



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



Índice

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

El curso

Profesores

Temario

Material de clase

Herramientas

Flujo de trabajo

Evaluación

Historia de C

Inicios

Influencias

¿Por qué C?

¿Para qué C?

Proyectos en C

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

Sumer 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



Material de clase

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://listas.us.es/mailman/listinfo/programacion-c>



Herramientas

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*)

Introducción

El curso

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

Historia

- Inicios
- Influencias

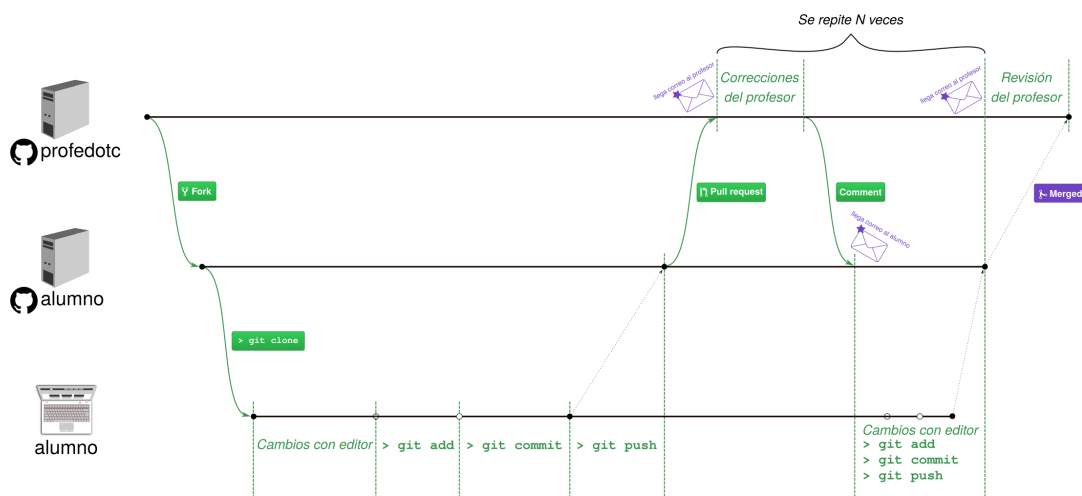
¿Por qué C?

¿Para qué C?

- Proyectos en C

Sumer Of Code

- GSOC
- Outreachy



Introducción

Juego de la vida de Conway

El curso

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

Historia

- Inicios
- Influencias

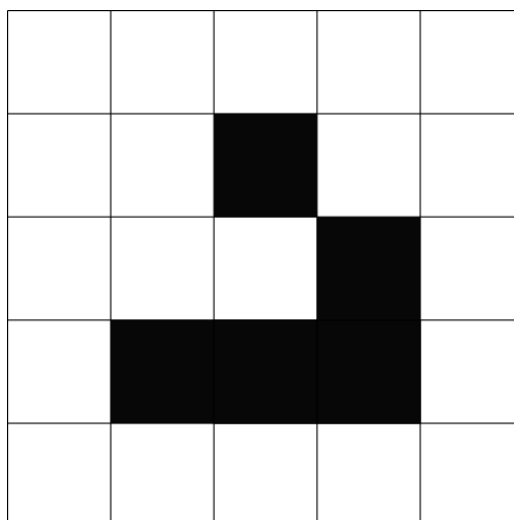
¿Por qué C?

¿Para qué C?

- Proyectos en C

Sumer Of Code

- GSOC
- Outreachy



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



Ken Thompson
Dennis Ritchie
Brian Kernighan

Bell Labs (de AT&T)

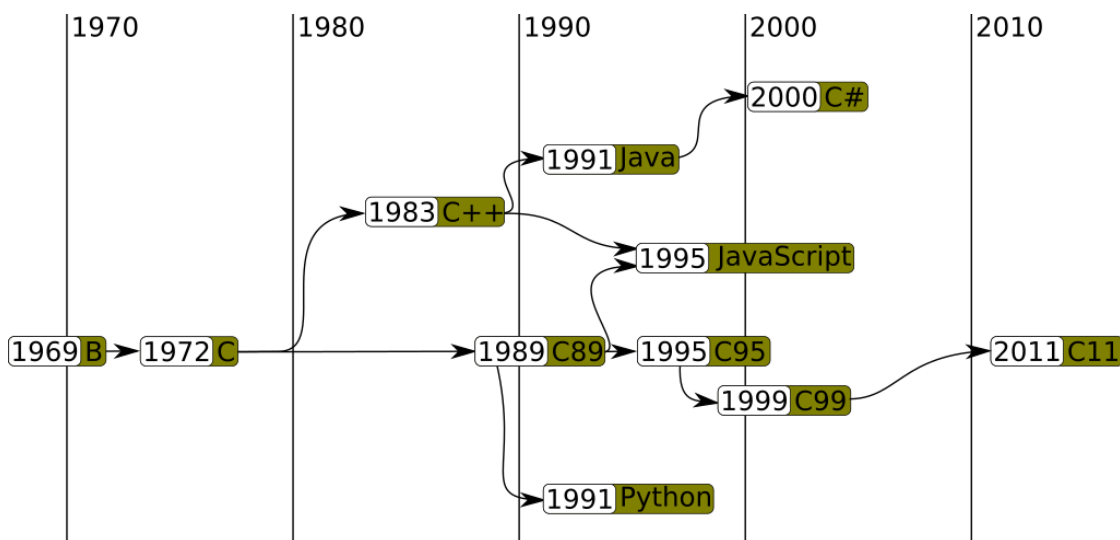
Ensamblador y **B** insuficientes → diseñan **C**

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.



¿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

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

Sumer Of Code

Code

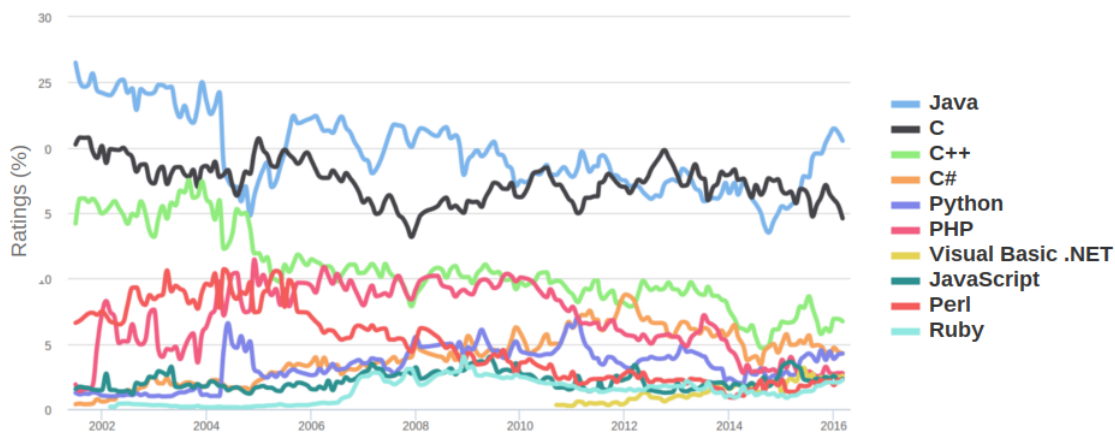
GSOC

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

Proyectos en 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

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 líneas)



