

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

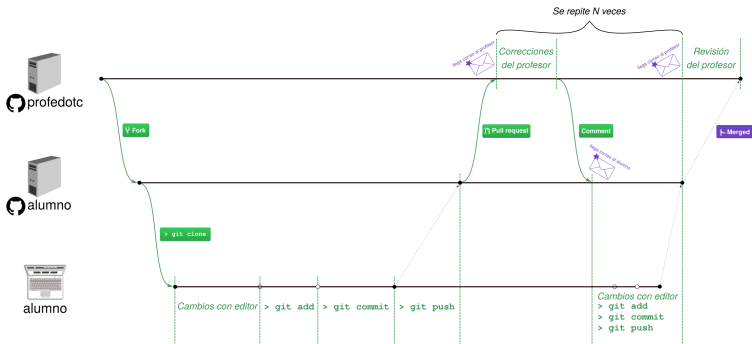
¿Por qué C?

¿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?

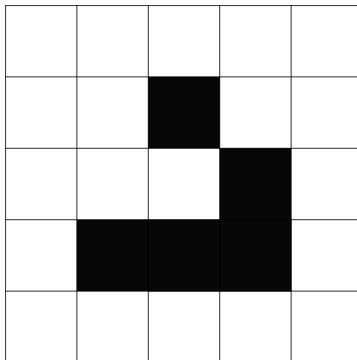
Proyectos en C

Summer Of Code

GSOC

Outreachy

Juego de la vida de Conway



https://en.wikipedia.org/wiki/Conway's_Game_of_Life



- 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



- 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.

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



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**

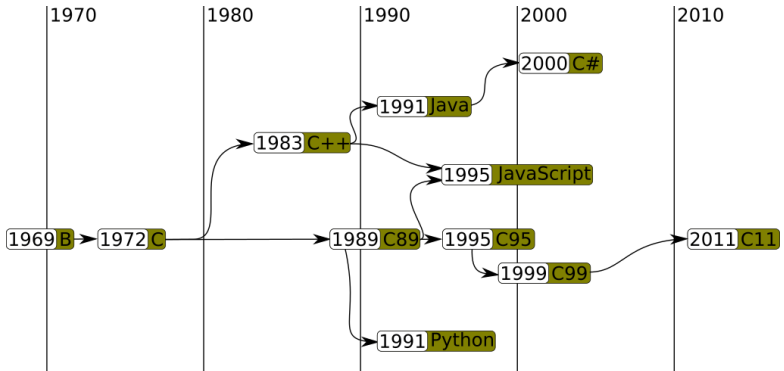
¿Por qué C?

¿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

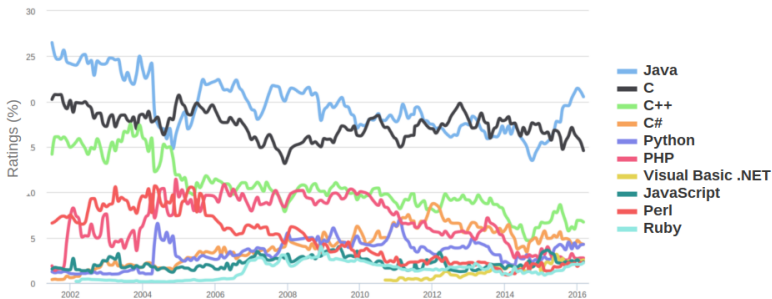
Summer Of Code

GSOC
Outreachy

• Popularidad

TIOBE Programming Community Index

Source: www.tiobe.com



¿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?

