b;
5     };
6 };
7
8 int main()
9 {
10     struct A a = {1, {2}};
11
12     printf("a = {%d, {%d}}\n", a.a, a.b);
13
14     return 0;
15 }
```

# Arrays y punteros

## Estructuras

### Struct

Alineación y tamaño

Anidamiento

Anónimas

**Arrays y punteros**

### Union

Ejemplo 1

Ejemplo 2

### Campos de bits

Ejemplo

### Enumerados

Macros

Ejemplos

Enum

Ejemplo

### Inicialización

```
1 #include <stdio.h>
2
3 struct cell {
4     int state;
5     int size;
6 };
7
8 int main()
9 {
10     struct cell culture[5] = {{1,10}, {0,342}, {1,7},
11                             {1,50}, {1,77}};
12     struct cell *c;
13
14     for (c = culture; c <= &culture[4]; c++)
15         printf("%d, %d\n", c->state, c->size);
16
17     return 0;
18 }
```

# Union

## Estructuras

### Struct

Alineación y tamaño  
Anidamiento  
Anónimas  
Arrays y punteros

### Union

Ejemplo 1  
Ejemplo 2

### Campos de bits

Ejemplo

### Enumerados

Macros  
Ejemplos

Enum  
Ejemplo

### Inicialización

- Una unión es un valor que tiene varias representaciones o formatos
- Estructura que permite guardar varios tipos de datos en la misma zona de memoria

```
1 union float_int {  
2     float f;  
3     int i;  
4 };  
5  
6 int main()  
7 {  
8     union float_int fi;  
9  
10    fi.f = 2.7182;  
11    printf("%X\n", fi.i);  
12  
13    return 0;  
14 }
```

# Ejemplo 1

## Estructuras

### Struct

Alineación y tamaño

Anidamiento

Anónimas

Arrays y punteros

### Union

**Ejemplo 1**

Ejemplo 2

### Campos de bits

Ejemplo

### Enumerados

Macros

Ejemplos

Enum

Ejemplo

### Inicialización

```
1 union char_int {
2     struct {
3         char c1;
4         char c2;
5         char c3;
6         char c4;
7     };
8     int i;
9 };
10
11 int main()
12 {
13     union char_int ci;
14
15     ci.i = 1701999205;
16     printf("%c%c%c%c\n", ci.c1, ci.c2, ci.c3, ci.c4);
17
18     return 0;
19 }
```

## Ejemplo 2

### Estructuras

#### Struct

Alineación y tamaño  
Anidamiento  
Anónimas  
Arrays y punteros

#### Union

Ejemplo 1  
Ejemplo 2

#### Campos de bits

Ejemplo

#### Enumerados

Macros  
Ejemplos  
Enum  
Ejemplo

#### Inicialización

```
1 #include <stdio.h>
2
3 struct gtype {
4     int type;
5
6     union {
7         char char_t;
8         int int_t;
9         float float_t;
10    };
11 };
12
13 int main()
14 {
15     struct gtype gt;
16
17     gt.int_t = 3;
18     gt.type = 1;
19
```

```
20
21     switch (gt.type) {
22     case 0:
23         printf("%c\n", gt.char_t);
24         break;
25     case 1:
26         printf("%d\n", gt.int_t);
27         break;
28     case 2:
29         printf("%f\n", gt.float_t);
30         break;
31     default:
32         printf("error: invalid type\n");
33         break;
34     };
35
36     return 0;
37 }
```

# Campos de bits

## Estructuras

### Struct

Alineación y tamaño  
Anidamiento  
Anónimas  
Arrays y punteros

### Union

Ejemplo 1  
Ejemplo 2

### Campos de bits

Ejemplo

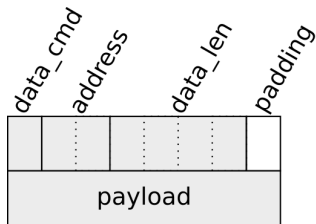
### Enumerados

Macros  
Ejemplos  
Enum  
Ejemplo

### Inicialización

- Característica de las estructuras y uniones que nos permite declarar campos de hasta un bit de longitud
- La memoria reservada es la que indica el tipo del campo
- Para acceder a nivel de bit se realizan numerosas operaciones por debajo

```
struct frame {
    uint16_t data_cmd   : 1;
    uint16_t address    : 2;
    uint16_t data_len   : 4;
    uint16_t            : 1;
    uint16_t payload    : 8;
};
```





# Ejemplo

## Estructuras

### Struct

Alineación y tamaño  
Anidamiento  
Anónimas  
Arrays y punteros

### Union

Ejemplo 1  
Ejemplo 2

### Campos de bits

#### Ejemplo

### Enumerados

Macros  
Ejemplos  
Enum  
Ejemplo

### Inicialización

```

1 union float_t{
2     float f;
3     struct {
4         uint32_t fra : 23;
5         uint32_t exp : 8;
6         uint32_t sig : 1;
7     };
8 };
9
10 int main()
11 {
12     union float_t f;
13
14     f.f = -3.1416;
15
16     printf("signo      = %u\n", f.sig);
17     printf("exponente = %d\n", f.exp);
18     printf("fraccion   = %d\n", f.fra);
19
20     return 0;
21 }

```

# Macros: El preprocesador de C

## Estructuras

### Struct

Alineación y tamaño  
Anidamiento  
Anónimas  
Arrays y punteros

### Union

Ejemplo 1  
Ejemplo 2

### Campos de bits

Ejemplo

### Enumerados

#### Macros

Ejemplos  
Enum  
Ejemplo

### Inicialización

- Preprocesador: Se ejecuta antes de compilar
- Lenguaje de **macros**
- Múltiples usos:
  - Declaración de constantes
  - Pequeñas funciones y utilidades
  - Compilación condicional de código
  - Depuración

# Ejemplos

## Estructuras

### Struct

Alineación y tamaño

Anidamiento

Anónimas

Arrays y punteros

### Union

Ejemplo 1

Ejemplo 2

### Campos de bits

Ejemplo

### Enumerados

Macros

**Ejemplos**

Enum

Ejemplo

### Inicialización

```

1  #include <stdio.h>
2
3  #define TAM_ARRAY 20
4  #define POW2(x) ((x)*(x))
5  #define PRINT 0
6
7  int main()
8  {
9      int array[TAM_ARRAY];
10     int i;
11
12     for (i = 0; i < TAM_ARRAY; i++)
13         array[i] = POW2(i);
14
15     #if PRINT == 1
16         for (i = 0; i < TAM_ARRAY; i++)
17             printf("%i ", array[i]);
18     #elif PRINT == -1
19         printf("Array initialized\n");
20     #else
21         #warning PRINT may be 1 or -1
22         printf("Error in %s:%d\n", __FILE__, __LINE__);
23     #endif
24
25     return 0;
26 }
```

## Estructuras

### Struct

Alineación y tamaño  
Anidamiento  
Anónimas  
Arrays y punteros

### Union

Ejemplo 1  
Ejemplo 2

### Campos de bits

Ejemplo

### Enumerados

Macros  
Ejemplos

**Enum**  
Ejemplo

### Inicialización

```
enum estado_coche {ARRANCADO, PARADO, EN_MARCHA,  
DETENIDO};
```

- **Tipo** formado por una lista de macros
- Las macros toman valores enteros de forma consecutiva

# Ejemplo

## Estructuras

### Struct

Alineación y tamaño  
Anidamiento  
Anónimas  
Arrays y punteros

### Union

Ejemplo 1  
Ejemplo 2

### Campos de bits

Ejemplo

### Enumerados

Macros  
Ejemplos  
Enum  
Ejemplo

### Inicialización

```

1 #include <stdio.h>
2
3 struct person {
4     char name[256];
5     char surname[256];
6     unsigned char age;
7     unsigned int phone;
8 };
9
10 enum person_attr {
11     NAME,
12     SURNAME,
13     AGE,
14     PHONE
15 };
16
17 int main ()
18 {
19     person p = {"Alice", "Smith",
20                25, 12434321};
21     enum person_attr choice =
22         NAME;

```

```

21
22     switch (choice) {
23     case NAME:
24         printf("%s\n", p.name);
25         break;
26     case SURNAME:
27         printf("%s\n", p.surname);
28         break;
29     case AGE:
30         printf("%d\n", p.age);
31         break;
32     case PHONE:
33         printf("%d\n", p.phone);
34         break;
35     default:
36         printf("error: %s:%d",
37                __FILE__, __LINE__);
38     }
39     return 0;
40 }

```

# Designated initializers

## Estructuras

### Struct

Alineación y tamaño

Anidamiento

Anónimas

Arrays y punteros

### Union

Ejemplo 1

Ejemplo 2

### Campos de bits

Ejemplo

### Enumerados

Macros

Ejemplos

Enum

Ejemplo

### Inicialización

```
1 struct bug {
2     unsigned int legs;
3     unsigned char color[3];
4     const char *name;
5 } bugs[30] = {
6     [0] = {
7         .legs = 6,
8         .color = {100, 100, 100},
9         .name = "ant",
10    },
11    [1] = {
12        .legs = 8,
13        .color = {[0] = 200},
14        .name = "spider",
15    },
16    [2 ... 10] = { // Only GNU
17        .legs = 100,
18        .color = {[1] = 255},
19        .name = "centipede",
20    },
21 };
```

## C Modular

Cabeceras  
Ejemplo

Compilación

Make

Estructura  
Ejemplo

# C Modular

## Tema 10

## C Modular

Cabeceras

Ejemplo

Compilación

Make

Estructura

Ejemplo

**33** Cabeceras

■ Ejemplo

**34** Compilación por bloques

**35** Make y Makefile

■ Estructura

■ Ejemplo



## C Modular

### Cabeceras

Ejemplo

### Compilación

### Make

Estructura

Ejemplo

- Podemos crear nuestros propios ficheros *\*.h*
- Un fichero de cabecera suele tener únicamente declaraciones y no definiciones
- En el fichero de código (*\*.c*) se definen las funciones declaradas en la cabecera
- Podemos tener una cabecera y varios tipos de definiciones
- Nos permite crear una interfaz que aisle al usuario de la definición de las funciones

# Ejemplo

## C Modular

Cabeceras

Ejemplo

Compilación

Make

Estructura

Ejemplo

## person.h

```
1 #ifndef _PERSON_H_
2 #define _PERSON_H_
3
4 #define MAX_NAME 256
5
6 struct person {
7     char name[MAX_NAME];
8     char surname[MAX_NAME];
9     unsigned int age;
10    int phone;
11 };
12
13 void print_person(const struct person *p);
14
15 #endif
```

## person.c

```
1 #include <stdio.h>
2 #include "person.h"
3
4 void print_person(const struct person *p)
5 {
6     printf("%s, %s\n", p->surname, p->name);
7     printf("\t- age: %d\n", p->age);
8     printf("\t- phone: %d\n", p->phone);
9 }
```

## Razones:

- Tiempo:
  - La compilación es un proceso complejo y costoso
  - Proyectos muy grandes pueden tardar mucho tiempo en compilar
  - Cuando se está desarrollando esto se vuelve prohibitivo
- Organización:
  - Dividir el código en bloques lógicos es una buena práctica
  - Mejora:
    - El mantenimiento
    - La legibilidad
    - La portabilidad
    - La escalabilidad
    - etc

# Compilación por bloques

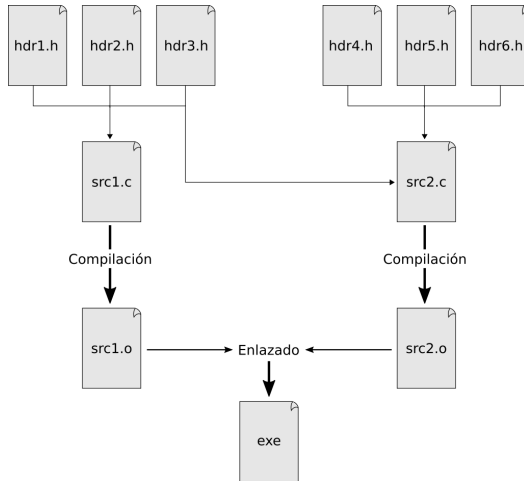
## C Modular

Cabeceras  
Ejemplo

Compilación

Make

Estructura  
Ejemplo



## C Modular

Cabeceras

Ejemplo

Compilación

Make

Estructura

Ejemplo

## Solución:

- Cada fichero de código puede compilarse de manera independiente, creando un **fichero objeto** (\*.o)
- El **linker** se encarga de enlazar todos los ficheros objetos para crear el ejecutable
- Un módulo objeto puede tener referencias a símbolos definidos en otro módulo

Cómo se genera: `gcc -c persona.c`

# Make y Makefile

## C Modular

Cabeceras  
Ejemplo

Compilación

**Make**

Estructura  
Ejemplo

Herramienta para automatizar la compilación del código (y mucho más)

- Gestiona dependencias para no compilar innecesariamente
- Facilita enormemente el trabajo
- También se suele utilizar para instalación y desinstalación

## C Modular

Cabeceras

Ejemplo

Compilación

Make

**Estructura**

Ejemplo

```
objetivo: prerequisite1 prerequisite2 ...  
ordenes para generar "objetivo"
```

- **Objetivos:** Un objetivo suele ser un archivo que se desea generar (por ejemplo, un ejecutable)
- **Prerequisitos:** Lista de objetivos
- **Órdenes:** Instrucciones a realizar para generar el objetivo. Siempre van precedidas de una tabulación

## makefile