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

Sumer Of Code

Code

GSOC

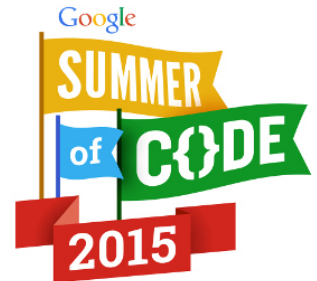
Outreachy

Beca de Google para estudiantes

Trabajas **3 meses** en un proyecto de **software libre**

Experiencia

Dinero: 5500\$



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



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

Code

GSOC

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\$

Hasta el 22 Marzo <https://gnome.org/outreachy/>



Workspace

Linux

Consola

Instalando
herramientas

Hola Mundo

Entorno de trabajo

Tema 2



Índice

Workspace

Linux

Consola

Instalando
herramientas

Hola Mundo

Linux

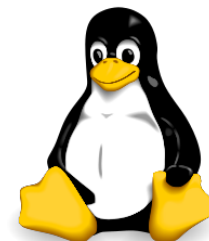
Consola

Instalando herramientas

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



CTRL + ALT + T

```

s/udata.c | 1/audit.h |
22 EXPORT_SYMBOL(nftnl_udata_attr_value);
21
20 struct nftnl_udata *nftnl_udata_attr_next(const struct nftnl_udata *attr)
19 {
18     return (struct nftnl_udata *)&attr->value[attr->len];
17 }
16 EXPORT_SYMBOL(nftnl_udata_attr_next);
15
14 int nftnl_udata_parse(const struct nftnl_udata_buf *buf, nftnl_udata_cb_t cb,
13     void *data)
12 {
11     int ret = 0;
10     const struct nftnl_udata *attr;
9
8     nftnl_udata_for_each(buf, attr) {
7         ret = cb(attr, data);
6         if (ret == 0)
5             return ret;
4     }
3     return ret;
2 }
1
134 EXPORT_SYMBOL(nftnl_udata_parse);
src/udata.c c 100% | 134:1 |
11 + | (1 byte) | (1 byte) |
10 + +-----+-----+-----+-----+-----+-----+-----+-----+
9 + <<- sizeof(nftnl_udata) -> <<- nftnl_udata->len ->>
8 +/+
7 struct nftnl_udata {
6     uint8_t type;
5     uint8_t len;
4     unsigned char value[];
3 } __attribute__((packed));
2
1 +/+
24 | |
1 + |-----+-----+-----+-----+-----+-----+-----+-----+
2 + | data[] |
3 + |-----+-----+-----+-----+-----+-----+-----+-----+
4 + | size | end | TLV | TLV | TLV | TLV | Empty |
5 + +-----+-----+-----+-----+-----+-----+-----+-----+
6 + |<<- nftnl_udata_len() ->>|
7 + |<----- nftnl_udata_size() ->|
8 +/+
9 struct nftnl_udata_buf {
10     size_t size;
11 }
include/udata.h nftnl_udata [] cpp utf-8[unix] | 66% | 24:1 | buffers

ls
arch      drivers  kbuild  mm          patches    System.map
block     firmware kconfig  modules.builtins  README     tools
certs     fs        kernel  modules.order  REPORTING-BUGS  usr
COPYING  include  kver-config-3.12  Module.symvers  samples    virt
CREDITS  init      lib      net          scripts    valinux
crypto    insnk.sh MAINTAINERS  nft-185895-gbce98f  security   valinux.o
Documentation  ipc      Makefile  NFT-g5742b9e    sound

git log -1
commit 3816c7947bd6c8db3a1c34987a5dfb76effa9100
Author: Carlos Falgueras Garcia <carlosfg@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 NF_TABLE_SET_USERDATA.
linker@fhs:~/nftables$ git log -1

```



Consola

Workspace

Linux

Consola

Instalando
herramientas

Hola Mundo

Comandos básicos:

Lista directorios

Cambia a directorio

Crea directorio

Crea archivo vacío*

Borra archivo

Borra directorio y lo que hay dentro



Instalando herramientas

Workspace

Linux

Consola

Instalando
herramientas

Hola Mundo

Comandos para instalar:

Ejecuta un comando con
permisos de administrador

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:

Linux

Consola

Instalando
herramientas

Hola Mundo

```
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

¿Qué es una variable?

Tipos básicos

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

`int contador;`
reservo 4 bytes
voy a contar numeros enteros (grandes)

`char c;`
reservo 1 byte
voy a guardar un caracter

`char contador;`
reservo 1 byte
voy a contar numeros enteros (pequeños)

Tipos básicos

Variables

¿Qué es?

Tipos básicos

Tipos de tamaño fijo

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	uint8_t
int16_t	uint16_t
int32_t	uint32_t
int64_t	uint64_t



Arrays

Descripción

Ejemplo

Cadenas

Array multi-
dimensional

Arrays y tipos

Tema 4



Índice

Arrays

Descripción

Ejemplo

Cadenas

Array multi-
dimensional

Descripción

Ejemplo

Cadenas

Array multidimensional

Arrays

Descripción

Ejemplo

Cadenas

Array multi-dimensional

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

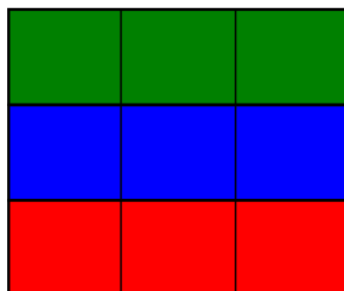
Descripción

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



Índice

GOL

Introducción

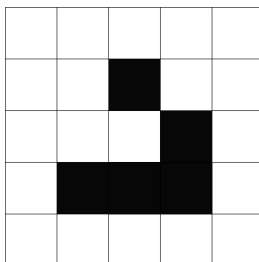
Ejemplo

Enlaces de
interés

[Introducción](#)

[Ejemplo](#)

[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.

	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	



Enlaces de interés

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

¿Qué es un puntero?

Ejemplo

Sintaxis

Arrays y punteros

Ejemplo

Formas de recorrer un
array

Jugando con Punteros

¿Qué es un puntero?

Punteros

¿Qué son?

Ejemplo
Sintaxis

Arrays y
punteros

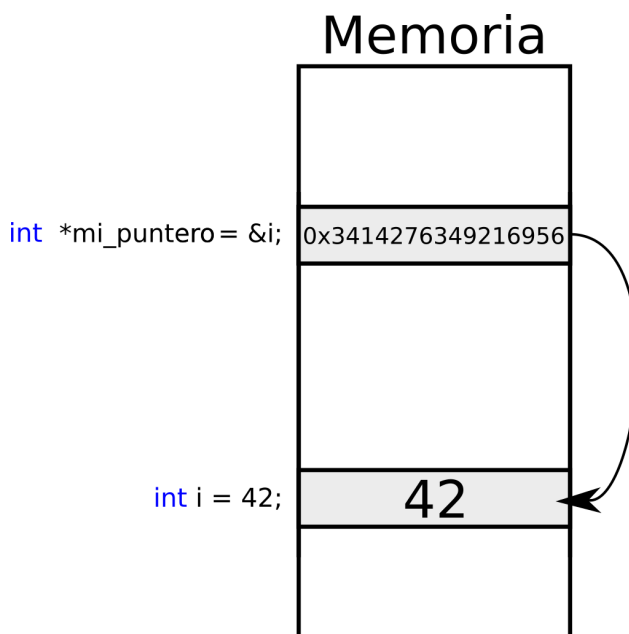
Ejemplo
Recorriendo
arrays

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**



¿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

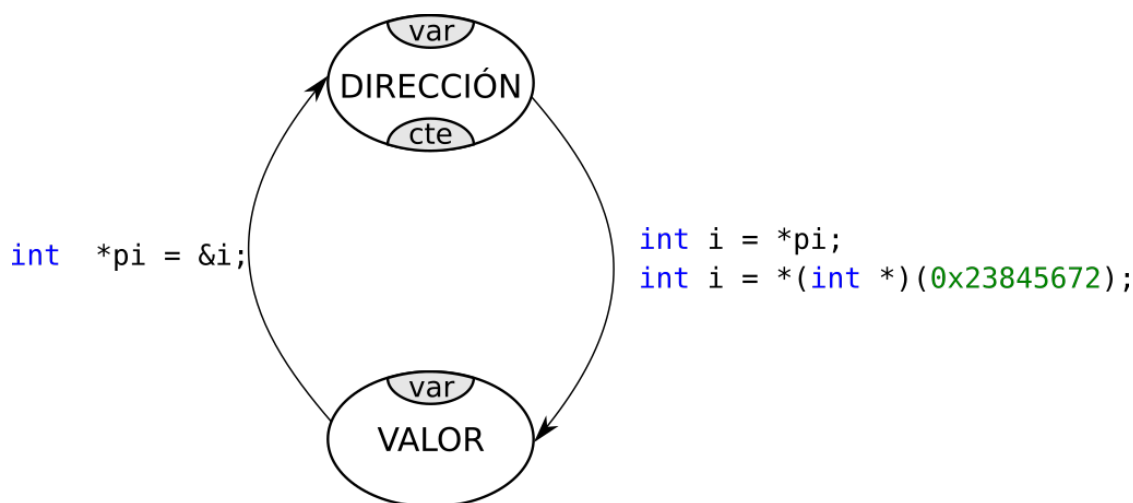
Sintaxis

Arrays y punteros

Ejemplo

Recorriendo arrays

Jugando con Punteros



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 }
```