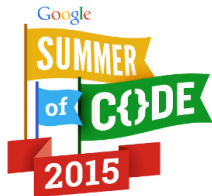
Proyectos en C

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

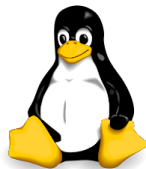
7 Consola

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
43 EXPORT_SYMBOL(nftnl_udata_parse);
44
45 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 modules.order README tools
certs fs kernel modules.order REPORTING-BUGS user
COPYING include kva-conf-3.12 Module.symvers samples virt
CREDITS init lib net scripts valinux vmlinux.o
crypto mskt.sh MAINTAINERS nft-10595-gbccc90f security valinux.o
Documentation ipc kexecfile RT-907429c sound

git log -1
commit 3816c7947bd0ebd3a1c34097a5df675eeaf190
Author: Carlos Falgout 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>

```
#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

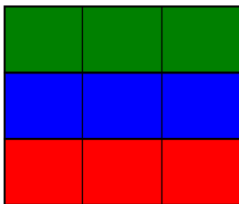
Descripción

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

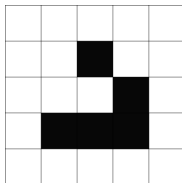
Ejemplo

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
- Simulador (muy bueno):
golly.sourceforge.net/
- Simulador web:
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

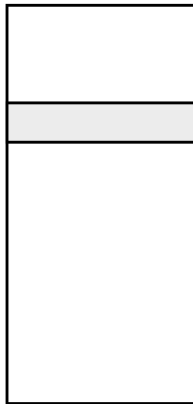
Ejemplo
Recorriendo
arrays

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?

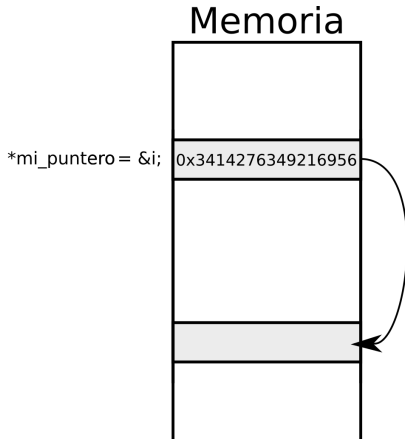
Ejemplo
Sintaxis

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?

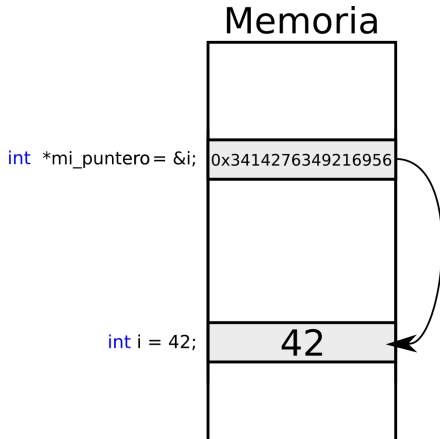
Ejemplo
Sintaxis

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

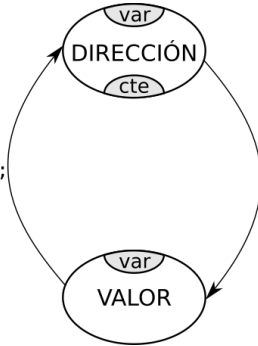
Arrays y
punteros

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

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