```
1 all: mi_prog
2
3 mi_prog: main.o persona.o
4     gcc main.o persona.o -o mi_prog
5
6 main.o: main.c
7     gcc -c main.c
8
9 persona.o: persona.h persona.c
10    gcc -c persona.c
```



Dyn memory

Mapa de memoria

Ejemplo

Stack Overflow

Fragmentación

arrays  
multiD

Forma 1

Forma 2

Uso tras liberación

valgrind  
Ejemplo

# Reserva dinámica de memoria

## Tema 11

## Dyn memory

Mapa de memoria

Ejemplo

Stack Overflow

Fragmentación

arrays multiD

Forma 1

Forma 2

Uso tras liberación

valgrind

Ejemplo

- 36** Mapa de memoria
  - Ejemplo
- 37** Stack Overflow
- 38** Fragmentación
- 39** Reserva de arrays multidimensionales
  - Forma 1 (la mala)
  - Forma 2 (la buena)
- 40** Uso tras liberación
- 41** Depuración con valgrind
  - Ejemplo

# Mapa de memoria

## Dyn memory

### Mapa de memoria

#### Ejemplo

### Stack Overflow

### Fragmentación

### arrays multiD

#### Forma 1

#### Forma 2

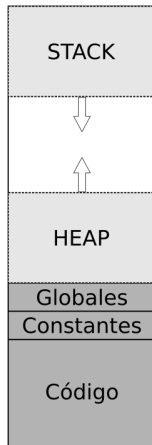
### Uso tras liberación

### valgrind

#### Ejemplo

- **Heap:** Se almacena la memoria reservada **dinámicamente** con **malloc**
- **Stack:** Se almacenan las **variables locales** de cada llamada a función
- **Globales:** Todas las variables globales
- **Constantes:** Todas las constantes (números, cadenas, etc)
- **Código:** El programa en sí

Direcciones altas de memoria



Direcciones bajas de memoria

# Ejemplo

## Dyn memory

### Mapa de memoria

#### Ejemplo

### Stack Overflow

### Fragmentación

### arrays multiD

#### Forma 1 Forma 2

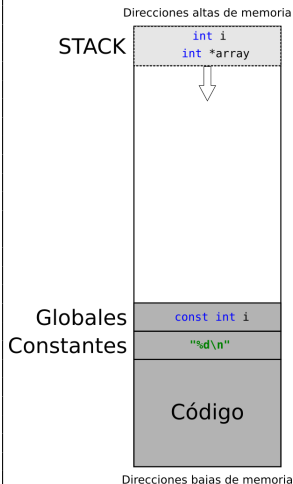
### Uso tras liberación

### valgrind Ejemplo

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int *array_alloc(int n)
5 {
6     int *p = (int *)malloc(n * sizeof(int))
7     return p;
8 }
9
10 static const int tam = 3;
11
12 int main()
13 {
14     int i;
15     int *array;
16
17     array = array_alloc(tam);
18
19     /* Inicializamos */
20     for (i = 0; i < tam; i++)
21         array[i] = i;
22
23     /* Imprimimos */
24     for (i = 0; i < tam; i++)
25         printf("%d\n", array[i]);
26
27     free(array);
28
29     return 0;
30 }

```



# Ejemplo

## Dyn memory

### Mapa de memoria

#### Ejemplo

### Stack Overflow

### Fragmentación

### arrays multiD

#### Forma 1

#### Forma 2

### Uso tras liberación

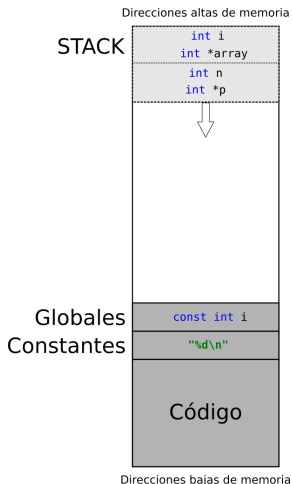
### valgrind

#### Ejemplo

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int *array_alloc(int n)
5 {
6     int *p = (int *)malloc(n * sizeof(int))
7     return p;
8 }
9
10 static const int tam = 3;
11
12 int main()
13 {
14     int i;
15     int *array;
16
17     array = array_alloc(tam);
18
19     /* Inicializamos */
20     for (i = 0; i < tam; i++)
21         array[i] = i;
22
23     /* Imprimimos */
24     for (i = 0; i < tam; i++)
25         printf("%d\n", array[i]);
26
27     free(array);
28
29     return 0;
30 }

```



# Ejemplo

## Dyn memory

### Mapa de memoria

#### Ejemplo

### Stack Overflow

### Fragmentación

### arrays multiD

### Forma 1 Forma 2

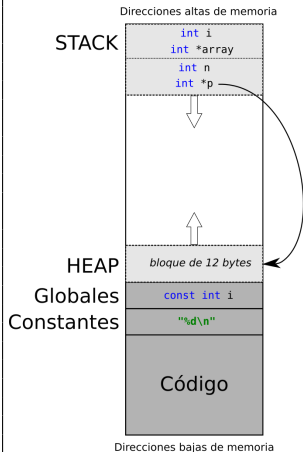
### Uso tras liberación

### valgrind Ejemplo

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int *array_alloc(int n)
5 {
6     int *p = (int *)malloc(n * sizeof(int))
7     return p;
8 }
9
10 static const int tam = 3;
11
12 int main()
13 {
14     int i;
15     int *array;
16
17     array = array_alloc(tam);
18
19     /* Inicializamos */
20     for (i = 0; i < tam; i++)
21         array[i] = i;
22
23     /* Imprimimos */
24     for (i = 0; i < tam; i++)
25         printf("%d\n", array[i]);
26
27     free(array);
28
29     return 0;
30 }

```



# Ejemplo

## Dyn memory

### Mapa de memoria

#### Ejemplo

### Stack Overflow

### Fragmentación

### arrays multiD

### Forma 1 Forma 2

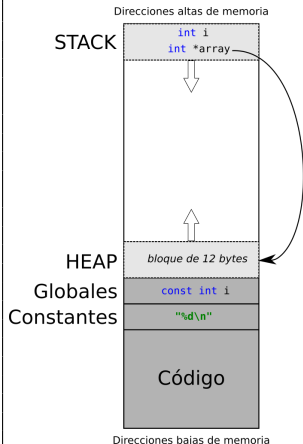
### Uso tras liberación

### valgrind Ejemplo

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int *array_alloc(int n)
5 {
6     int *p = (int *)malloc(n * sizeof(int))
7     return p;
8 }
9
10 static const int tam = 3;
11
12 int main()
13 {
14     int i;
15     int *array;
16
17     array = array_alloc(tam);
18
19     /* Inicializamos */
20     for (i = 0; i < tam; i++)
21         array[i] = i;
22
23     /* Imprimimos */
24     for (i = 0; i < tam; i++)
25         printf("%d\n", array[i]);
26
27     free(array);
28
29     return 0;
30 }

```



# Ejemplo

## Dyn memory

### Mapa de memoria

#### Ejemplo

### Stack Overflow

### Fragmentación

### arrays multiD

#### Forma 1 Forma 2

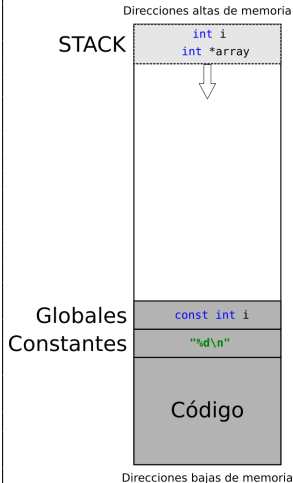
### Uso tras liberación

### valgrind Ejemplo

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int *array_alloc(int n)
5 {
6     int *p = (int *)malloc(n * sizeof(int))
7     return p;
8 }
9
10 static const int tam = 3;
11
12 int main()
13 {
14     int i;
15     int *array;
16
17     array = array_alloc(tam);
18
19     /* Inicializamos */
20     for (i = 0; i < tam; i++)
21         array[i] = i;
22
23     /* Imprimimos */
24     for (i = 0; i < tam; i++)
25         printf("%d\n", array[i]);
26
27     free(array);
28
29     return 0;
30 }

```





# Stack Overflow

## Dyn memory

### Mapa de memoria

#### Ejemplo

### Stack Overflow

#### Fragmentación

#### arrays multiD

#### Forma 1 Forma 2

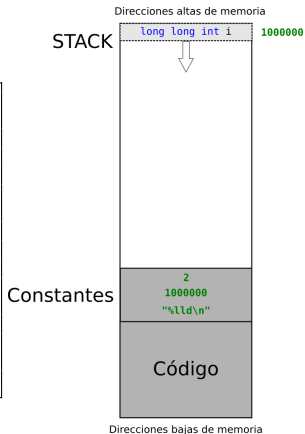
#### Uso tras liberación

#### valgrind Ejemplo

El uso de funciones recursivas puede ocasionar un agotamiento de la pila

```

1  #include <stdio.h>
2
3  long long int sum_all(long long int i)
4  {
5      if (i >= 2)
6          return i + sum_all(i - 1);
7      else
8          return i;
9  }
10
11 int main()
12 {
13     printf("%lld\n", sum_all(1000000));
14
15     return 0;
16 }
    
```



# Stack Overflow

## Dyn memory

### Mapa de memoria

#### Ejemplo

### Stack Overflow

#### Fragmentación

#### arrays multiD

#### Forma 1 Forma 2

#### Uso tras liberación

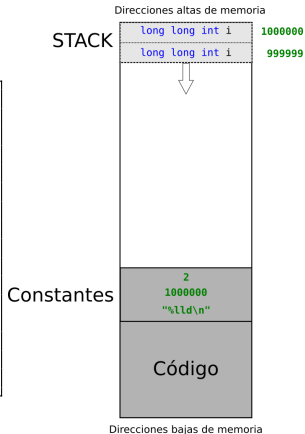
#### valgrind Ejemplo

El uso de funciones recursivas puede ocasionar un agotamiento de la pila

```

1 #include <stdio.h>
2
3 long long int sum_all(long long int i)
4 {
5     if (i >= 2)
6         return i + sum_all(i - 1);
7     else
8         return i;
9 }
10
11 int main()
12 {
13     printf("%lld\n", sum_all(1000000));
14
15     return 0;
16 }

```



# Stack Overflow

## Dyn memory

## Mapa de memoria

### Ejemplo

## Stack Overflow

## Fragmentación

## arrays multiD

### Forma 1 Forma 2

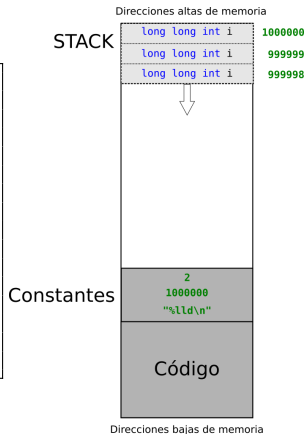
## Uso tras liberación

## valgrind Ejemplo

```

1 #include <stdio.h>
2
3 long long int sum_all(long long int i)
4 {
5     if (i >= 2)
6         return i + sum_all(i - 1);
7     else
8         return i;
9 }
10
11 int main()
12 {
13     printf("%lld\n", sum_all(1000000));
14
15     return 0;
16 }
    
```

El uso de funciones recursivas puede ocasionar un agotamiento de la pila



# Stack Overflow

## Dyn memory

## Mapa de memoria

### Ejemplo

## Stack Overflow

## Fragmentación

## arrays multiD

### Forma 1 Forma 2

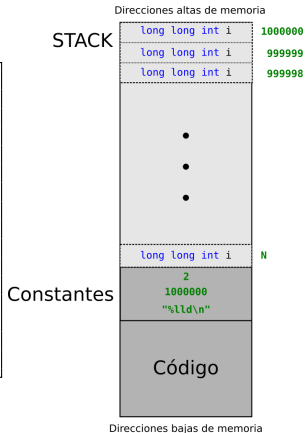
## Uso tras liberación

## valgrind Ejemplo

El uso de funciones recursivas puede ocasionar un agotamiento de la pila

```

1 #include <stdio.h>
2
3 long long int sum_all(long long int i)
4 {
5     if (i >= 2)
6         return i + sum_all(i - 1);
7     else
8         return i;
9 }
10
11 int main()
12 {
13     printf("%lld\n", sum_all(1000000));
14
15     return 0;
16 }
    
```



# Fragmentación

## Dyn memory

Fragmentos pequeños de memoria libre entre bloques de memoria reservada.