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

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
Correcciones
Aceptado
Network



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
Correcciones
Aceptado

GIT

Tema 8

Índice

- Git
 - Características
 - Distribuido VS Centralizado
 - Funcionamiento
- Cuenta en GitHub
 - Plan gratuito
 - Confirmar correo
- Fork de mi repositorio
 - Buscar mi repositorio
 - Fork
 - Clonar mi repositorio
- Flujo de trabajo
 - Crea y revisa tus cambios
 - Sube los cambios a tu repositorio
 - Crea un nuevo pull request
 - Comprobar los cambios
 - Descripción
 - Solicitud terminada

GIT

Git

Características
Distribuido
VS
Centralizado
Funcionamiento

Cuenta en
GitHub
Plan gratuito
Confirmar
correo

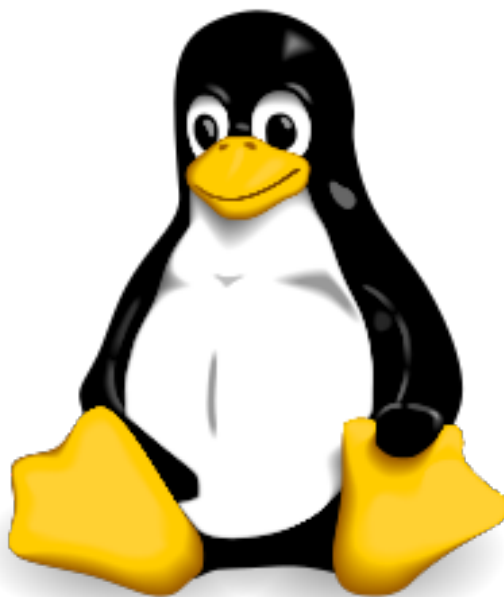
Fork de mi
repositorio
Buscar mi
repositorio
Fork
Clonar mi
repositorio

Workflow

Cambios
git push
Pull request
Comprobación
Descripción
Solicitado
Revisiones
Correcciones
Aceptado
Network



git



Características

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
Correcciones
Aceptado
Network

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
Descripción
Solicitado
Revisiones
Correcciones
Aceptado
Network

Funcionamiento

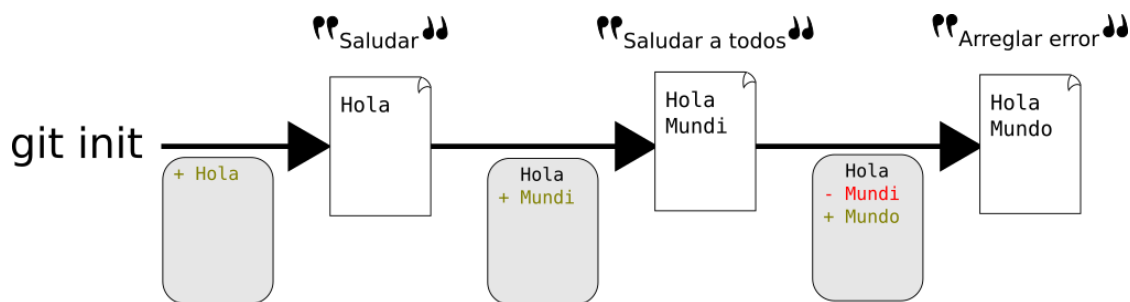
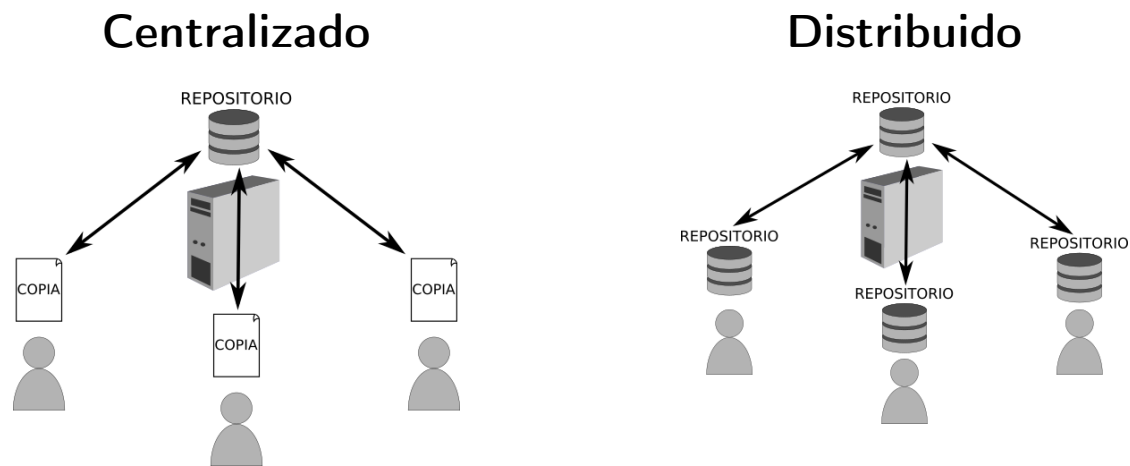
GIT

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
Correcciones
Aceptado
Network



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

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

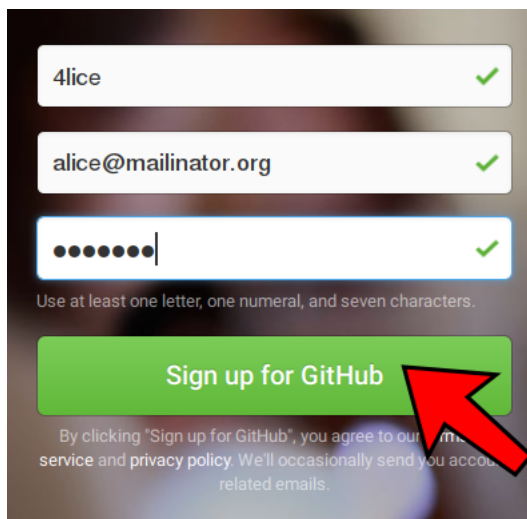
Solicitado

Revisiones

Correcciones

Aceptado

Network



Plan gratuito

GIT

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

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

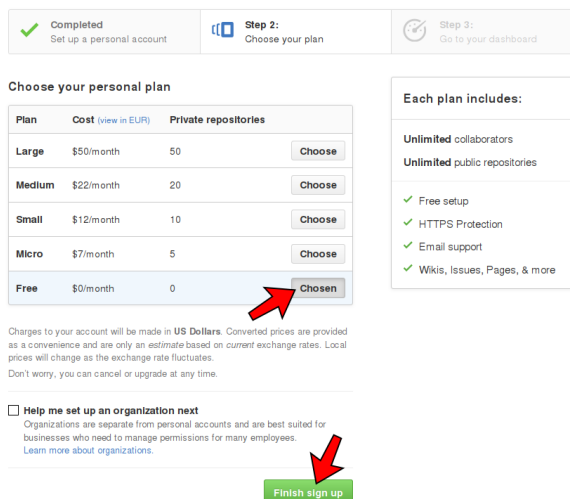
Solicitado

Revisiones

Correcciones

Welcome to GitHub

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



Plan	Cost (view in EUR)	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

Confirmar correo

GIT

Debemos confirmar la dirección de correo

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
Correcciones
Aceptado
Network