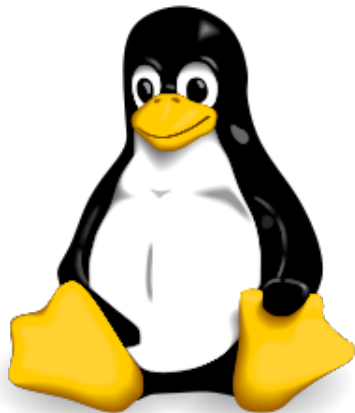
Descripción

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

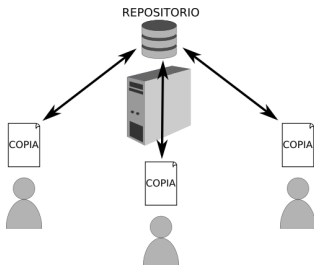
Fork de mi
repositorio

Buscar mi
repositorio
Fork
Clonar mi
repositorio

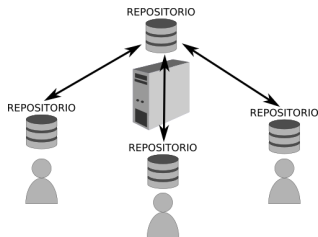
Workflow

Cambios
git push
Pull request
Comprobación
Solicitado
Revisiones

Centralizado



Distribuido



GIT

Git

Características
Distribuido
VS
Centralizado
Funcionamiento

Cuenta en
GitHub

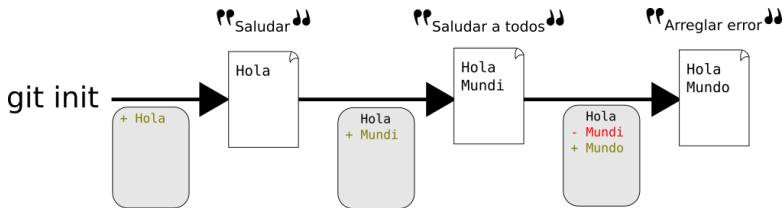
Plan gratuito
Confirmar
correo

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

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: (with a green checkmark)
- Email: (with a green checkmark)
- Password: (with a green checkmark)

Below the password field, there is a note: "Use at least one letter, one numeral, and seven characters."

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

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

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	Choose
Medium	\$22/month	20	Choose
Small	\$12/month	10	Choose
Micro	\$7/month	5	Choose
Free	\$0/month	0	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.](#)

Finish sign up

Debemos confirmar la dirección de correo

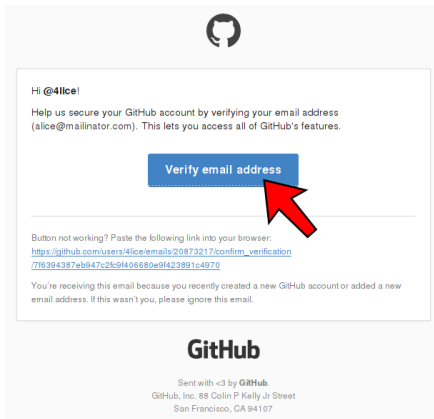


Please verify your email address

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](#).

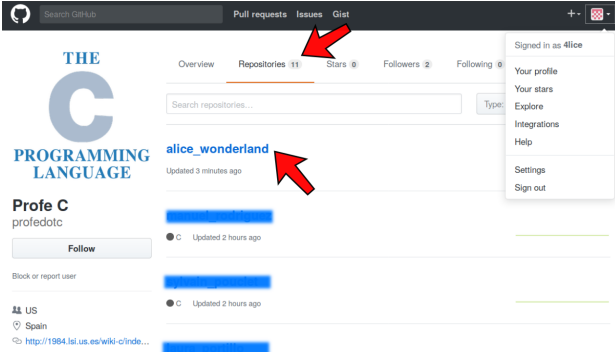
Buscamos el correo de confirmación en nuestro buzón y hacemos click en “*Verify email address*”



Buscar mi repositorio

GIT

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