## Mapa de memoria

### Ejemplo

## Stack Overflow

## Fragmentación

## arrays multiD

### Forma 1

### Forma 2

## Uso tras liberación

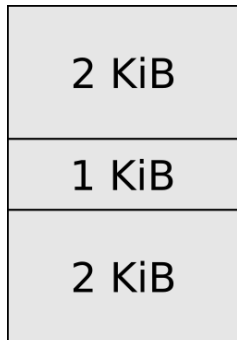
## valgrind

### Ejemplo

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     void *p1 = malloc(2 * 1024);
7     void *p2 = malloc(1 * 1024);
8     void *p3 = malloc(2 * 1024);
9
10    void *p;
11
12    printf("%ld\n", (long int)p1 / 1024);
13    printf("%ld\n", (long int)p2 / 1024);
14    printf("%ld\n", (long int)p3 / 1024);
15
16    free(p2);
17    p = malloc(2 * 1024);
18
19    printf("\n%ld\n", (long int)p / 1024);
20
21    return 0;
22 }

```



# Fragmentación

## Dyn memory

### Mapa de memoria

#### Ejemplo

### Stack Overflow

## Fragmentación

### arrays multiD

#### Forma 1 Forma 2

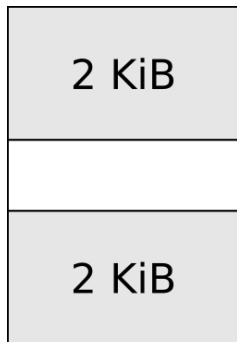
### Uso tras liberación

### valgrind Ejemplo

Fragmentos pequeños de memoria libre entre bloques de memoria reservada.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     void *p1 = malloc(2 * 1024);
7     void *p2 = malloc(1 * 1024);
8     void *p3 = malloc(2 * 1024);
9
10    void *p;
11
12    printf("%ld\n", (long int)p1 / 1024);
13    printf("%ld\n", (long int)p2 / 1024);
14    printf("%ld\n", (long int)p3 / 1024);
15
16    free(p2);
17    p = malloc(2 * 1024);
18
19    printf("\n%ld\n", (long int)p / 1024);
20
21    return 0;
22 }
    
```



# Fragmentación

## Dyn memory

Fragmentos pequeños de memoria libre entre bloques de memoria reservada.

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     void *p1 = malloc(2 * 1024);
7     void *p2 = malloc(1 * 1024);
8     void *p3 = malloc(2 * 1024);
9
10    void *p;
11
12    printf("%ld\n", (long int)p1 / 1024);
13    printf("%ld\n", (long int)p2 / 1024);
14    printf("%ld\n", (long int)p3 / 1024);
15
16    free(p2);
17    p = malloc(2 * 1024);
18
19    printf("\n%ld\n", (long int)p / 1024);
20
21    return 0;
22 }
    
```



## Mapa de memoria

### Ejemplo

## Stack Overflow

## Fragmentación

## arrays multiD

### Forma 1

### Forma 2

## Uso tras liberación

## valgrind

### Ejemplo

# Forma 1 (la mala)

## Dyn memory

Mapa de memoria

Ejemplo

Stack Overflow

Fragmentación

arrays multiD

Forma 1  
Forma 2

Uso tras liberación

valgrind  
Ejemplo

## Un malloc por cada fila del array (array de punteros)

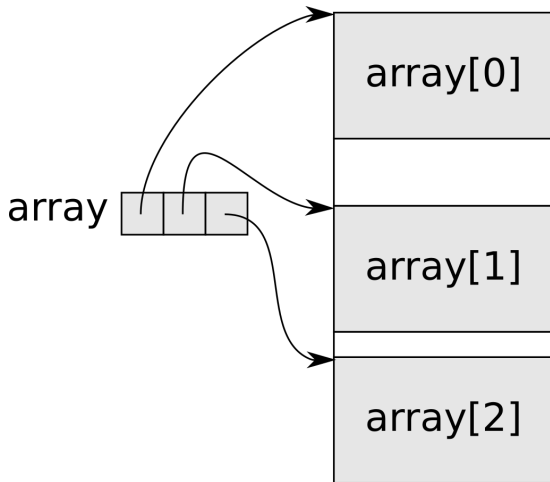
```
1 int main()
2 {
3     int i, j;
4     int **array;
5
6     /* Reservamos */
7     array = (int **)malloc(ROWS * sizeof(int *));
8     for (i = 0; i < ROWS; i++)
9         array[i] = (int *)malloc(COLS * sizeof(int)); // Falta comprobar
10
11    /* Inicializamos */
12    for (i = 0; i < ROWS; i++)
13        for (j = 0; j < ROWS; j++)
14            array[i][j] = i * 10 + j;
15
16    /* Imprimimos */
17    for (i = 0; i < ROWS; i++) {
18        for (j = 0; j < ROWS; j++) {
19            printf("%02d ", array[i][j]);
20        }
21        printf("\n");
22    }
23
24    /* Liberamos */
25    for (i = 0; i < ROWS; i++)
26        free(array[i]);
27
28    return 0;
29 }
```



# Forma 1 (la mala)

Dyn memory

No se garantiza continuidad entre los bloques reservados



Mapa de memoria

Ejemplo

Stack Overflow

Fragmentación

arrays multiD

Forma 1

Forma 2

Uso tras liberación

valgrind

Ejemplo

# Forma 2 (la buena)

## Dyn memory

## Mapa de memoria

### Ejemplo

## Stack Overflow

## Fragmentación

## arrays multiD

### Forma 1 Forma 2

## Uso tras liberación

## valgrind Ejemplo

```
1 #define ROWS 3
2 #define COLS 3
3
4 int main()
5 {
6     int i, j;
7     int *array;
8
9     /* Reservamos */
10    array = (int *)malloc(ROWS * COLS * sizeof(int));
11    if (!array) {
12        printf("Can't allocate the array\n");
13        return -1;
14    }
15
16    /* Inicializamos */
17    for (i = 0; i < ROWS; i++)
18        for (j = 0; j < COLS; j++)
19            *(array + i * COLS + j) = i * 10 + j;
20
21    /* Imprimimos */
22    for (i = 0; i < ROWS; i++) {
23        for (j = 0; j < COLS; j++) {
24            printf("%02d ", *(array + i * COLS + j));
25        }
26        printf("\n");
27    }
28
29    free(array); /* Liberamos */
30    return 0;
31 }
```

# Uso tras liberación

## Dyn memory

### Mapa de memoria

#### Ejemplo

### Stack Overflow

### Fragmentación

### arrays multiD

#### Forma 1

#### Forma 2

### Uso tras liberación

### valgrind

#### Ejemplo

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 struct person
6 {
7     char name[200];
8     char surname[200];
9     unsigned char age;
10 };
11
12 int main()
13 {
14     struct person *p;
15
16     p = (struct person *)malloc(sizeof(struct person));
17
18     strcpy(p->name, "Bob");
19     strcpy(p->surname, "Smith");
20     p->age = 20;
21
22     free(p);
23
24     printf("name    = %s\n", p->name);
25     printf("surname = %s\n", p->surname);
26     printf("age     = %d\n", p->age);
27
28     return 0;
29 }
```

# Depuración con valgrind

## Dyn memory

Mapa de memoria

Ejemplo

Stack Overflow

Fragmentación

arrays multiD

Forma 1

Forma 2

Uso tras liberación

valgrind

Ejemplo

- Valgrind es una herramienta que nos avisa de los errores cometidos en el manejo de memoria
- Un **leak** o fuga de memoria es un error de programación que hace que la memoria reservada no se libere cuándo ya no se usa. Un programa que no libere correctamente la memoria reservada puede acabar consumiendo toda la memoria del sistema y dejarlo inutilizado.
- Para que valgrind nos muestre más información podemos hacer dos cosas:
  - 1 Compilar con símbolos de depuración (`gcc -g`)
  - 2 Ejecutar valgrind con el flag `-leak-check=full`

# Ejemplo

## Dyn memory

Mapa de memoria  
Ejemplo

Stack  
Overflow

Fragmentación

arrays  
multiD

Forma 1  
Forma 2

Uso tras liberación

valgrind  
Ejemplo

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int *p = (int *)malloc(20);
7
8     if (!p)
9         return EXIT_FAILURE;
10
11     printf("La direccion de p es %p\n", p);
12
13     /* free(p) */
14
15     return 0;
16 }
```

# Ejemplo

## Dyn memory

### Mapa de memoria

#### Ejemplo

### Stack Overflow

### Fragmentación

### arrays multiD

#### Forma 1

#### Forma 2

### Uso tras liberación

### valgrind

#### Ejemplo

```
gcc -g valgrind.c -o valg_ej
valgrind --leak-check=full valg_ej
```

```
1 Memcheck, a memory error detector
2 ....
3 Command: valg_ej
4
5 La direccion de p es 0x51d7040
6
7 HEAP SUMMARY:
8     in use at exit: 20 bytes in 1 blocks
9     total heap usage: 1 allocs, 0 frees, 20 bytes allocated
10
11 20 bytes in 1 blocks are definitely lost in loss record 1 of 1
12   at 0x4C2ABD0: malloc (in /usr/lib/...)
13   by 0x4004F7: main (valgrind.c:6)
14
15 LEAK SUMMARY:
16     definitely lost: 20 bytes in 1 blocks
17     indirectly lost: 0 bytes in 0 blocks
18     possibly lost: 0 bytes in 0 blocks
19     still reachable: 0 bytes in 0 blocks
20     suppressed: 0 bytes in 0 blocks
21
22 For counts of detected and suppressed errors, rerun with: -v
23 ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

## Objetos

Qué es?

Ejemplo

Ocultación

flags

Reserva

Setters

Getters

Print

# Objetos

## Tema 12

## Objetos

Qué es?

Ejemplo

Ocultación

flags

Reserva

Setters

Getters

Print

**42** ¿Qué es un objeto?

**43** Ejemplo de objeto en C

- Ocultación
- Flags
- Reserva
- Setters
- Getters
- Print



# ¿Qué es un objeto?

## Objetos

### Qué es?

### Ejemplo

Ocultación  
flags  
Reserva  
Setters  
Getters  
Print

- Un objeto es una *entidad* con:
  - Un **estado**: datos que guarda
  - Un **comportamiento**: métodos que interactúan con ese estado.
- Un objeto es una **instancia** de una **clase** específica
- Un clase puede **heredar** propiedades de otras

# Ejemplo de objeto en C

## Objetos

Qué es?

## Ejemplo

Ocultación

flags

Reserva

Setters

Getters

Print

Ejemplo de objeto  
persona

## Objetos

Qué es?

Ejemplo

**Ocultación**

flags

Reserva

Setters

Getters

Print

## person.h

```
1 #ifndef PERSON_H
2 #define _PERSON_H_
3
4 struct person;
5
6 #endif
```

## person.c

```
1 #include "person.h"
2
3 #define DNI_LEN 9
4
5 struct person {
6     char *name;
7     char dni[DNI_LEN];
8     int age;
9
10    uint32_t flags;
11 };
12
13 enum person_attr {
14     PERSON_NAME,
15     PERSON_DNI,
16     PERSON_AGE,
17 };
```

## main.c

```
1 #include "person.h"
2
3 int main()
4 {
5     struct person *p;
6
7     return EXIT_SUCCESS;
8 }
```

## Objetos

Qué es?

Ejemplo

Ocultación

flags

Reserva

Setters

Getters

Print

person.c

```
1 #define ATTR_SET(flags , attr)    (flags) |= (1 << (attr))  
2 #define ATTR_IS_SET(flags , attr) ((flags) & (1 << (attr)))
```

Necesitamos saber si un atributo ha sido o no establecido para:

- Saber si el valor que guarda o no es válido
- Saber si se ha reservado memoria y hay que liberarla
- Imprimir o no los atributos
- ...

## Objetos

Qué es?

Ejemplo

Ocultación

flags

**Reserva**

Setters

Getters

Print

## person.c

```
1 struct person *person_alloc()
2 {
3     struct person *p;
4
5     p = (struct person *)malloc(sizeof(struct person));
6     p->flags = 0;
7
8     return p;
9 }
10
11 void person_free(struct person *p)
12 {
13     if (ATTR_IS_SET(p->flags, PERSON_NAME))
14         free(p->name);
15     free(p);
16 }
```

## main.c

```
1 int main()
2 {
3     struct person *p;
4
5     p = person_alloc();
6     person_free(p);
7
8     return EXIT_SUCCESS;
9 }
```

# Setters

## Objetos

Qué es?

Ejemplo

Ocultación

flags

Reserva

**Setters**

Getters

Print

```
1 void person_set_name(struct person *p, const char *name)
2 {
3     if (ATTR_IS_SET(p->flags, PERSON_NAME))
4         free(p->name);
5
6     p->name = strdup(name);
7     ATTR_SET(p->flags, PERSON_NAME);
8 }
9
10 int person_set_dni(struct person *p, const char *dni)
11 {
12     if (strlen(dni) != DNI_LEN)
13         return 0;
14
15     strcpy(p->dni, dni);
16     ATTR_SET(p->flags, PERSON_DNI);
17
18     return 1;
19 }
20
21 int person_set_age(struct person *p, int age)
22 {
23     if (age < 0 || age > 150)
24         return 0;
25
26     p->age = age;
27     ATTR_SET(p->flags, PERSON_AGE);
28
29     return 1;
30 }
```

# Getters

## Objetos

Qué es?

Ejemplo

Ocultación  
flags

Reserva

Settings

**Getters**

Print

```
1  const char *person_get_name(const struct person *p)
2  {
3      if (ATTR_IS_SET(p->flags , PERSON_NAME))
4          return p->name;
5      else
6          return NULL;
7  }
8
9  const char *person_get_dni(const struct person *p)
10 {
11     if (ATTR_IS_SET(p->flags , PERSON_DNI))
12         return p->dni;
13     else
14         return NULL;
15 }
16
17 int person_get_age(const struct person *p)
18 {
19     if (ATTR_IS_SET(p->flags , PERSON_AGE))
20         return p->age;
21     else
22         return -1;
23 }
```

## Objetos

Qué es?