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

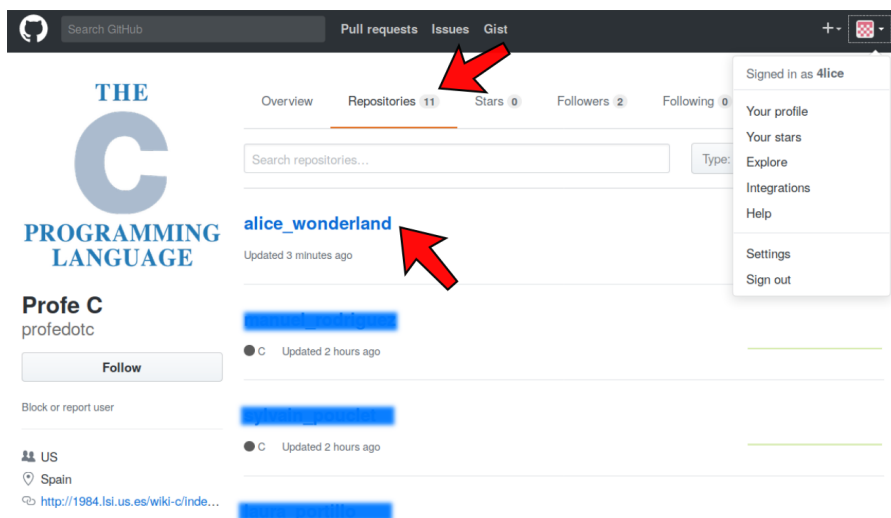
Entramos en la cuenta del profesor (github.com/profedotc), y
en la pestaña “*Repositories*” buscamos el repositorio que tenga
nuestro nombre y apellido

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
Correcciones



Fork

GIT

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

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
Correcciones
Aceptado
Network

GIT

Clonar mi repositorio

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

Git

Características
Distribuido
VS
Centralizado
Funcionamiento

Cuenta en
GitHub

Plan gratuito
Confirmar
correo

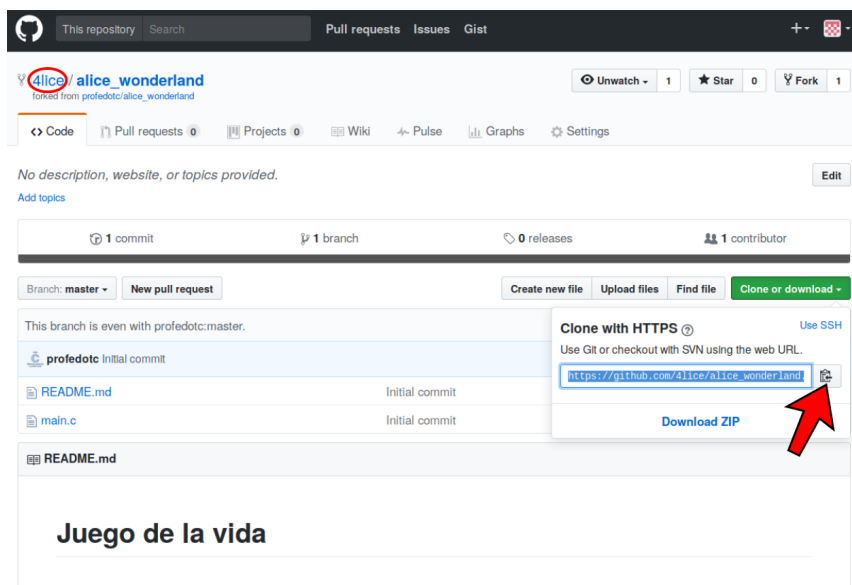
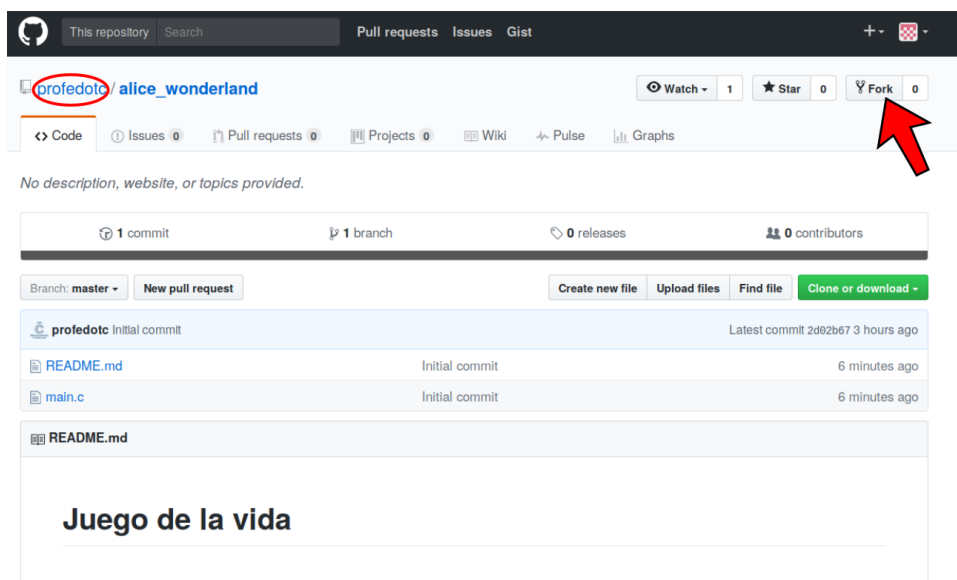
Fork de mi
repositorio

Buscar mi
repositorio

Fork
Clonar mi
repositorio

Workflow

Cambios
git push
Pull request
Comprobación
Descripción
Solicitado
Revisiones
Correcciones



Para clonar nuestro repositorio abrimos un terminal, navegamos hasta la carpeta dónde lo queremos clonar y ejecutamos el siguiente

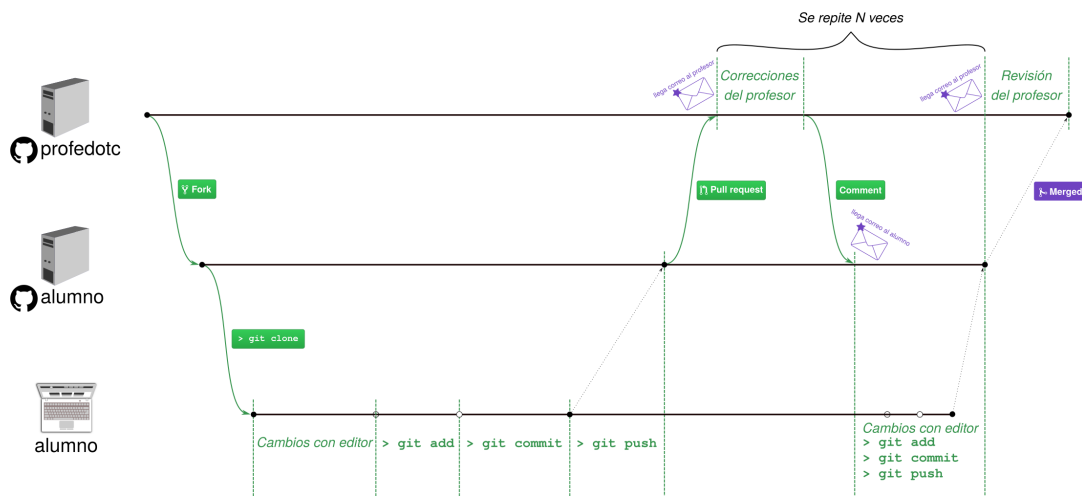
GIT

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
 Correcciones
 Aceptado
 Network