The screenshot shows the GitHub profile page for the 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 signed in as '4lice' and have 11 repositories, 0 stars, 2 followers, and 0 following.

Git

Características
Distribuido
VS
Centralizado
Funcionamiento

Cuenta en
GitHub

Plan gratuito
Confirmar
correo

Fork de mi
repositorio

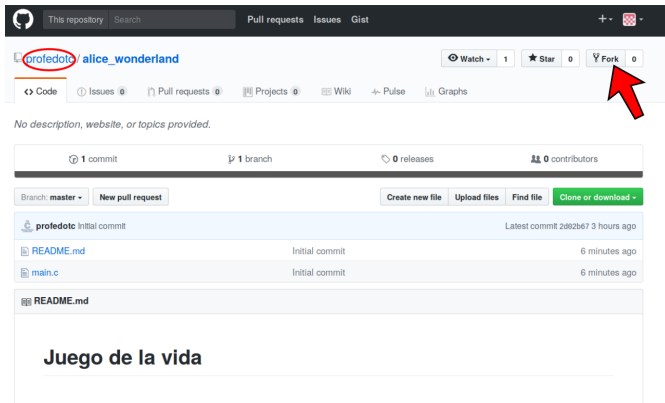
Buscar mi
repositorio

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

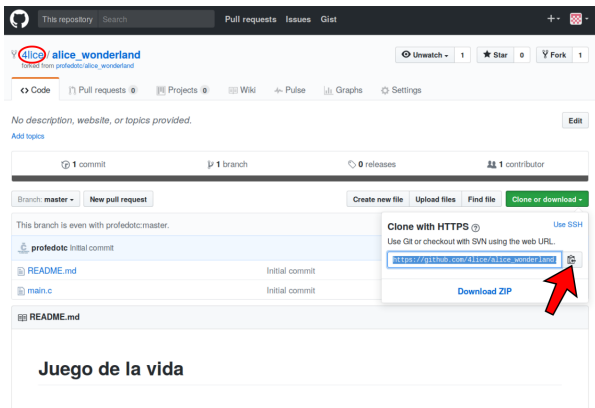


The screenshot shows the GitHub interface for the repository 'profedotc/alice_wonderland'. The repository name 'profedotc' is circled in red. A red arrow points to the 'Fork' button, which is currently set to 0 forks. Below the repository name, there are buttons for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Pulse', and 'Graphs'. A description states 'No description, website, or topics provided.' Below this, there are statistics for 1 commit, 1 branch, 0 releases, and 0 contributors. A table shows the commit history with columns for file names (README.md, main.c) and their commit times (Initial commit, 6 minutes ago). At the bottom, the README content is visible, starting with '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 'profedotc'. The 'Clone or download' button is highlighted in green, and its dropdown menu is open. The menu contains the following options:

- Clone with HTTPS (selected): Use Git or checkout with SVN using the web URL. The URL `https://github.com/alice/alice_wonderland` is displayed and highlighted with a blue box. A red arrow points to the copy icon next to the URL.
- Use SSH
- Download ZIP