Ejemplo

Ocultación

flags

Reserva

Setters

Getters

Print

```
1 void person_print(const struct person *p)
2 {
3     printf("Person {");
4
5     if (ATTR_IS_SET(p->flags, PERSON_NAME))
6         printf("\n\tname = \"%s\"", p->name);
7
8     if (ATTR_IS_SET(p->flags, PERSON_DNI))
9         printf("\n\tdni = \"%s\"", p->dni);
10
11    if (ATTR_IS_SET(p->flags, PERSON_AGE))
12        printf("\n\tage = %d", p->age);
13
14    printf("\n}\n");
15 }
```



**main() args**

argc y argv

Ejemplo

getopt

getopt short

getopt long

# Argumentos de main()

## Tema 13

[main\(\) args](#)

[argc y argv](#)

[Ejemplo](#)

[getopt](#)

[getopt short](#)

[getopt long](#)

**44** [argc y argv](#)  
■ [Ejemplo](#)

**45** [getopt](#)

**46** [getopt short](#)

**47** [getopt long](#)

## argc y argv

main() args

```
int main(int argc, char *argv [])
```

argc y argv

Ejemplo

getopt

getopt short

getopt long

- La función main admite dos parámetros que están relacionados con los argumentos que pasamos a nuestro programa al ejecutarlo:
  - **argc**: Número de argumentos pasados
  - **argv**: Vector de argumentos
- Cada argumento es una cadena de texto
- Siempre se pasa al menos un argumento con el nombre del programa (incluida la ruta relativa de llamada)

# Ejemplo

main() args

código:

```
1 int main(int argc, char *argv[])
2 {
3     int i;
4
5     printf("Argumento i = <argumento>\n");
6     for (i = 0; i < argc; i++)
7         printf("%d = %s\n", i, argv[i]);
8
9     return 0;
10 }
```

salida:

```
#> ./args_example foo bar
Argumento i = <argumento>
0 = ./args_example
1 = foo
2 = bar
```

argc y argv  
Ejemplo

getopt

getopt short

getopt long

[main\(\) args](#)

[argc y argv](#)

[Ejemplo](#)

[getopt](#)

[getopt short](#)

[getopt long](#)

- Es una utilidad de GNU C Library
- Sigue el estándar POSIX sobre argumentos
- Permite analizar los argumentos fácilmente

# getopt short

main() args

> gopts -a -b 3 -c hola

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4
5 int main(int argc, char **argv)
6 {
7     int aflag;
8     int bvalue;
9     char *cvalue;
10    int c;
11
12    while ((c = getopt(argc, argv, "ab:c:")) != -1) {
13        switch (c)
14        {
15            case 'a':
16                aflag = 1;
17                break;
18            case 'b':
19                bvalue = strtol(optarg, NULL, 0);
20                break;
21            case 'c':
22                cvalue = optarg;
23                break;
24            default:
25                return EXIT_FAILURE;
26        }
27    }
```

argc y argv  
Ejemplo

getopt

getopt short

getopt long

# getopt short

main() args

argc y argv

Ejemplo

getopt

getopt short

getopt long

```
30     printf("bvalue = %d\n", bvalue);
31     printf("cvalue = \"%s\"\n", cvalue);
32
33     for (int i = optind; i < argc; i++)
34         printf("Argumento desconocido \"%s\"\n", argv[i]);
35
36     return 0;
37 }
```

```
> gopts -a -b 3 -c hola
aflag = 1
bvalue = 3
cvalue = "hola"
```

# getopt long

main() args

```
> goptl -u -w 10 -h 20 --random=52432
```

argc y argv

Ejemplo

getopt

getopt short

getopt long

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4 #include <getopt.h>
5
6 int main(int argc, char **argv)
7 {
8     int usage = false;
9     size_t width = 0, height = 0;
10    bool random = false;
11    int rseed = 0;
12
13    int option_index = 0;
14    int c;
15
16    static struct option long_options [] =
17    {
18        {"usage", no_argument, NULL, 'u'},
19        {"width", required_argument, NULL, 'w'},
20        {"height", required_argument, NULL, 'h'},
21        {"random", optional_argument, NULL, 'r'},
22        {0, 0, 0, 0}
23    };
```



# getopt long

main() args

argc y argv

Ejemplo

getopt

getopt short

getopt long

```
25 while ((c = getopt_long(argc, argv, "uw:h:r::", long_options,
26     &option_index)) != -1) {
27     switch (c) {
28     case 'u':
29         usage = true;
30         break;
31     case 'w':
32         width = strtol(optarg, NULL, 0);
33         break;
34     case 'h':
35         height = strtol(optarg, NULL, 0);
36         break;
37     case 'r':
38         random = true;
39         if (optarg)
40             rseed = strtol(optarg, NULL, 0);
41         break;
42     case '?':
43         /* getopt_long imprime un mensaje de error*/
44         break;
45     default:
46         printf("Error\n");
47         exit(EXIT_FAILURE);
48     }
49 }
```

# getopt long

main() args

argc y argv  
Ejemplo

getopt

getopt short

getopt long

```
51  printf("usage = %s\n", usage? "TRUE" : "FALSE");
52  printf("width = %lu\n", width);
53  printf("height = %lu\n", height);
54  printf("random = %s\n", random? "TRUE" : "FALSE");
55  printf("rseed = %d\n", rseed);
56
57  for (int i = optind; i < argc; i++)
58      printf("Argumento desconocido: \"%s\"\n", argv[i]);
59
60  exit(0);
61 }
```

```
> goptl -u -w 10 -h 20 --random=52432
usage = TRUE
width = 10
height = 20
random = TRUE
rseed = 52432
```

## Depuración

Introducción

GDB

Cómo utilizar  
GDB

Comandos

Inicio  
Breakpoints  
Paso a paso  
Estado del  
programa  
Otros  
comandos  
útiles

# Depuración

## Tema 14

## Depuración

Introducción

GDB

Cómo utilizar  
GDB

Comandos

Inicio  
Breakpoints  
Paso a paso  
Estado del  
programa  
Otros  
comandos  
útiles

**48** Introducción

**49** GDB

**50** Cómo utilizar GDB

**51** Comandos

- Inicio
- Breakpoints
- Paso a paso
- Estado del programa
- Otros comandos útiles

# Introducción

## Depuración

### Introducción

### GDB

### Cómo utilizar GDB

### Comandos

Inicio  
Breakpoints  
Paso a paso  
Estado del  
programa  
Otros  
comandos  
útiles

- **¿Qué es depurar (debugging)?**: El proceso por el cual buscas y/o eliminas fallos (bugs) de tu programa
- **¿Qué es un depurador (debugger)?**: Una herramienta que facilita la depuración al permitir detener tu programa en cualquier instante y mirar que está pasando
- **¿Por qué es importante usarlo?**: . El método tradicional de los `printf` de depuración está bien, pero se queda corto en cuanto el programa crece en complejidad. En este punto, el depurador se vuelve la manera más eficiente y fácil de depurarlo.

## Depuración

### Introducción

### GDB

### Cómo utilizar GDB

### Comandos

Inicio  
Breakpoints  
Paso a paso  
Estado del  
programa  
Otros  
comandos  
útiles

- Depurador del proyecto GNU
- Un proyecto muy maduro (30 años)
- Repleto de utilidades para depurar
- Capaz de depurar varios lenguajes (C, C++, Ada, Objective-C, Free Pascal, Fortran, Java ...)
- Soporta muchas arquitecturas (x86, x86-64, ARM, AVR, ...)
- Sistema cliente-servidor que permite depuración remota
- Base de muchos depuradores gráficos actuales (CodeBlocs, Eclipse, NetBeans, VisualStudio, QtCreator, ...)

# Cómo utilizar GDB

## Depuración

### Introducción

### GDB

### Cómo utilizar GDB

### Comandos

Inicio  
Breakpoints  
Paso a paso  
Estado del programa  
Otros comandos útiles

- 1 Compilar con símbolos de depuración `gcc -g main.c -o miprog (-ggdb para símbolos específicos de gdb)`
- 2 Arrancar nuestro programa con `gdb miprog`
- 3 Utilizar los comandos de `gdb` para depurar

Depuración

Introducción

GDB

Cómo utilizar  
GDB

**Comandos**

Inicio  
Breakpoints  
Paso a paso  
Estado del  
programa  
Otros  
comandos  
útiles

# COMANDOS



## Depuración

### Introducción

### GDB

### Cómo utilizar GDB

### Comandos

#### Inicio

#### Breakpoints

#### Paso a paso

#### Estado del programa

#### Otros comandos

#### útiles

El programa se inicia con el comando `run`. Si nuestro programa recibe parámetros, se los pasamos a `run`:

```
(gdb) run -w 10 -h 15
```

# Breakpoints

## Depuración

### Introducción

### GDB

### Cómo utilizar GDB

### Comandos

#### Inicio

#### **Breakpoints**

#### Paso a paso

#### Estado del programa

#### Otros comandos útiles

Un **breakpoint** define el momento en el que se va a detener nuestro programa para permitirnos examinarlo.

Podemos elegir que se detenga cuando:

- Alcance cierta línea de código:  
`(gdb) break main.c:15`
- Entre en una función determinada:  
`(gdb) break world_alloc`
- Se modifique cierta variable:  
`(gdb) watch contador`

## Otros comandos:

- Listar los breakpoints (y watchpoints):  
`(gdb) info breakpoints`
- Elimiar breakpoint:  
`(gdb) delete <num que sale al listar>`
- Elimiar todos los breakpoint:  
`(gdb) delete breakpoints`

## Depuración

### Introducción

### GDB

### Cómo utilizar GDB

### Comandos

Inicio  
Breakpoints  
**Paso a paso**  
Estado del  
programa  
Otros  
comandos  
útiles

Tenemos varios comandos para realizar una ejecución “paso a paso” de nuestro programa:

- Ejecutar hasta el siguiente breakpoint:  
(gdb) `continue`
- Ejecutar una línea (sin entrar en funciones):  
(gdb) `next`
- Ejecutar una línea (entrado en funciones):  
(gdb) `step`
- Salir de la función actual:  
(gdb) `finish`

# Estado del programa

## Depuración

### Introducción

### GDB

### Cómo utilizar GDB

### Comandos

- Inicio
- Breakpoints
- Paso a paso
- Estado del programa**
- Otros comandos útiles

- Ver una variable:  
`(gdb) print contador`
- Ver una estructura:  
`(gdb) print *world`
- Ver el resultado de una expresión:  
`(gdb) print size_x * size_y`
- Ver el resultado de una función:  
`(gdb) print world_get_cell(w, 1, 2)`
- Ver las variables locales:  
`(gdb) info locals`
- Ver algunas líneas de código de alrededor:  
`(gdb) list`
- Ver la pila de llamadas:  
`(gdb) backtrace`

## Otros comandos útiles

### Depuración

### Introducción

### GDB

### Cómo utilizar GDB

### Comandos

### Inicio Breakpoints Paso a paso Estado del programa Otros comandos útiles

- Ver clases de comandos:  
`(gdb) help`
- Ver comandos de un clase (por ejemplo, breakpoints):  
`(gdb) help breakpoints`
- Ver ayuda sobre un comando (por ejemplo, break):  
`(gdb) help break`
- Para salir:  
`(gdb) quit`
- Puedes abreviar un comando siempre que no sea ambiguo:  
`(gdb) help n == (gdb) help next`
- Para ejecutar el comando anterior pulsar ENTER
- Para autocompletar (comandos, nombres de funciones y variables, etc) pulsar TAB
- Chuleta:

[http://lab46.corning-cc.edu/\\_media/opus/fall2014/mgardne8/70120637.png](http://lab46.corning-cc.edu/_media/opus/fall2014/mgardne8/70120637.png)

fopen  
fwrite  
fread  
fputs  
fgets  
fprintf  
fscanf

# Entrada/Salida

## Tema 15

## Entrada/Salida

### Streams

### Funciones

fopen  
fwrite  
fread  
fputs  
fgets  
fprintf  
fscanf

### Funciones peligrosas

**52** ¿Qué es un stream?

**53** Funciones básicas

- fopen
- fwrite
- fread
- fputs
- fgets
- fprintf
- fscanf

**54** Funciones peligrosas



# ¿Qué es un stream?

## Entrada/Salida

## Streams

## Funciones

fopen  
fwrite  
fread  
fputs  
fgets  
fprintf  
fscanf

## Funciones peligrosas

**Flujo de bytes** de longitud indeterminada, al que se accede de forma **secuencial**.

En un *stream*:

- Al **leer** uno o más bytes, en la próxima lectura obtendremos los siguientes.
- Al **escribir** uno o más bytes, en la próxima escritura los añadiremos a continuación.