Crea y revisa tus cambios

GIT

> git diff

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
 Correcciones

```

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
14 void world_init(/* Recibo un mundo */);
15 -void world_print(/* Recibo un mundo */);
16 +void world_print(bool w[W_SIZE_X][W_SIZE_Y]);
17 void world_step(/* Recibo dos mundos */);
18 int world_count_neighbors(/* Recibo un mundo y unas coordenadas */);
19 ;
20 bool world_get_cell(/* Recibo un mundo y unas coordenadas */);
  
```

```

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];
  
```

Sube los cambios a tu repositorio

GIT

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

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
Correcciones
Aceptado
Network

GIT

Crea un nuevo pull request

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

Git

Características
Distribuido
VS
Centralizado
Funcionamiento

Cuenta en
GitHub

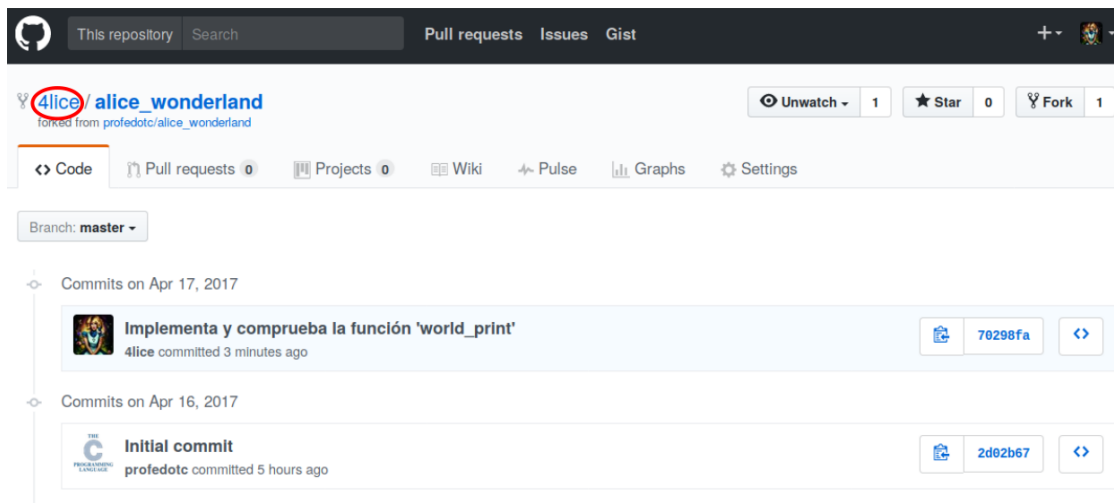
Plan gratuito
Confirmar
correo

Fork de mi
repositorio

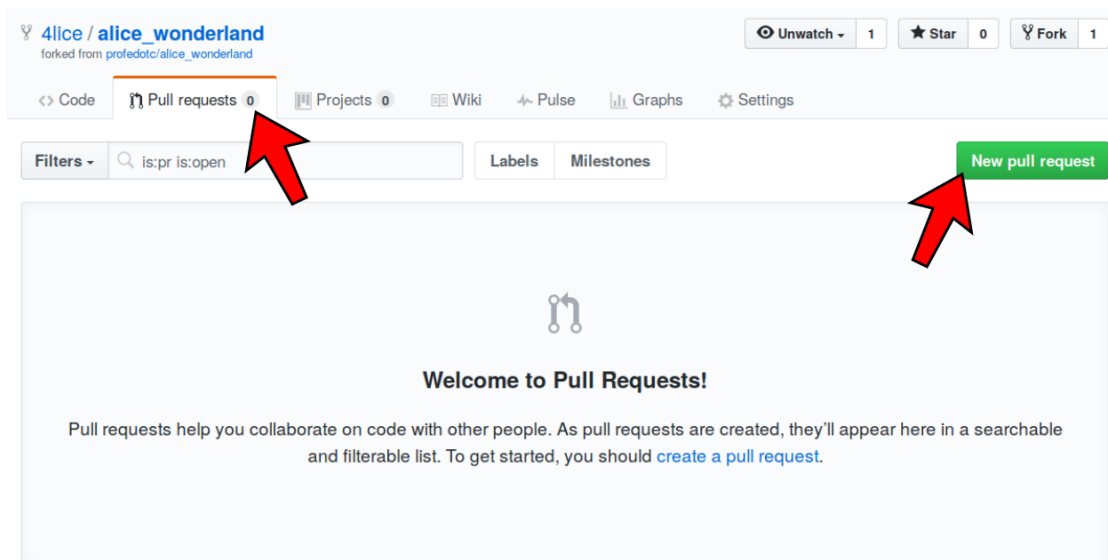
Buscar mi
repositorio
Fork
Clonar mi
repositorio

Workflow

Cambios
git push
Pull request
Comprobación
Descripción
Solicitado
Revisiones
Correcciones



This screenshot shows the 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. The most recent commit is titled 'Implementa y comprueba la función 'world_print'' by '4lice', committed 3 minutes ago. Below it is the 'Initial commit' by 'profedotc', committed 5 hours ago. The page includes navigation tabs for Code, Pull requests, Projects, Wiki, Pulse, Graphs, and Settings.



This screenshot shows the GitHub Pull Requests page for the '4lice / alice_wonderland' repository. The 'Pull requests' tab is selected and highlighted with a red arrow. A red arrow also points to the 'New pull request' button in the top right corner. The main content area displays a 'Welcome to Pull Requests!' message, explaining that pull requests help collaborate on code and that they will appear in a searchable and filterable list. A link to 'create a pull request' is provided.

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
Correcciones
Aceptado
Network

GIT

Descripción

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

Git

Características
Distribuido
VS
Centralizado
Funcionamiento

Cuenta en
GitHub

Plan gratuito
Confirmar
correo

Fork de mi
repositorio

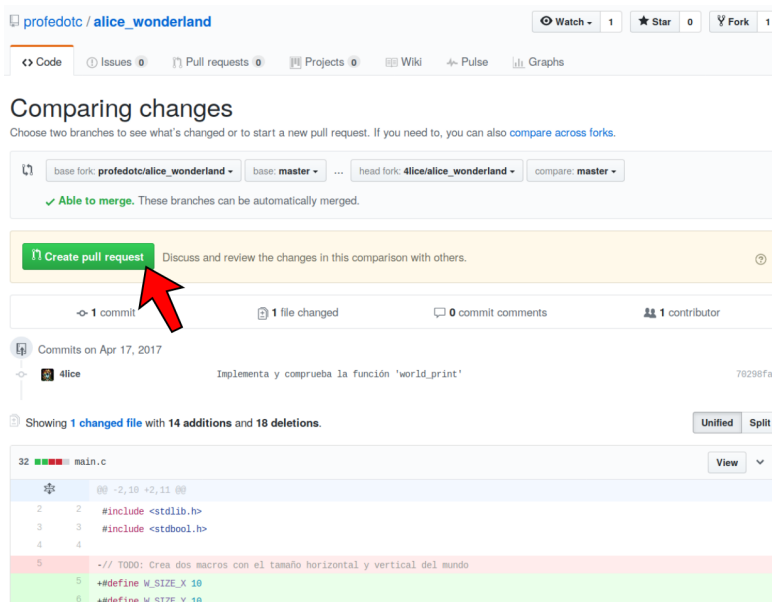
Buscar mi
repositorio

Fork
Clonar mi
repositorio

Workflow

Cambios
git push
Pull request
Comprobación

Descripción
Solicitado
Revisiones
Correcciones



profedotc / alice_wonderland

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

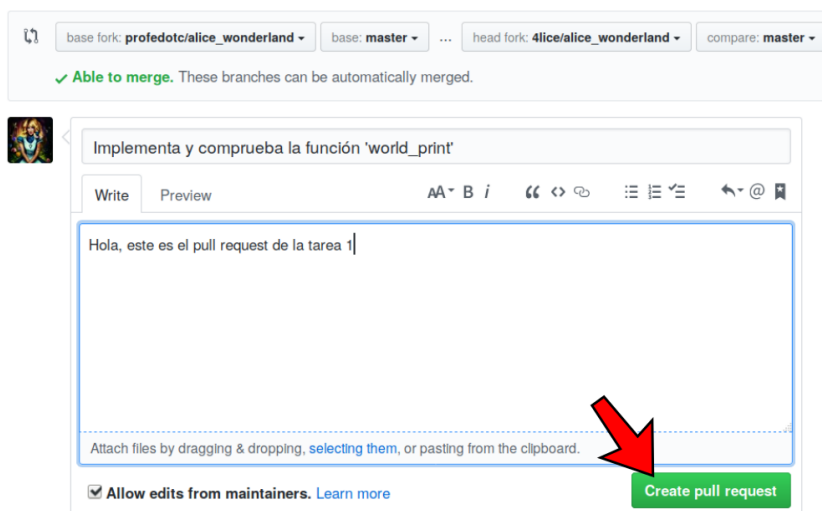
4lice Implementa y comprueba la función 'world_print' 78298fa