The repository page also shows 1 commit, 1 branch, 0 releases, and 1 contributor. The main content area displays the README for '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

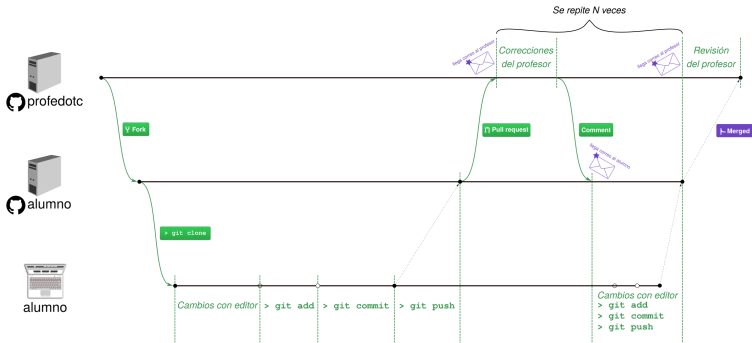
Pull request

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

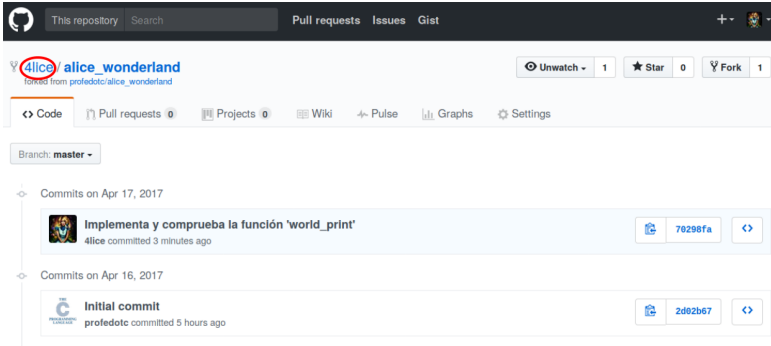
Cambios

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 name '4lice' is circled in red. The page displays the repository name, a search bar, and navigation links for Pull requests, Issues, and Gist. Below the repository name, there are statistics for Unwatch (1), Star (0), and Fork (1). The main content area shows the commit history for the 'master' branch, with two commits listed: 'Implementa y comprueba la función 'world_print'' (committed 3 minutes ago) and 'Initial commit' (committed 5 hours ago).

Git

Características
Distribuido
VS
Centralizado
Funcionamiento

Cuenta en GitHub

Plan gratuito
Confirmar correo

Fork de mi repositorio

Buscar mi repositorio
Fork
Clonar mi repositorio

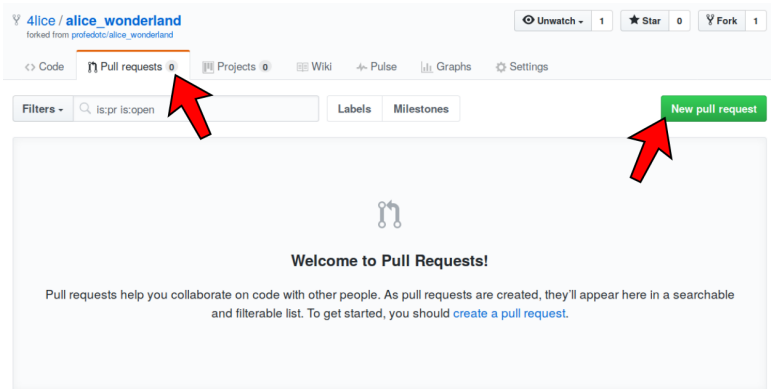
Workflow

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 profedote/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*”

Git

Características
Distribuido
VS
Centralizado
Funcionamiento

Cuenta en
GitHub

Plan gratuito
Confirmar
correo

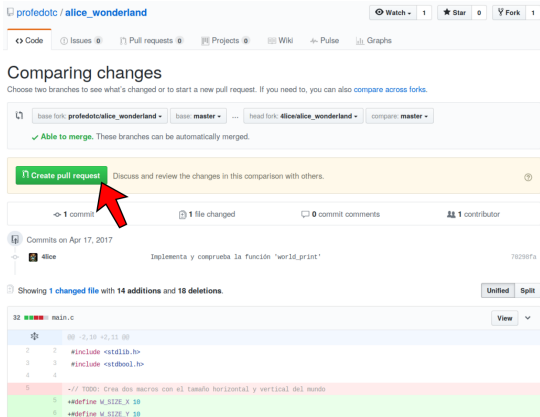
Fork de mi
repositorio
Buscar mi
repositorio
Fork
Clonar mi
repositorio

Workflow

Cambios
git push
Pull request

Comprobación

Descripción
Solicitado
Revisiones



profetoto / 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: profetoto/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.

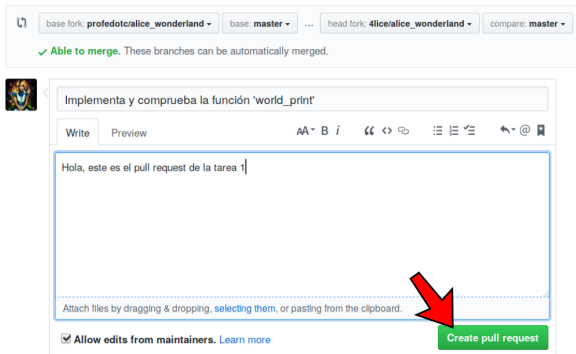
```