Streams estándar:

- **stdout**: *Salida estándar* (normalmente por consola)
- **stdin**: *Entrada estándar* (normalmente por teclado)
- **stderr**: *Salida estándar de errores* (normalmente por consola)

# Funciones básicas

## Entrada/Salida

## Streams

## Funciones

fopen  
fwrite  
fread  
fputs  
fgets  
fprintf  
fscanf

## Funciones peligrosas

- **Apertura y cierre de ficheros:**
  - `FILE *f = fopen("file.txt", "r");`
  - `fclose(f);`
- **Lectura/Escritura en crudo:**
  - `size_t read = fread(buf, sizeof(char), bufsize, f);`
  - `size_t written = fwrite(buf, sizeof(char), bufsize, f);`
- **Lectura/Escritura sin formato:**
  - `fputs("Hola Mundo", f);`
  - `fgets(buf, bufsize, f);`
- **Lectura/Escritura con formato:**
  - `fprintf(f, "Hola", name);`
  - `fscanf(f, "%10s %d", str, &i);`

Para más información ver:

<http://es.cppreference.com/w/c/io>

# fopen

## Entrada/Salida

### Streams

### Funciones

fopen  
fwrite  
fread  
fputs  
fgets  
fprintf  
fscanf

### Funciones peligrosas

```
FILE *fopen(const char *fname, const char *mode);
int fclose(FILE *stream);
```

## Modos:

modo	significado	si ya existe	si no existe
"r"	<b>Abre para leer</b>	lee desde el <b>principio</b>	error
"w"	<b>Crea para escribir</b>	<b>descarta</b> el contenido	lo crea
"a"	<b>Crea para añadir</b>	escribe al <b>final</b>	lo crea
"r+"	<b>Abre para leer/escribir</b>	lee/escribe desde el <b>principio</b>	error
"w+"	<b>Crea para leer/escribir</b>	<b>descarta</b> el contenido	lo crea
"a+"	<b>Crea para leer/añadir</b>	lee*/escribe desde el <b>final</b>	lo crea

# fwrite

Entrada/Salida `size_t fwrite(const void *buf, size_t size, size_t count, FILE *strm);`

Streams

Funciones

fopen  
fwrite  
fread  
fputs  
fgets  
fprintf  
fscanf

Funciones  
peligrosas

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define FNAME "file"
6
7 int main() {
8     FILE *f;
9     char buf[] = "Hola Mundo";
10
11     f = fopen(FNAME, "w+");
12     if (!f) {
13         perror("Error al abrir");
14         exit(EXIT_FAILURE);
15     }
16
17     fwrite(buf, sizeof(char), strlen(buf), f);
18     if (ferror(f)) {
19         perror("Error al escribir");
20         exit(EXIT_FAILURE);
21     }
22
23     fclose(f);
24     return 0;
25 }
```

# fread

Entrada/Salida `size_t fread(void *buf, size_t size, size_t cnt, FILE *strm);`

## Streams

## Funciones

fopen  
fwrite  
**fread**  
fputs  
fgets  
fprintf  
fscanf

## Funciones peligrosas

```
1 #define FNAME "file"
2 #define BUFSIZE 256
3
4 int main() {
5     FILE *f;
6     char buf[BUFSIZE];
7     int read;
8
9     f = fopen(FNAME, "r");
10    if (!f) {
11        perror("No se ha podido abrir el fichero");
12        exit(EXIT_FAILURE);
13    }
14
15    read = fread(buf, sizeof(char), BUFSIZE - 1, f);
16    if (ferror(f)) {
17        perror("Error al leer");
18        exit(EXIT_FAILURE);
19    }
20
21    buf[read] = '\0';
22    printf("%s", buf);
23
24    fclose(f);
25    return 0;
26 }
```

# fputs

Entrada/Salida `int fputs(const char *str, FILE *strm);`

## Streams

## Funciones

fopen  
fwrite  
fread  
fputs  
fgets  
fprintf  
fscanf

## Funciones peligrosas

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define FNAME "file"
6
7 int main()
8 {
9     FILE *f;
10    char buf[] = "Hola Mundo";
11
12    f = fopen(FNAME, "w+");
13    if (!f) {
14        perror("Error al abrir");
15        exit(EXIT_FAILURE);
16    }
17
18    fputs(buf, f);
19    if (ferror(f)) {
20        perror("Error al escribir");
21        exit(EXIT_FAILURE);
22    }
23
24    fclose(f);
25    return 0;
26 }
```

# fgets

Entrada/Salida `char * fgets(char *buf, int bufsize, FILE *strm);`

Streams

Funciones

fopen  
fwrite  
fread  
fputs  
fgets  
fprintf  
fscanf

Funciones  
peligrosas

```
1 #define FNAME "file"
2 #define BUFSIZE 256
3
4 int main()
5 {
6     FILE *f;
7     char buf[BUFSIZE];
8
9     f = fopen(FNAME, "r");
10    if (!f) {
11        perror("Error al abrir");
12        exit(EXIT_FAILURE);
13    }
14
15    int i = 0;
16    while (fgets(buf, BUFSIZE, f))
17        printf("%d: \"%s\\n\"", i++, buf);
18    if (ferror(f)) {
19        perror("Error al leer");
20        exit(EXIT_FAILURE);
21    }
22
23    fclose(f);
24    return 0;
25 }
```

# fprintf

Entrada/Salida `int fprintf (FILE *strm, const char *frmt, ...);`

Streams

Funciones

fopen  
fwrite  
fread  
fputs  
fgets  
fprintf  
fscanf

Funciones  
peligrosas

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define FNAME "file"
6
7 int main()
8 {
9     FILE *f;
10
11     f = fopen(FNAME, "w+");
12     if (!f) {
13         perror("Error al abrir");
14         exit(EXIT_FAILURE);
15     }
16
17     fprintf(f, "Hola Mundo\n 2 + 2 = %d", 2 + 2);
18     if (ferror(f)) {
19         perror("Error al escribir");
20         exit(EXIT_FAILURE);
21     }
22
23     fclose(f);
24     return 0;
25 }
```



# fscanf

Entrada/Salida `int fscanf(FILE *strm, const char *frmt, ...);`

## Streams

## Funciones

fopen  
fwrite  
fread  
fputs  
fgets  
fprintf  
fscanf

## Funciones peligrosas

```
1 int main()
2 {
3     FILE *f;
4     char str1[10], str2[10];
5     int a, b, c, ret;
6
7     f = fopen(FNAME, "r");
8     if (!f) {
9         perror("Error al abrir");
10        exit(EXIT_FAILURE);
11    }
12
13    while ((ret = fscanf(f, "%10s %10s %d + %d = %d",
14                        str1, str2, &a, &b, &c)) == 5) {
15        printf("%s %s\n %d + %d = %d\n", str1, str2, a, b, c);
16    }
17    if (ferror(f)) {
18        perror("Error al leer");
19        exit(EXIT_FAILURE);
20    }
21    else if (ret != EOF) {
22        fprintf(stderr, "Error al analizar: %d/%d\n", ret, 5);
23    }
24
25    fclose(f);
26    return 0;
27 }
```

# Funciones peligrosas

## Entrada/Salida

### Streams

### Funciones

fopen  
fwrite  
fread  
fputs  
fgets  
fprintf  
fscanf

### Funciones peligrosas

Todas las funciones que lean un número indefinido de bytes sin comprobar el tamaño del buffer de destino.

- `scanf` y sus variantes si no especificamos el tamaño al escanear una cadena: `"%S"`
- `gets`, que lee una cadena de `stdin` sin posibilidad de especificar ningún límite para su tamaño. Eliminada en el estándar de 2011.

# Punteros a funciones

## Tema 16

## Punteros a funciones

Introducción

Ejemplo 1

Ejemplo 2

Ejemplo 3

**55** Introducción

**56** Ejemplo 1: Declaración, asignación y llamada

**57** Ejemplo 2: Comparator

**58** Ejemplo 3: Array

```
const char *(*ptr_2_func)(int, char *);
```

- Guardan la dirección de una función
- Muy útiles en ciertas situaciones
- Sintaxis complicada (Consultar <http://cdecl.org/>)

# Ejemplo 1: Declaración, asignación y llamada

## Punteros a funciones

### Introducción

### Ejemplo 1

### Ejemplo 2

### Ejemplo 3

```
1 int mul(int i)
2 {
3     return i*2;
4 }
5
6 int div(int i)
7 {
8     return i/2;
9 }
10
11 int main()
12 {
13     int (*pf)(int);
14
15     pf = mul;
16     printf("%d\n", pf(10));
17
18     pf = div;
19     printf("%d\n", pf(10));
20
21     return 0;
22 }
```

## Ejemplo 2: Comparador

### Punteros a funciones

#### Introducción

#### Ejemplo 1

#### Ejemplo 2

#### Ejemplo 3

```
1 struct obj {  
2     int a;  
3     int b;  
4 };
```

```
1 int main()  
2 {  
3     struct obj a = {1, 2};  
4     struct obj b = {2, 1};  
5  
6     obj_print(max(&a, &b, cmp_a));  
7     obj_print(max(&a, &b, cmp_b));  
8  
9     return 0;  
10 }
```

## Ejemplo 2: Comparator

### Punteros a funciones

Introducción

Ejemplo 1

Ejemplo 2

Ejemplo 3

```
1 int cmp_a(struct obj *a, struct obj *b)
2 {
3     if (a->a > b->a)
4         return 1;
5     else if (a->a < b->a)
6         return -1;
7     else
8         return 0;
9 }
10
11 int cmp_b(struct obj *a, struct obj *b)
12 {
13     if (a->b > b->b)
14         return 1;
15     else if (a->b < b->b)
16         return -1;
17     else
18         return 0;
19 }
```



## Ejemplo 2: Comparador

### Punteros a funciones

#### Introducción

#### Ejemplo 1

#### Ejemplo 2

#### Ejemplo 3

```
1 typedef int (*ptrf_cmp_t)(struct obj *a, struct obj *b);  
2  
3 const struct obj *max(struct obj *a, struct obj *b,  
4                       ptrf_cmp_t cmpf)  
5 {  
6     if (cmpf(a, b) > 0)  
7         return a;  
8     else  
9         return b;  
10 }
```

## Ejemplo 2: Comparator

### Punteros a funciones

Introducción

Ejemplo 1

Ejemplo 2

Ejemplo 3

```
1 int main()  
2 {  
3     struct obj a = {1, 2};  
4     struct obj b = {2, 1};  
5  
6     obj_print(max(&a, &b, cmp_a));  
7     obj_print(max(&a, &b, cmp_b));  
8  
9     return 0;  
10 }
```

```
> ./comparator
```

```
{2, 1}
```

```
{1, 2}
```

## Ejemplo 3: Array

### Punteros a funciones

### Introducción

### Ejemplo 1

### Ejemplo 2

### Ejemplo 3

```
1 #include <stdio.h>
2
3 void es() { printf("Hola\n"); }
4 void en() { printf("Hello\n"); }
5 void fr() { printf("Salut\n"); }
6
7 int main()
8 {
9     printf("Chose an option:\n"
10           "\t0- spanish\n"
11           "\t1- english\n"
12           "\t2- french\n"
13           );
14
15     switch (getchar()) {
16         case '0': es(); break;
17         case '1': en(); break;
18         case '2': fr(); break;
19         default : es();
20     };
21
22     return 0;
23 }
```

## Ejemplo 3: Array

### Punteros a funciones

Introducción

Ejemplo 1

Ejemplo 2

Ejemplo 3

```
1 #include <stdio.h>
2
3 void es() { printf("Hola\n"); }
4 void en() { printf("Hello\n"); }
5 void fr() { printf("Salut\n"); }
6
7 int main()
8 {
9     void (*salute[])( ) = {es, en, fr};
10
11     printf("Chose an option:\n"
12           "\t0- spanish\n"
13           "\t1- english\n"
14           "\t2- french\n"
15           );
16
17     salute[getchar() - '0']();
18
19     return 0;
20 }
```

## Objetos (II): Herencia

Ejemplo

El objetivo

La forma

Herencia

Estructura de  
ficheros

Encapsulación

Polimorfismo

Punteros a  
funciones

Métodos

Constructor  
vehicle

Constructor  
car

Destruyores

# Objetos (II): Herencia

## Tema 17

## Objetos (II): Herencia

Ejemplo

El objetivo

La forma

Herencia

Estructura de  
ficheros

Encapsulación

Polimorfismo

Punteros a  
funciones

Métodos

Constructor  
vehicle

Constructor  
car

Destruyores

**59** Ejemplo de herencia

**60** El objetivo

**61** La forma

**62** Herencia

- Estructura de ficheros
- Encapsulación

**63** Polimorfismo

- Punteros a funciones
- Métodos
- Constructor `vehicle`
- Constructor `car`
- Destruyores

# Ejemplo de herencia

## Objetos (II): Herencia

### Ejemplo

### El objetivo

### La forma

### Herencia

### Estructura de ficheros

### Encapsulación

### Polimorfismo

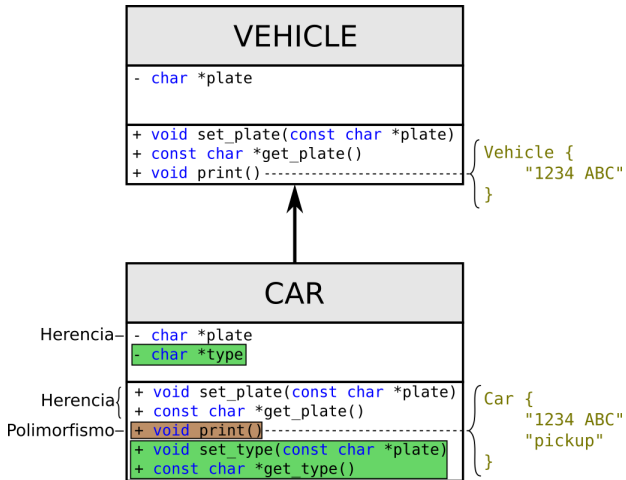
### Punteros a funciones

### Métodos

### Constructor vehicle

### Constructor car

### Destructores



# El objetivo

## Objetos (II): Herencia

### Ejemplo

### El objetivo

### La forma

### Herencia

### Estructura de archivos

### Encapsulación

### Polimorfismo

### Punteros a funciones

### Métodos

### Constructor vehicle

### Constructor car

### Destruyores