Showing 1 changed file with 14 additions and 18 deletions.

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

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:

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
Correcciones
Aceptado
Network

GIT

Esperar las revisiones del profesor

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

GIT

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

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
Correcciones
Aceptado
Network

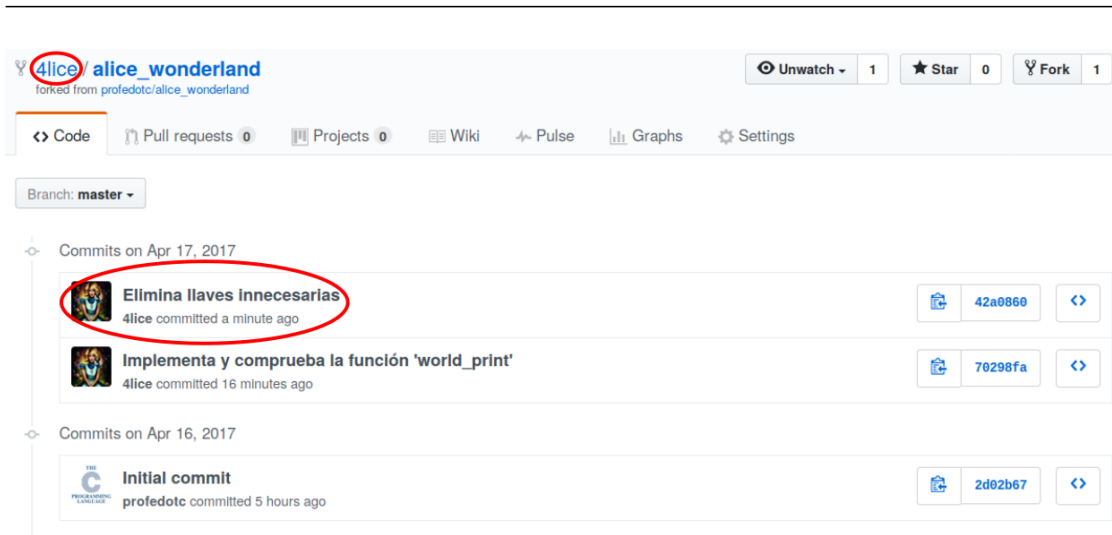
GIT

Git
Características
Distribuido
VS
Centralizado
Funcionamiento

Cuenta en
GitHub
Plan gratuito
Confirmar
correo

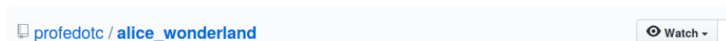
Fork de mi
repositorio
Buscar mi
repositorio
Fork
Clonar mi
repositorio

Workflow
Cambios
git push
Pull request
Comprobación
Descripción
Solicitado
Revisiones
Correcciones

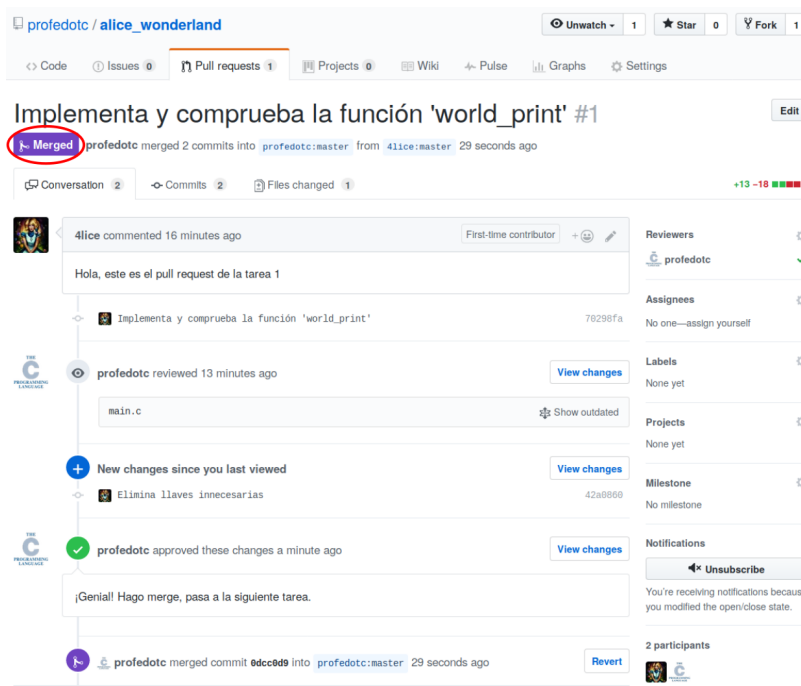


En

la página del pull request deben aparecer los nuevos commits automáticamente



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
Correcciones
Aceptado
Network

Estructuras

Struct
Alineación y
tamaño
Anidamiento
Anónimas
Arrays y
punteros

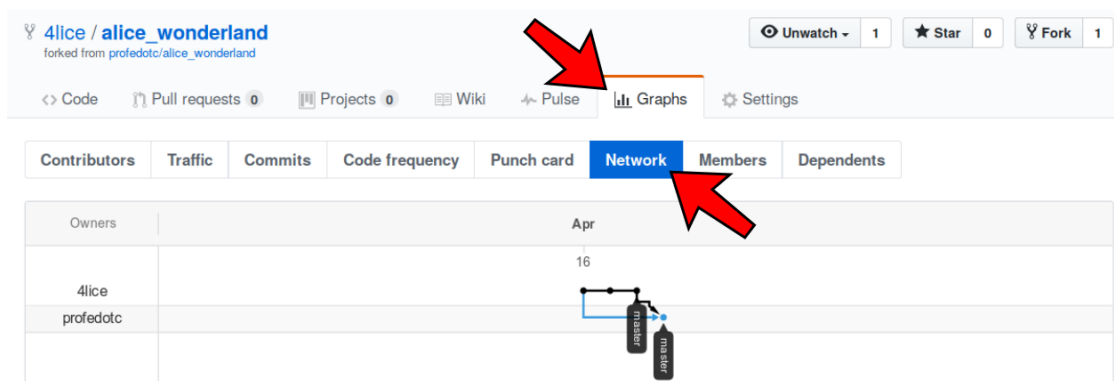
Union
Ejemplo 1
Ejemplo 2

Campos de
bits
Ejemplo

Enumerados
Macros
Ejemplos
Enum
Ejemplo

Inicialización

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



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

Struct

- Alineación y tamaño

- Anidamiento

- Estructuras anónimas

- Arrays y punteros

Union

- Ejemplo 1

- Ejemplo 2

Campos de bits

- Ejemplo

Enumerados

- Macros: El preprocesador de C

- Ejemplos

Enum

- Ejemplo

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 }
```

Estructuras

Struct

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

Union

- Ejemplo 1
- Ejemplo 2

Campos de bits

Ejemplo

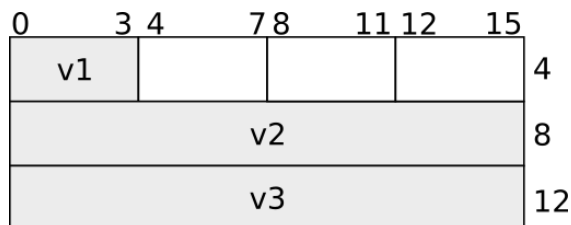
Enumerados

- 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

Una unión es un valor que tiene varias representaciones o formatos

Struct

Alineación y tamaño
Anidamiento
Anónimas
Arrays y punteros

Union

Ejemplo 1
Ejemplo 2

Campos de bits

Ejemplo

Enumerados

Macros
Ejemplos

Enum
Ejemplo

Inicialización

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\n");
32         break;
33     };
34
35     return 0;
36 }