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
  
```

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.

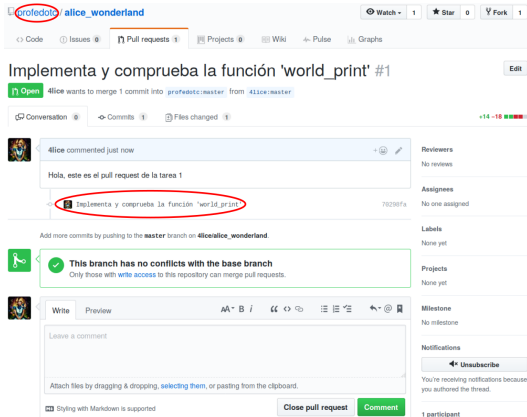
Allow edits from maintainers. [Learn more](#)

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. At the top, the repository name 'profedotc/alice_wonderland' is circled in red. The pull request title is 'Implementa y comprueba la función 'world_print' #1'. A green 'Open' button indicates the pull request is ready for review. Below the title, a comment from '4lice' states: 'Hola, este es el pull request de la tarea 1'. The title of the pull request is also circled in red. A green success message reads: 'This branch has no conflicts with the base branch'. At the bottom, there is a 'Close pull request' button and a 'Comment' 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

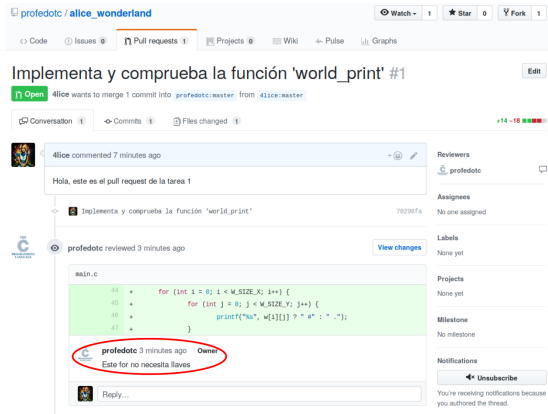
Workflow

Cambios
git push
Pull request
Comprobación
Descripción
Solicitado
Revisiones

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 titled "Implementa y comprueba la función 'world_print' #1". The pull request is from "Alice:master" to "profedotc:master".

Comments:

- Alice commented 7 minutos ago: "Hola, este es el pull request de la tarea 1"
- profedotc reviewed 3 minutos ago:


```

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 +       }
      
```

 Below the code, a comment from "profedotc 3 minutos ago" says "Este for no necesita llaves" (This for loop doesn't need braces). This comment is circled in red in the original image.

Right sidebar:

- Reviewers: profedotc
- Assignees: No one assigned
- Labels: None yet
- Projects: None yet
- Milestone: No milestone
- Notifications: Unsubscribe

Git
Características
Distribuido
VS
Centralizado
Funcionamiento

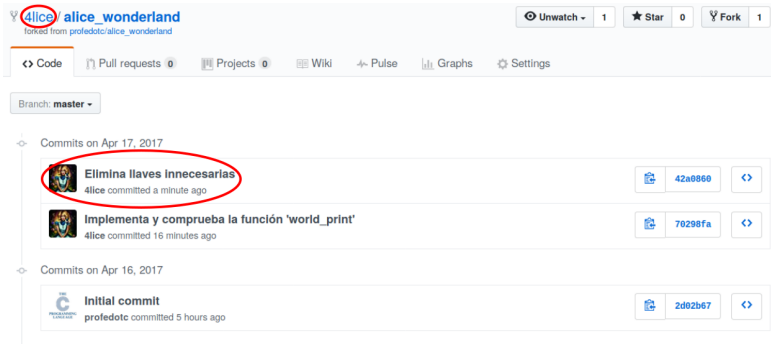
Cuenta en
GitHub
Plan gratuito
Confirmar
correo

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 it, 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' and is titled 'Implementa y comprueba la función 'world_print''. The second commit is by 'profedotc' and is titled 'Elimina llaves innecesarias', which is circled in red in the image. Below the commit list, there is a 'main.c' file that is 'Show outdated'. At the bottom, there is a note: 'Add more commits by pushing to the master branch on 4lice/alice_wonderland.'

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 42ab660

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

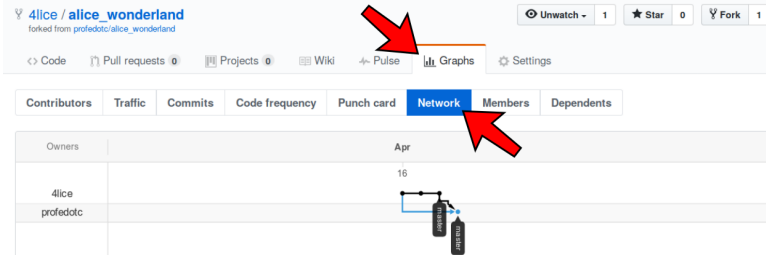
Milestone: No milestone

Notifications: Unsubscribe

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

2 participants

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 diagram showing a pull request from profedotc to 4lice.

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

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