```
1 #include "vehicle.h"
2 #include "car.h"
3
4 int main()
5 {
6     struct vehicle *v;
7     struct vehicle *c;
8
9     v = vehicle_alloc("1234 ABC");
10    c = (struct vehicle *)car_alloc("4321 CBA", "pickup");
11
12    vehicle_print(v);
13    vehicle_print(c);
14
15    return 0;
16};
```

```
> ./vehicles
Vehicle {
  plate = "1234 ABC"
}
Car {
  plate = "4321 CBA"
  type = "pickup"
}
```



## Objetos (II): Herencia

### Ejemplo

### El objetivo

### La forma

### Herencia

### Estructura de ficheros

### Encapsulación

### Polimorfismo

### Punteros a funciones

### Métodos

### Constructor vehicle

### Constructor car

### Destruyores

- **Encapsular** la estructura del objeto padre en la del objeto hijo para implementar la **herencia**
- Utilizar **punteros a funciones** para implementar el **polimorfismo**: Un puntero guardará una función u otra dependiendo del tipo de objeto

## Objetos (II): Herencia

Ejemplo

El objetivo

La forma

**Herencia**

Estructura de  
ficheros

Encapsulación

Polimorfismo

Punteros a  
funciones

Métodos

Constructor  
vehicle

Constructor  
car

Destruyores

# HERENCIA

# Estructura de ficheros

## Objetos (II): Herencia

Ejemplo

El objetivo

La forma

Herencia

**Estructura de  
ficheros**

Encapsulación

Polimorfismo

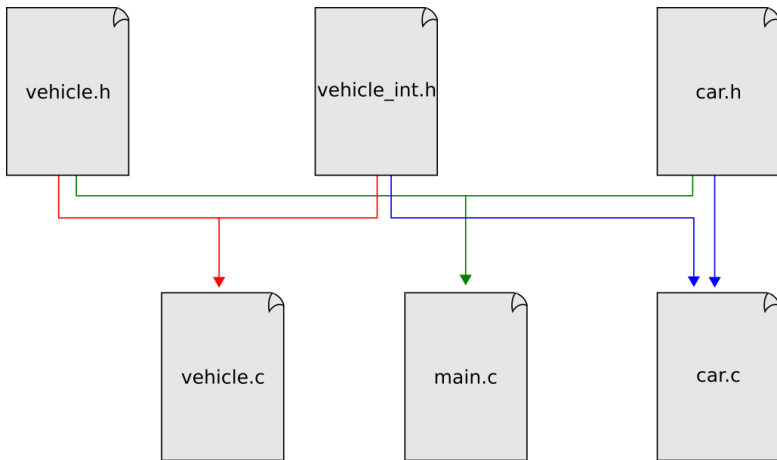
Punteros a  
funciones

Métodos

Constructor  
vehicle

Constructor  
car

Destructores



# Encapsulación

## Objetos (II): Herencia

### Ejemplo

### El objetivo

### La forma

### Herencia

### Estructura de archivos

### Encapsulación

### Polimorfismo

### Punteros a funciones

### Métodos

### Constructor vehicle

### Constructor car

### Destruyores

## vehicle\_int.h

```
1 #include <stdint.h>
2
3 struct vehicle
4 {
5     char *plate;
6     uint32_t flags;
7 };
8
9 enum vehicle_attr {
10     VEHICLE_PLATE,
11 };
```

## car.c

```
1 #include "vehicle_int.h"
2
3 struct car {
4     struct vehicle super; /* Siempre el primero */
5     char *type;
6     uint32_t flags;
7 };
8
9 enum car_attr {
10     CAR_TYPE,
11 };
```

# Encapsulación

## Objetos (II): Herencia

Ejemplo

El objetivo

La forma

Herencia

Estructura de  
ficheros

**Encapsulación**

Polimorfismo

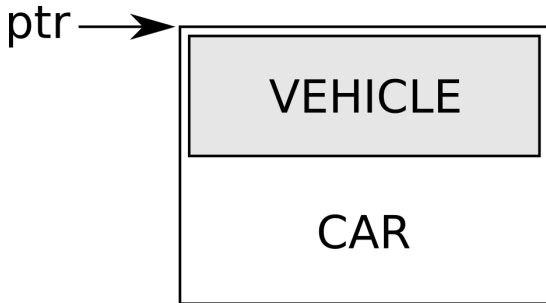
Punteros a  
funciones

Métodos

Constructor  
vehicle

Constructor  
car

Destruyores



```
struct vehicle *v = (struct vehicle *)car_alloc(...);
```

## Objetos (II): Herencia

Ejemplo

El objetivo

La forma

Herencia

Estructura de  
ficheros

Encapsulación

**Polimorfismo**

Punteros a  
funciones

Métodos

Constructor  
vehicle

Constructor  
car

Destruyores

# POLIMORFISMO

# Punteros a funciones

## Objetos (II): Herencia

Ejemplo

El objetivo

La forma

Herencia

Estructura de  
archivos

Encapsulación

Polimorfismo

**Punteros a  
funciones**

Métodos

Constructor  
vehicle

Constructor  
car

Destruyores

```
1 #ifndef _VEHICLE_INT_
2 #define _VEHICLE_INT_
3
4 #include <stdint.h>
5
6 struct vehicle
7 {
8     char *plate;
9
10    void (*set_plate)(struct vehicle *, const char *);
11    const char *(*get_plate)(const struct vehicle *);
12    void (*print)(const struct vehicle *);
13
14    uint32_t flags;
15 };
16
17 enum vehicle_attr {
18     VEHICLE_PLATE,
19 };
20
21 #endif
```

```
1 static void default_print(const struct vehicle *v)
2 {
3     printf("Vehicle {");
4
5     if (ATTR_IS_SET(v->flags, VEHICLE_PLATE))
6         printf("\n\tplate = \"%s\"", v->plate);
7
8     printf("\n}\n");
9 }
10
11 void vehicle_print(const struct vehicle *v)
12 {
13     v->print(v);
14 }
```

Nota: Para crear un método/clase **abstract@** basta con no crear el/los método(s) por defecto



# Constructor vehicle

## Objetos (II): Herencia

Ejemplo

El objetivo

La forma

Herencia

Estructura de  
archivos

Encapsulación

Polimorfismo

Punteros a  
funciones

Métodos

**Constructor**  
vehicle

Constructor  
car

Destruyores

```
1 void vehicle_init(struct vehicle *v, const char *plate)
2 {
3     v->flags = 0;
4
5     v->set_plate = default_set_plate;
6     v->get_plate = default_get_plate;
7     v->print = default_print;
8
9     default_set_plate(v, plate);
10 }
11
12 struct vehicle *vehicle_alloc(const char *plate)
13 {
14     struct vehicle *v;
15
16     v = (struct vehicle *)malloc(sizeof(struct vehicle));
17     if (!v)
18         return NULL;
19
20     vehicle_init(v, plate);
21
22     return v;
23 }
```

# Constructor car

## Objetos (II): Herencia

Ejemplo

El objetivo

La forma

Herencia

Estructura de  
ficheros

Encapsulación

Polimorfismo

Punteros a  
funciones

Métodos

Constructor  
vehicle

**Constructor  
car**

Destructores

```
1 struct car *car_alloc(const char *plate, const char *type
2 )
3 {
4     struct car *c;
5     c = (struct car *)malloc(sizeof(struct car));
6     if (!c)
7         return NULL;
8
9     vehicle_init(&c->super, plate);
10
11    c->flags = 0;
12    c->super.print = car_printf;
13    car_set_type(c, type);
14
15    return c;
16 }
```

## vehicle

```
1 void vehicle_free(struct vehicle *v)
2 {
3     if (ATTR_IS_SET(v->flags, VEHICLE_PLATE))
4         free(v->plate);
5
6     free(v);
7 }
```

## car

```
1 void car_free(struct car *c)
2 {
3     if (ATTR_IS_SET(c->flags, CAR_TYPE))
4         free(c->type);
5     vehicle_free(&c->super);
6 }
```

## Listas

¿Qué es?

Utilidad

Operaciones

Insertar  
Eliminar

Lista del  
Kernel

Ejemplo  
list\_entry  
Funciones  
Macros

# Listas encadenadas

## Tema 18

## Listas

¿Qué es?

Utilidad

Operaciones

Insertar  
Eliminar

Lista del  
Kernel

Ejemplo  
list\_entry  
Funciones  
Macros

**64** ¿Qué es una lista encadenada?

**65** ¿Cuándo son útiles?

**66** Operaciones

- Insertar
- Eliminar

**67** Lista del Kernel

- Ejemplo
- ¿Cómo se obtiene el elemento a partir del `list_head`?
- Funciones principales
- Macros principales

# ¿Qué es una lista encadenada?

## Listas

¿Qué es?

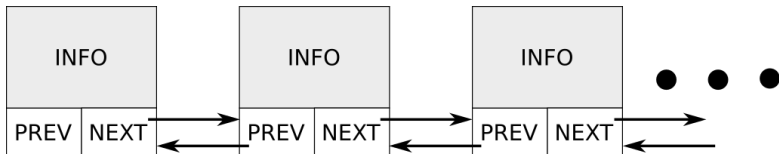
Utilidad

Operaciones

Insertar  
Eliminar

Lista del  
Kernel

Ejemplo  
list\_entry  
Funciones  
Macros



- Cada elemento guarda:
  - 1 Información
  - 2 Una referencia al siguiente elemento [y al anterior]
- Los elementos están dispersos en la memoria. Se reservan individualmente.

# ¿Cuándo son útiles?

## Listas

¿Qué es?

Utilidad

Operaciones

Insertar  
Eliminar

Lista del  
Kernel

Ejemplo  
list\_entry  
Funciones  
Macros

- **No sabemos cuántos** elementos vamos a tener que guardar
- Vamos a recorrer los elementos de manera **secuencial**
- Necesitamos hacer **inserciones** y/o **eliminaciones** de elementos o sublistas

# Insertar

## Listas

¿Qué es?

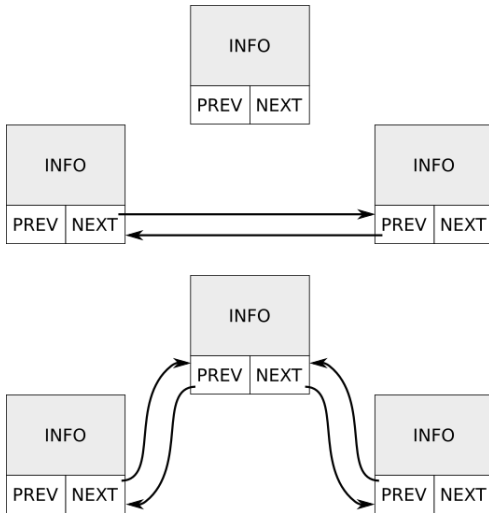
Utilidad

Operaciones

**Insertar**  
Eliminar

Lista del  
Kernel

Ejemplo  
list\_entry  
Funciones  
Macros





# Eliminar

## Listas

¿Qué es?

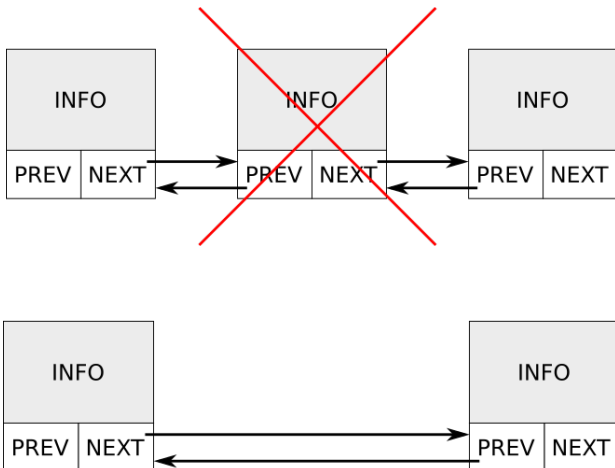
Utilidad

Operaciones

Insertar  
**Eliminar**

Lista del  
Kernel

Ejemplo  
list\_entry  
Funciones  
Macros



# Lista del Kernel

## Listas

¿Qué es?

Utilidad

Operaciones

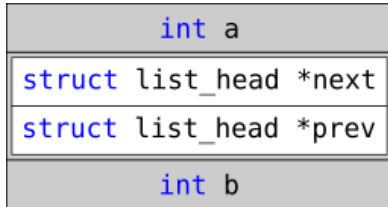
Insertar  
Eliminar

Lista del  
Kernel

Ejemplo  
list\_entry  
Funciones  
Macros

```
struct list_head *next  
struct list_head *prev
```

```
struct list_head {  
    struct list_head *next;  
    struct list_head *prev;  
};
```



```
struct element {  
    int a;  
    struct list_head list;  
    int b;  
};
```

# Ejemplo

## Listas

¿Qué es?

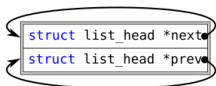
Utilidad

Operaciones

Insertar  
Eliminar

Lista del  
Kernel

**Ejemplo**  
list\_entry  
Funciones  
Macros