```

Campos de bits

Estructuras

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

Alineación y tamaño
Anidamiento
Anónimas
Arrays y punteros

Union

Ejemplo 1
Ejemplo 2

Campos de bits

Ejemplo

Enumerados

Macros

Ejemplos

Enum

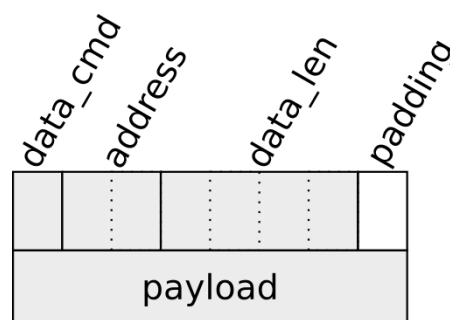
Ejemplo

Inicialización

```

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 }
```

Enum

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	1	<code>#include <stdio.h></code>	21	
Alineación y tamaño	2		22	<code>switch (choice) {</code>
Anidamiento	3	<code>struct person {</code>	23	<code>case NAME:</code>
Anónimas	4	<code>char name[256];</code>	24	<code>printf("%s\n", p.name);</code>
Arrays y punteros	5	<code>char surname[256];</code>	25	<code>break;</code>
Union	6	<code>unsigned char age;</code>	26	<code>case SURNAME:</code>
Ejemplo 1	7	<code>unsigned int phone;</code>	27	<code>printf("%s\n", p.surname);</code>
Ejemplo 2	8	<code>};</code>	28	<code>break;</code>
Campos de bits	9		29	<code>case AGE:</code>
Ejemplo	10	<code>enum person_attr {</code>	30	<code>printf("%d\n", p.age);</code>
Enumerados	11	<code>NAME,</code>	31	<code>break;</code>
Macros	12	<code>SURNAME,</code>	32	<code>case PHONE:</code>
Ejemplos	13	<code>AGE,</code>	33	<code>printf("%d\n", p.phone);</code>
Enum	14	<code>PHONE</code>	34	<code>break;</code>
Ejemplo	15	<code>};</code>	35	<code>default:</code>
Inicialización	16		36	<code>printf("error: %s:%d",</code>
	17	<code>int main ()</code>	37	<code>__FILE__, __LINE__);</code>
	18	<code>{</code>	38	<code>}</code>
	19	<code>person p = {"Alice", "Smith",</code>	39	<code>return 0;</code>
	20	<code>25, 12434321};</code>	40	<code>}</code>
		<code>enum person_attr choice =</code>		
		<code>NAME;</code>		

Designated initializers

Estructuras

Struct	1	<code>struct bug {</code>
Alineación y tamaño	2	<code>unsigned int legs;</code>
Anidamiento	3	<code>unsigned char color[3];</code>
Anónimas	4	<code>const char *name;</code>
Arrays y punteros	5	<code>} bugs[30] = {</code>
Union	6	<code>[0] = {</code>
Ejemplo 1	7	<code>.legs = 6,</code>
Ejemplo 2	8	<code>.color = {100, 100, 100},</code>
Campos de bits	9	<code>.name = "ant",</code>
Ejemplo	10	<code>},</code>
Enumerados	11	<code>[1] = {</code>
Macros	12	<code>.legs = 8,</code>
Ejemplos	13	<code>.color = {[0] = 200},</code>
Enum	14	<code>.name = "spider",</code>
Ejemplo	15	<code>},</code>
Inicialización	16	<code>[2 ... 10] = { // Only GNU</code>
	17	<code>.legs = 100,</code>
	18	<code>.color = {[1] = 255},</code>
	19	<code>.name = "centipede",</code>
	20	<code>},</code>
	21	<code>};</code>



[C Modular](#)

Cabeceras

[Ejemplo](#)

Compilación

Make

[Estructura](#)

[Ejemplo](#)

C Modular

Tema 10



Índice

[C Modular](#)

Cabeceras

[Ejemplo](#)

Compilación

Make

[Estructura](#)

[Ejemplo](#)

[Cabeceras](#)
[Ejemplo](#)
[Compilación por bloques](#)

[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 }
```

Compilación por bloques

C Modular

Cabeceras
Ejemplo
Compilación
Make
Estructura
Ejemplo

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:

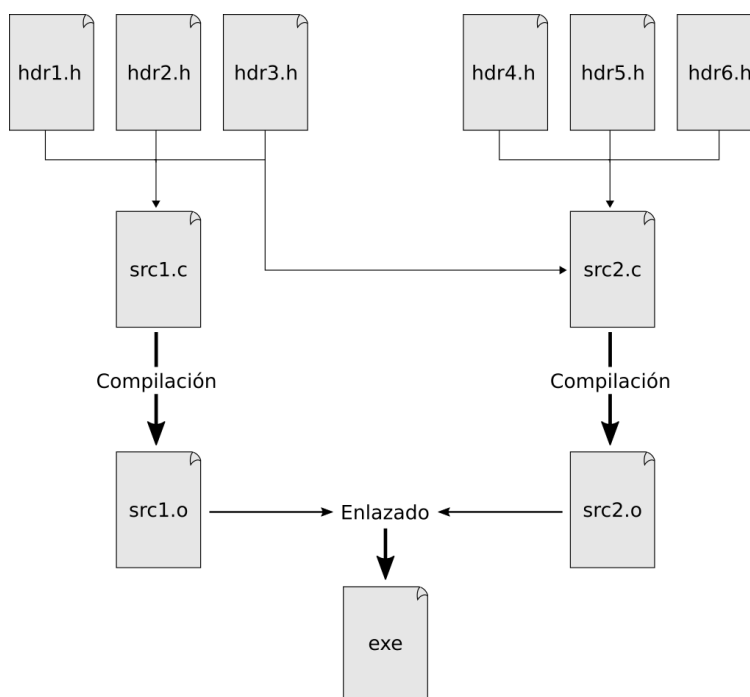
Dividir el código en bloques lógicos es una buena práctica
Mejora:

- El mantenimiento
- La legibilidad
- La portabilidad
- La escalabilidad
- etc

Compilación por bloques

C Modular

Cabeceras
Ejemplo
Compilación
Make
Estructura
Ejemplo



Compilación por bloques

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

Estructura

C Modular

Cabeceras
Ejemplo
Compilación
Make
Estructura
Ejemplo

objetivo: prerequisito1 prerequisito2 ...
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

Ejemplo

C Modular

Cabeceras
Ejemplo
Compilación
Make
Estructura
Ejemplo

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



Índice

[Dyn memory](#)

[Mapa de memoria](#)
[Ejemplo](#)

Mapa de memoria

Ejemplo

Stack
Overflow

[Stack Overflow](#)

Fragmentación

[Fragmentación](#)

arrays
multiD

Forma 1

Forma 2

[Reserva de arrays multidimensionales](#)

[Forma 1 \(la mala\)](#)

[Forma 2 \(la buena\)](#)

Uso tras liberación

[Uso tras liberación](#)

valgrind
Ejemplo

[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**

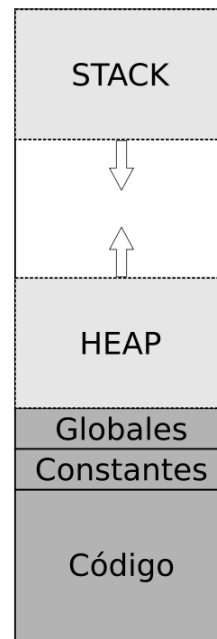
Stack: Se almacenan las **variables locales** de cada llamada a función

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 }