```

Campos de bits

Estructuras

Struct

Alineación y tamaño
Anidamiento
Anónimas
Arrays y punteros

Union

Ejemplo 1
Ejemplo 2

Campos de bits

Ejemplo

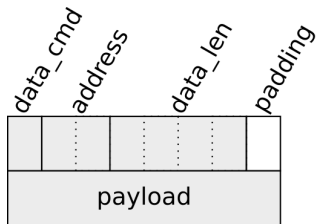
Enumerados

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

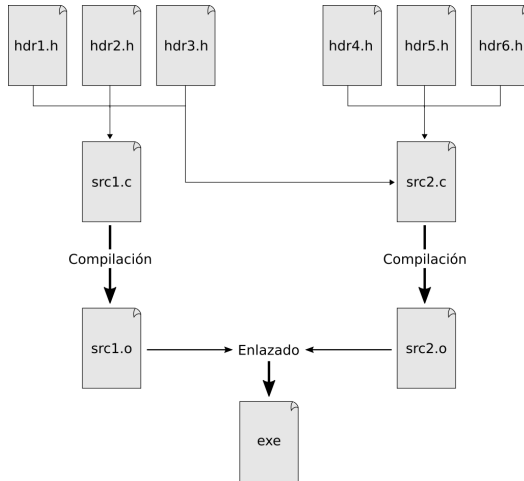
C Modular

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

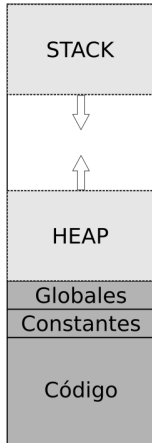
Uso tras liberación

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

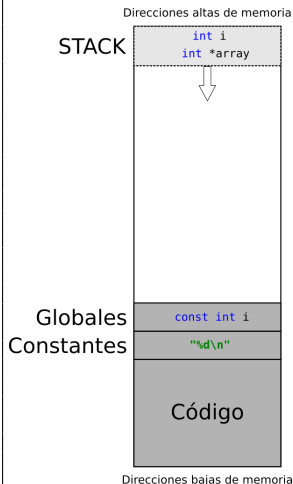
Uso tras liberación

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

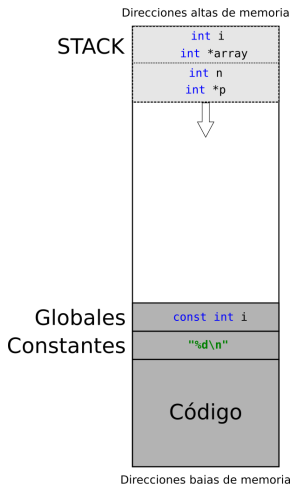
valgrind

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

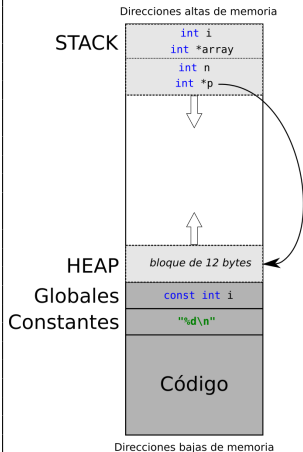
Uso tras liberación

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

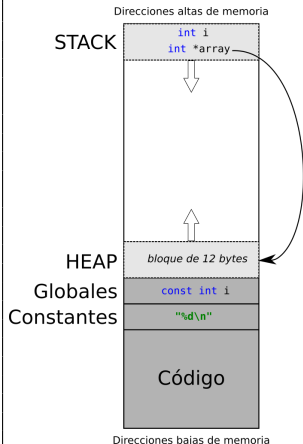
Uso tras liberación

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

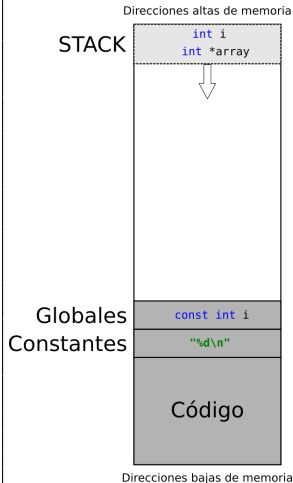
Uso tras liberación

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

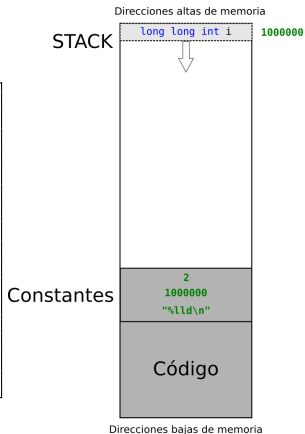
Uso tras liberación

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

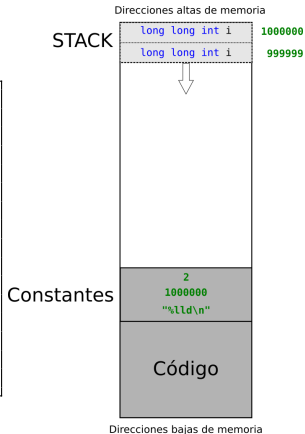
valgrind Ejemplo

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

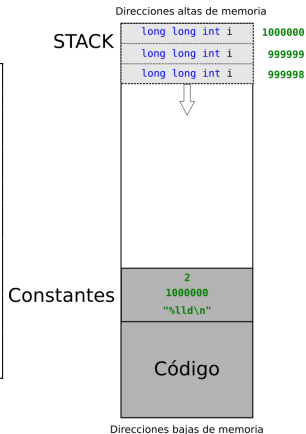
valgrind Ejemplo

```

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

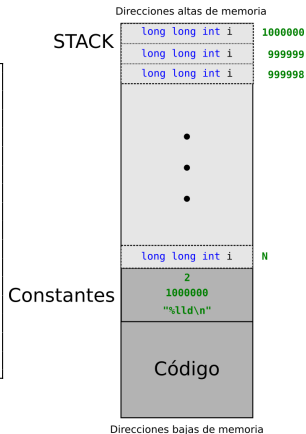
Ejemplo

```

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



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

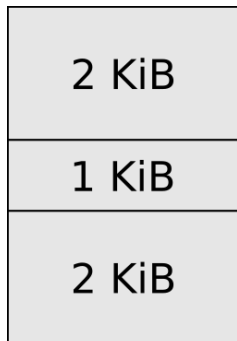
valgrind

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

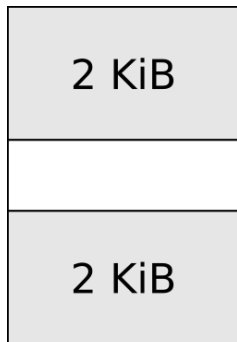
valgrind Ejemplo

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

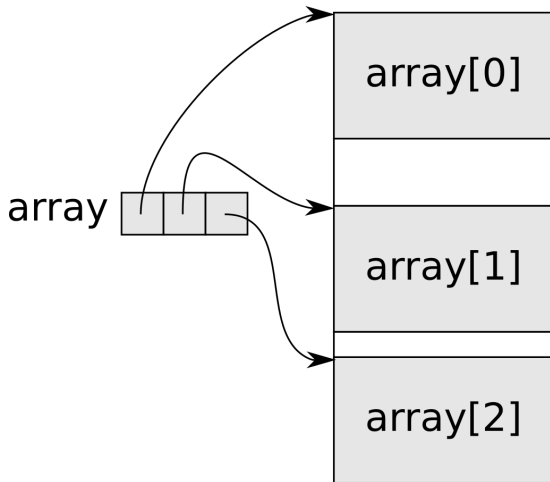
Un malloc por cada fila del array (array de punteros)

```
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)
```