```
1 int main()
2 {
3     struct list_head list;
4     struct element *element;
5
6     INIT_LIST_HEAD(&list);
7
8     element = element_alloc(1, 1);
9     list_add(&(element->list), &list);
10
11    element = element_alloc(2, 2);
12    list_add(&(element->list), &list);
13
14    list_for_each_entry(element, &list, list)
15        printf("{%d, %d}\n", element->a, element->b);
16
17    return 0;
18 }
```

# Ejemplo

## Listas

¿Qué es?

Utilidad

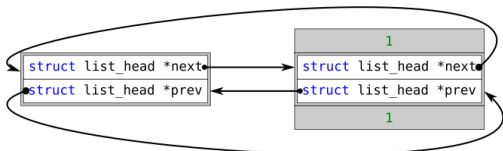
Operaciones

Insertar  
Eliminar

Lista del  
Kernel

**Ejemplo**

list\_entry  
Funciones  
Macros



```

1  int main()
2  {
3      struct list_head list;
4      struct element *element;
5
6      INIT_LIST_HEAD(&list);
7
8      element = element_alloc(1, 1);
9      list_add(&(element->list), &list);
10
11     element = element_alloc(2, 2);
12     list_add(&(element->list), &list);
13
14     list_for_each_entry(element, &list, list)
15         printf("{%d, %d}\n", element->a, element->b);
16
17     return 0;
18 }

```

# Ejemplo

## Listas

¿Qué es?

Utilidad

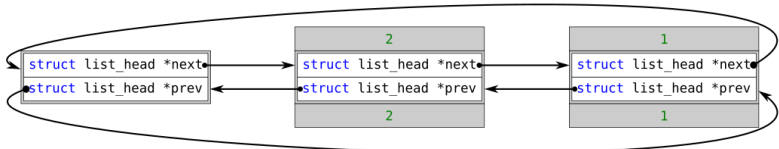
Operaciones

Insertar  
Eliminar

Lista del  
Kernel

**Ejemplo**

list\_entry  
Funciones  
Macros



```

1  int main()
2  {
3      struct list_head list;
4      struct element *element;
5
6      INIT_LIST_HEAD(&list);
7
8      element = element_alloc(1, 1);
9      list_add(&(element->list), &list);
10
11     element = element_alloc(2, 2);
12     list_add(&(element->list), &list);
13
14     list_for_each_entry(element, &list, list)
15         printf("{%d, %d}\n", element->a, element->b);
16
17     return 0;
18 }

```

# Ejemplo

## Listas

¿Qué es?

Utilidad

Operaciones

Insertar  
Eliminar

Lista del  
Kernel

**Ejemplo**

list\_entry

Funciones

Macros

```
> list_example  
{2, 2}  
{1, 1}
```

```
1 int main()  
2 {  
3     struct list_head list;  
4     struct element *element;  
5  
6     INIT_LIST_HEAD(&list);  
7  
8     element = element_alloc(1, 1);  
9     list_add(&(element->list), &list);  
10  
11    element = element_alloc(2, 2);  
12    list_add(&(element->list), &list);  
13  
14    list_for_each_entry(element, &list, list)  
15        printf("{%d, %d}\n", element->a, element->b);  
16  
17    return 0;  
18 }
```

# ¿Cómo se obtiene el elemento a partir del list\_head?

## Listas

¿Qué es?

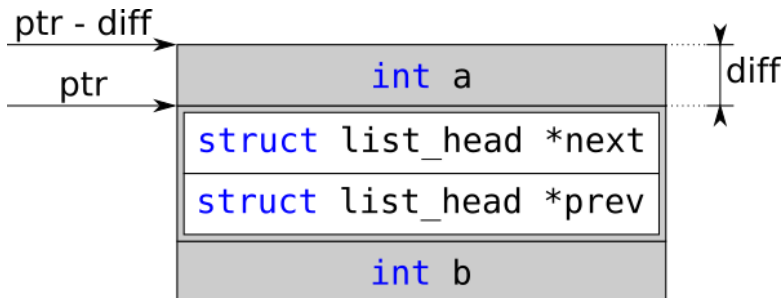
Utilidad

Operaciones

Insertar  
Eliminar

Lista del  
Kernel

Ejemplo  
**list\_entry**  
Funciones  
Macros



```

1 /**
2  * list_entry - get the struct for this entry
3  * @ptr: the &struct list_head pointer.
4  * @type: the type of the struct this is embedded in.
5  * @member: the name of the list_struct within the struct.
6  */
7 #define list_entry(ptr, type, member) \
8     ((type *)((char *)(ptr)-(unsigned long)&((type *)0)->member))

```

# Funciones principales

## Listas

¿Qué es?

Utilidad

Operaciones

Insertar  
Eliminar

Lista del  
Kernel

Ejemplo  
list\_entry

**Funciones**

Macros

- **list\_add**: Añade **después** de la cabeza (**pila**)
- **list\_add\_tail**: Añade **antes** de la cabeza (**cola**)
- **list\_move**: Mueve un elemento de una lista a **después** de la cabeza de otra (**pila**)
- **list\_move\_tail**: Mueve un elemento de una lista a **antes** de la cabeza de otra (**cola**)
- **list\_del**: Borra el nodo que recibe
- **list\_splice**: Une dos listas
- **list\_empty**: Comprueba si una lista está vacía



# Macros principales

## Listas

¿Qué es?

Utilidad

Operaciones

Insertar  
Eliminar

Lista del  
Kernel

Ejemplo  
list\_entry  
Funciones

**Macros**

- **INIT\_LIST\_HEAD**: Inicializa la cabeza de una lista
- **list\_for\_each**: Recorre cada **nodo** de una lista
- **list\_for\_each\_safe**: Recorre cada **nodo** de una lista y se puede borrar un elemento mientras se recorre la lista.
- **list\_for\_each\_entry**: Recorre cada **elemento** de una lista
- **list\_for\_each\_entry\_safe**: Recorre cada **elemento** de una lista y se puede borrar un elemento mientras se recorre la lista.

# Como usar listas encadenadas en el Juego de la Vida

## Tema 19

GOL con  
listas

**68** Introducción

**69** Estructura

**70** Algoritmo

■ Paso 1

■ Paso 2

■ Paso 3

■ Paso 4

■ Paso 5

■ Paso 6

■ Paso 7

■ Paso 8

■ Paso 9

■ Paso 10

■ Paso 11

■ Paso 12

Introducción

Estructura

Algoritmo

Paso 1

Paso 2

Paso 3

Paso 4

Paso 5

Paso 6

Paso 7

Paso 8

Paso 9

Paso 10

Paso 11

Paso 12

- Las únicas células que pueden (o no) cambiar de estados son **las vivas y sus vecinas**
- Para mundos grandes con pocas células se comprueban muchas células muertas inútilmente
- Una lista de células vivas nos permite recorrer las células de interés, obviando todas las que no pueden cambiar de estado
- Esto puede mejorar el rendimiento en mundos grandes con pocas células, pero podría empeorarlo en mundos pequeños con muchas células en un estado estable

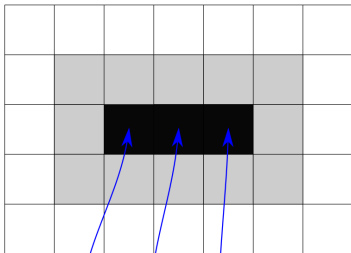
Utilizaremos tres listas cuyos nodos guardarán las coordenadas de una célula de nuestro array:

- **alive\_cells**: Lista de células vivas
- **to\_kill**: Lista de células que mueren en la siguiente iteración
- **to\_revive**: Lista de células que nacen/reviven en la siguiente iteración

- 1 Recorremos **alive\_cells**, y por cada nodo:
  - Comprobamos si sobrevive:
    - SÍ: no hacemos nada
    - NO: movemos el nodo a **to\_kill**
  - Comprobamos si nace alguna de sus vecinas muertas
    - SÍ: creamos un nuevo nodo en **to\_revive**
    - NO: no hacemos nada
- 2 Recorremos **to\_kill**, y por cada nodo: cambiamos el estado a **MUERTA** en el array y eliminamos el nodo
- 3 Recorremos **to\_revive**, y por cada nodo:
  - Si es una célula muerta: Revivir y mover el nodo a **alive\_cells**
  - Si es una célula viva: Eliminar el nodo

# Paso 1

Las células negras son las vivas y junto sus vecinas grises, son las únicas que pueden cambiar de estado



Introducción

Estructura

Algoritmo

**Paso 1**

Paso 2

Paso 3

Paso 4

Paso 5

Paso 6

Paso 7

Paso 8

Paso 9

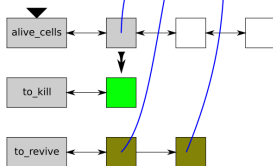
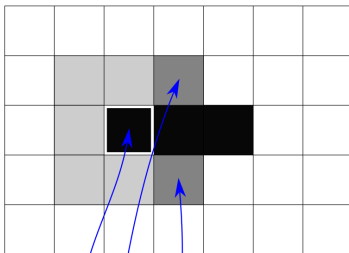
Paso 10

Paso 11

Paso 12

## Paso 2

La célula muere, movemos el nodo a *to\_kill* y añadimos los dos nacimientos a *to\_revive*



Introducción

Estructura

Algoritmo

Paso 1

**Paso 2**

Paso 3

Paso 4

Paso 5

Paso 6

Paso 7

Paso 8

Paso 9

Paso 10

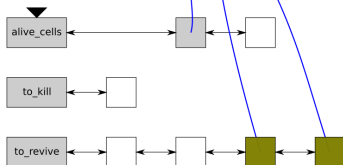
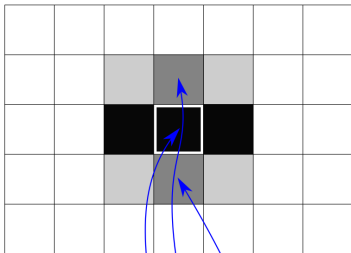
Paso 11

Paso 12



## Paso 3

La célula se queda viva. Añadimos de nuevo otros dos nodos a *to\_revive* (no importa que estén repetidos)



Introducción

Estructura

Algoritmo

Paso 1

Paso 2

**Paso 3**

Paso 4

Paso 5

Paso 6

Paso 7

Paso 8

Paso 9

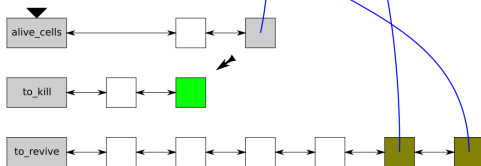
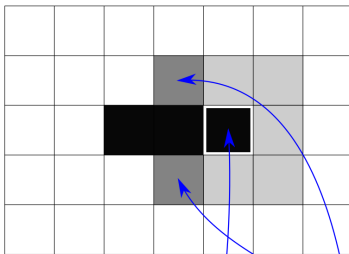
Paso 10

Paso 11

Paso 12

# Paso 4

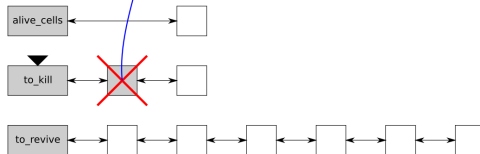
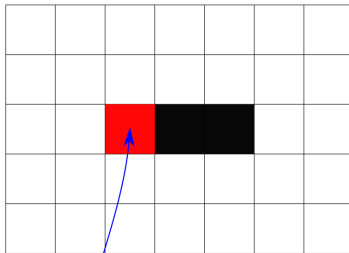
La célula muere, movemos el nodo a *to\_kill* y añadimos de nuevo dos nodos a *to\_revive*



# Paso 5

GOL con listas

Marcamos la célula como muerta en el array y eliminamos el nodo

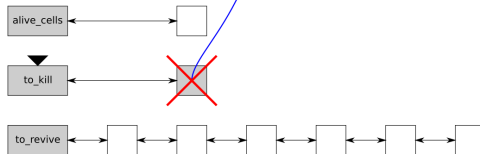
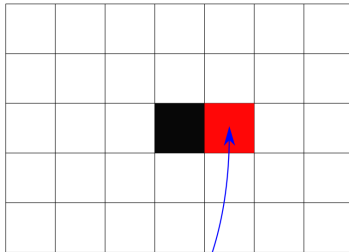


- Introducción
- Estructura
- Algoritmo
  - Paso 1
  - Paso 2
  - Paso 3
  - Paso 4
  - Paso 5**
  - Paso 6
  - Paso 7
  - Paso 8
  - Paso 9
  - Paso 10
  - Paso 11
  - Paso 12

# Paso 6

GOL con listas

Marcamos la célula como muerta en el array y eliminamos el nodo



Introducción

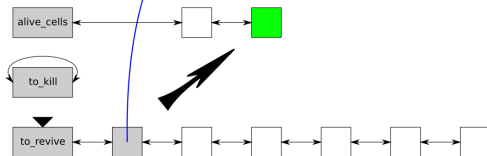
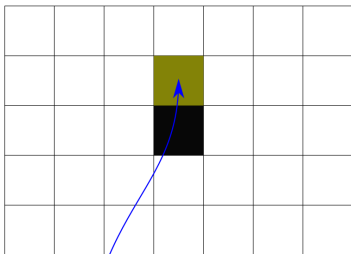
Estructura

Algoritmo

- Paso 1
- Paso 2
- Paso 3
- Paso 4
- Paso 5
- Paso 6**
- Paso 7
- Paso 8
- Paso 9
- Paso 10
- Paso 11
- Paso 12

# Paso 7

La célula está muerta así que movemos el nodo a *alive\_cells* y la marcamos como viva en el array



Introducción

Estructura

Algoritmo

Paso 1

Paso 2

Paso 3

Paso 4

Paso 5

Paso 6

**Paso 7**

Paso 8

Paso 9

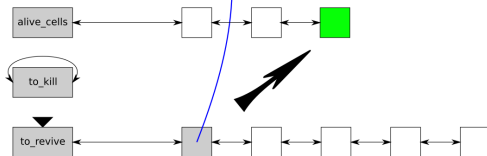
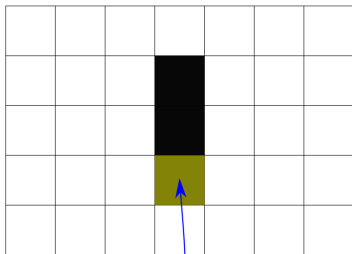
Paso 10

Paso 11

Paso 12

# Paso 8

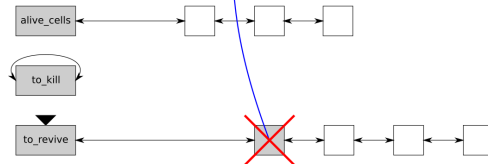
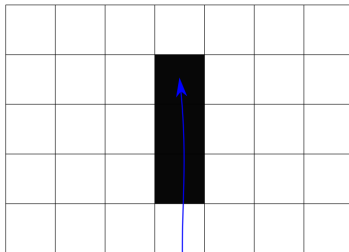
La célula está muerta así que movemos el nodo a *alive\_cells* y la marcamos como viva en el array



# Paso 9

GOL con listas

La célula ya está viva así que eliminamos el nodo



# Paso 10

## GOL con listas

Introducción

Estructura

Algoritmo

Paso 1

Paso 2

Paso 3

Paso 4

Paso 5

Paso 6

Paso 7

Paso 8

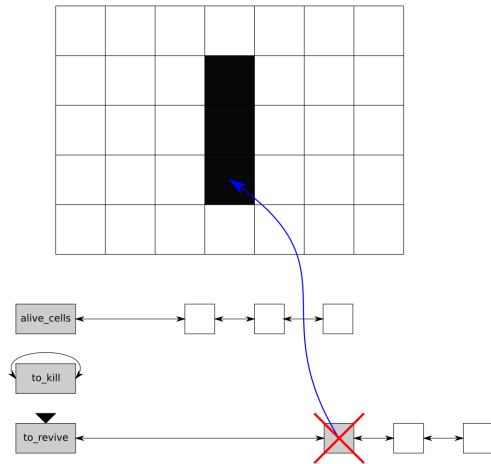
Paso 9

**Paso 10**

Paso 11

Paso 12

La célula ya está viva así que eliminamos el nodo





# Paso 11

GOL con listas

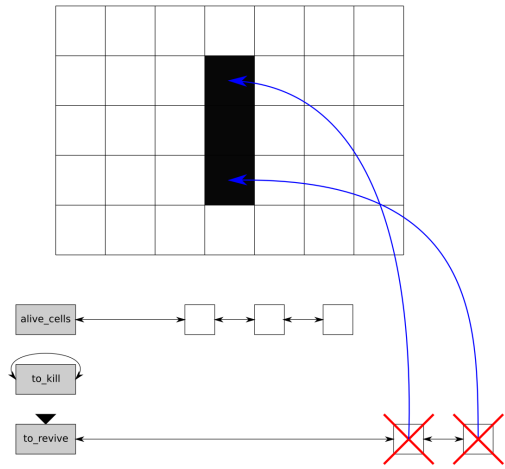
Esto mismo ocurre con todos los nodos ya revividos (no importa que estén duplicados)

Introducción

Estructura

Algoritmo

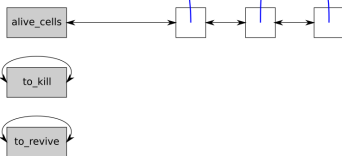
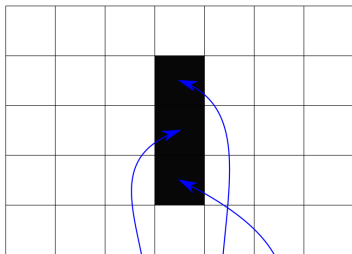
- Paso 1
- Paso 2
- Paso 3
- Paso 4
- Paso 5
- Paso 6
- Paso 7
- Paso 8
- Paso 9
- Paso 10
- Paso 11**
- Paso 12



# Paso 12

GOL con listas

Ya tenemos la iteración completada y nuestra lista *alive\_cells* preparada para la siguiente



Introducción

Estructura

Algoritmo

- Paso 1
- Paso 2
- Paso 3
- Paso 4
- Paso 5
- Paso 6
- Paso 7
- Paso 8
- Paso 9
- Paso 10
- Paso 11
- Paso 12**

## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area  
Timer  
Ejercicio

# GTK

## Tema 20

## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area  
Timer  
Ejercicio

- 71** Sobre GTK
- 72** Bucle principal
- 73** Señales
- 74** Callbacks
- 75** Glade y builder
- 76** Ejemplo simple
- 77** Juego de la vida
  - main
  - gui object
  - drawing area
  - Timer
  - Ejercicio

## GTK

### Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area  
Timer  
Ejercicio

- **GTK** (Gimp ToolKit) es una biblioteca libre y multiplataforma para el desarrollo de interfaces gráficas de usuario
- Está escrita en C, aunque tiene *bindings* a una infinidad de lenguajes distintos (C++, C#, Python, Lua, Haskell, ...)
- Se trata de una biblioteca muy popular, con mucha comunidad alrededor y muchos años de madurez
- Las interfaces se construyen mediante la anidación de diversos **widget**. Tiene una enorme variedad de widgets (ejecutar “`gtk3-widget-factory`”)
- Dispone de una documentación detallada y de calidad (<https://developer.gnome.org/gtk3/stable/>). Además de numerosos tutoriales y ejemplos desarrollados por su gran comunidad.

# Bucle principal

## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area  
Timer  
Ejercicio

- Toda la lógica de la interfaz se realiza dentro de un bucle infinito `gtk_main()`;
- Antes del bucle la interfaz no es funcional
- Después del bucle no existe interfaz
- No podemos modificar el interior del bucle
- **¿Cómo interacciono con la interfaz gráfica?**

```
1 #include <gtk/gtk.h>
2
3 int main(int argc, char **argv)
4 {
5     GtkWidget *window;
6
7     gtk_init(&argc, &argv);
8     window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
9     gtk_widget_show_all(window);
10    gtk_main();
11
12    return 0;
13 }
```

## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area  
Timer  
Ejercicio

- Una señal puede verse como un **mensaje** que nos envía GTK cada vez que ocurre un **evento**
- Un **evento** puede ser cualquier acción del usuario: un click en un botón, mover el ratón sobre una ventana, arrastrar algún objeto, . . . . También hay eventos internos de la interfaz gráfica: se acaba de crear/destruir la ventana, se va a redibujar algún objeto, . . .
- En GTK se implementan mediante **callbacks**

## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area  
Timer  
Ejercicio

- Un callback es una función que pasamos a alguna librería, para que ella se encargue de llamarla cuando convenga
- GTK guarda una lista de punteros a funciones por cada posible evento
- Como usuarios de GTK, **conectamos** una o más funciones de callback a una señal (pulsar un botón). Cuando ocurra el evento oportuno, GTK se encargará de llamarlas a todas



# Glade y builder

## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

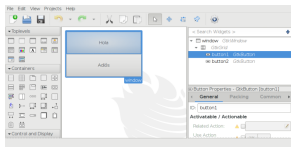
Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area  
Timer  
Ejercicio

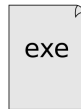
- Crear una interfaz compleja directamente en un lenguaje de programación es complejo y poco versátil
- GTK puede crear e inicializar todos los objetos necesarios a partir de un XML con la descripción de la interfaz
- **Glade** es un programa que nos ayuda a construir este XML de manera gráfica



Glade



builder.ui



gol

# Ejemplo simple

## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area  
Timer  
Ejercicio

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <gtk/gtk.h>
4
5 #define BUILDER_FILE "builder.ui"
6
7 int main(int argc, char **argv)
8 {
9     GtkBuilder *builder;
10    GtkWidget *window;
11    GtkWidget *button1;
12    GtkWidget *button2;
13
14    gtk_init(&argc, &argv);
15
16    /* builder */
17    builder = gtk_builder_new();
18    if (!gtk_builder_add_from_file(builder, BUILDER_FILE, NULL)) {
19        fprintf(stderr, "Can't open file \"%s\"\n", BUILDER_FILE);
20        exit(EXIT_FAILURE);
21    }
22
23    /* objects */
24    window = GTK_WIDGET(gtk_builder_get_object(builder, "window"));
25    button1 = GTK_WIDGET(gtk_builder_get_object(builder, "button1"));
26    button2 = GTK_WIDGET(gtk_builder_get_object(builder, "button2"));
```

# Ejemplo simple

## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area  
Timer  
Ejercicio

```
27
28  /* signals */
29  g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit),
      NULL);
30  g_signal_connect(button1, "clicked", G_CALLBACK(button_cb), "Hola"
      );
31  g_signal_connect(button2, "clicked", G_CALLBACK(button_cb), "Adios"
      "");
32  gtk_builder_connect_signals(builder, NULL);
33
34  gtk_widget_show_all(window);
35  gtk_main();
36
37  return 0;
38 }
39
40 static void button_cb(GtkWidget *widget, const char *str)
41 {
42     printf("%s mundo!\n", str);
43 }
```

```
gcc -g main.c $(pkg-config --cflags gtk+-3.0 --libs
gtk+-3.0)
```

## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area  
Timer  
Ejercicio

# Juego de la vida



# main

## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

**main**  
gui object  
drawing area  
Timer  
Ejercicio

```
1 int main(int argc, char **argv)
2 {
3     struct world *w;
4     struct gui *g;
5
6     gtk_init(&argc, &argv);
7
8     w = world_alloc(WORLD_X, WORLD_Y);
9     if (!w) {
10         perror("Can't allocate world");
11         exit(EXIT_FAILURE);
12     }
13     world_init(w);
14
15     g = gui_alloc("builder.ui", w, WINDOW_X, WINDOW_Y);
16     if (!g) {
17         perror("Can't create gui");
18         exit(EXIT_FAILURE);
19     }
20
21     gtk_main();
22
23     world_free(w);
24     gui_free(g);
25
26     return 0;
27 }
```

# gui object

## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area  
Timer  
Ejercicio

```
1 struct gui {  
2     GtkBuilder *builder;  
3  
4     GtkWidget *window;  
5     GtkWidget *btn_quit;  
6     GtkWidget *btn_step;  
7     GtkWidget *btn_pause;  
8     GtkGrid *grid;  
9     GtkWidget *drawing_area;  
10  
11     struct {  
12         double x;  
13         double y;  
14     } cell_size;  
15  
16     bool run;  
17  
18     struct world *world;  
19 };
```

# drawing area

## GTK

### Sobre GTK

### Bucle principal

### Señales

### Callbacks

### Glade y builder

### Ejemplo simple

### Juego de la vida

### main gui object **drawing area** Timer Ejercicio

```

1  /* drawing area */
2  g->drawing_area = gtk_drawing_area_new();
3  // Tamaño de la ventana de dibujado
4  gtk_widget_set_size_request(g->drawing_area, ws_x, ws_y);
5  // Necesario para capturar los click del raton
6  gtk_widget_set_events(g->drawing_area,
7    gtk_widget_get_events(g->drawing_area) | GDK_BUTTON_PRESS_MASK);
8  // Anyadimos el nuevo widget al grid
9  gtk_grid_attach_next_to(g->grid, g->drawing_area, g->btn_quit,
10    GTK_POS_BOTTOM, 1, 1);

```

```

1  static gboolean draw_cb(GtkWidget *widget, cairo_t *cr, struct gui *
    g)
2  {
3    /* Clear screen */
4    cairo_set_source_rgb(cr, 0, 0, 0);
5    cairo_paint(cr);
6
7    /* TODO
8     * Dibuja el mundo sabiendo que el código de abajo dibuja un
        rectángulo
9     * blanco en las coordenadas (2, 3) de 5x4 pixeles de tamaño
10    */
11    cairo_set_source_rgb(cr, 1, 1, 1);
12    cairo_rectangle(cr, 2, 3, 5, 4);
13    cairo_fill(cr);
14
15    return false;
16 }

```

# Timer

## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area

**Timer**  
Ejercicio

```

1 static void pause_cb(GtkWidget *widget, struct gui *g)
2 {
3     if (!g->run)
4         g_timeout_add(VEL_MS, timer_cb, g); // Conectamos la senyal de
           nuevo
5     g->run = !g->run;
6 }

```

```

1 static gboolean timer_cb(gpointer gui)
2 {
3     struct gui *g = (struct gui *)gui;
4     step_cb(NULL, g);
5     /* Si devuelve false, la senyal se desconecta y el callback no se
           vuelve
6     * a llamar
7     */
8     return g->run;
9 }

```



## GTK

Sobre GTK

Bucle principal

Señales

Callbacks

Glade y builder

Ejemplo simple

Juego de la vida

main  
gui object  
drawing area  
Timer  
**Ejercicio**

- 1 Estudia detenidamente el código proporcionado
- 2 Crea el fichero “`builder.ui`” con **Glade**. Presta atención al identificador de cada widget
- 3 Completa el código de “`gui.c`”. Las zonas a completar están marcadas con un **TODO**