```

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

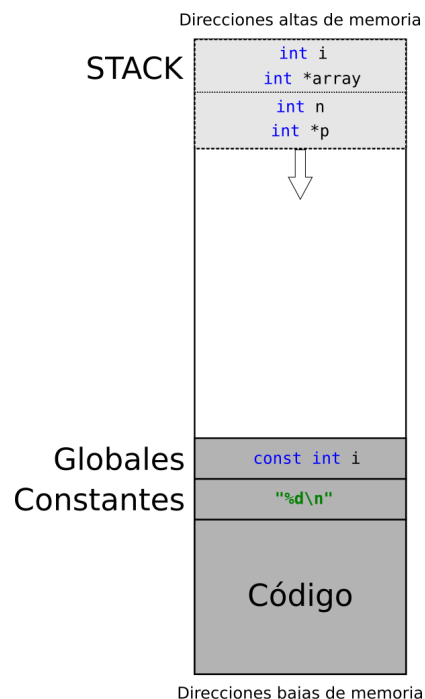
valgrind

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

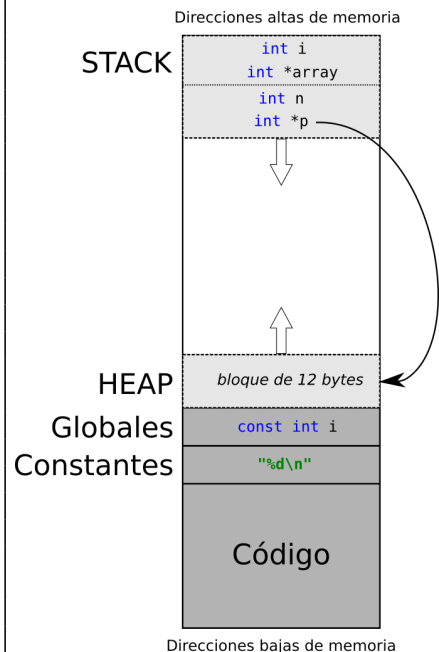
valgrind

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

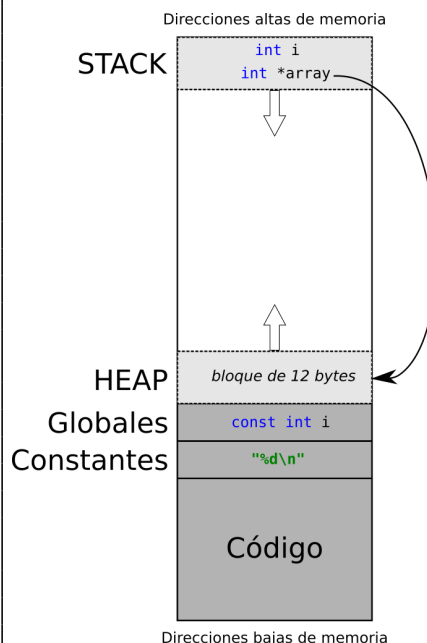
Forma 1
Forma 2

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

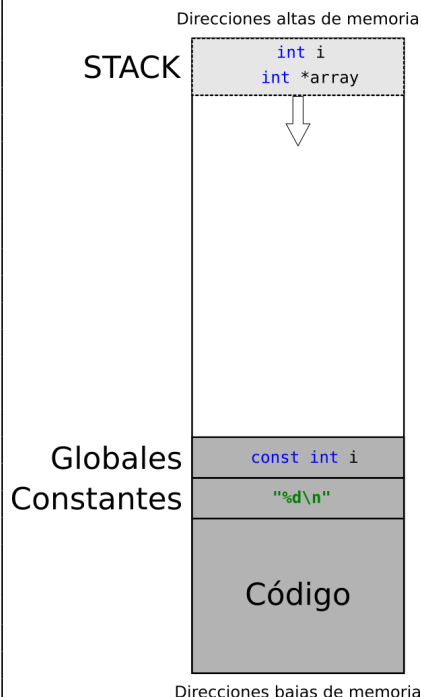
Forma 1
Forma 2

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 }
    
```



Stack Overflow

Dyn memory

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

Mapa de memoria

Ejemplo

Stack Overflow

Fragmentación

arrays

multiD

Forma 1

Forma 2

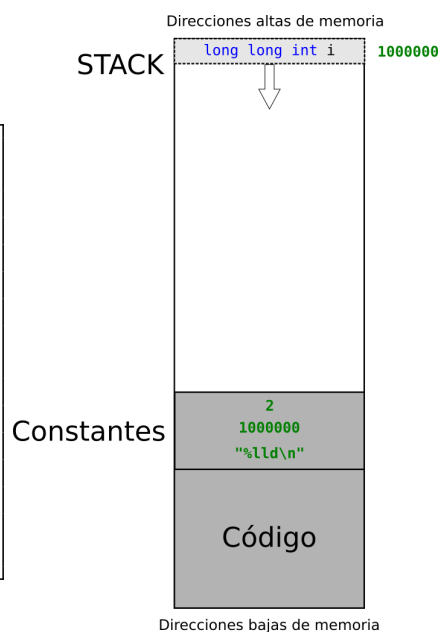
Uso tras liberación

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 }
    
```



Stack Overflow

Dyn memory

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

Mapa de memoria

Ejemplo

Stack Overflow

Fragmentación

arrays

multiD

Forma 1

Forma 2

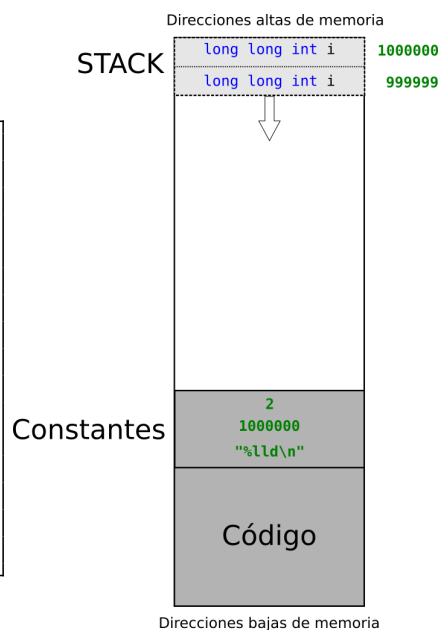
Uso tras liberación

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 }
    
```



Stack Overflow

Dyn memory

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

Mapa de memoria

Ejemplo

Stack Overflow

Fragmentación

arrays multiD

Forma 1

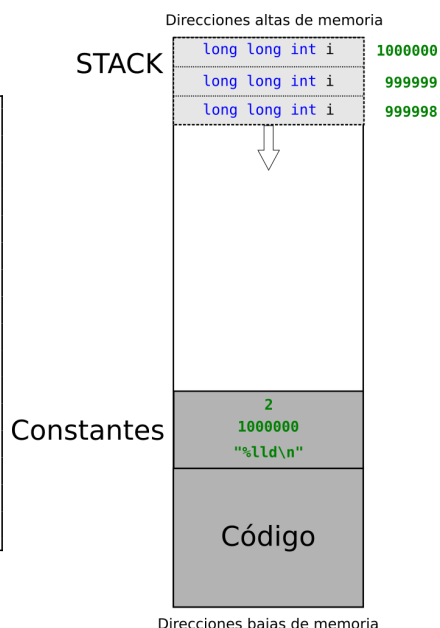
Forma 2

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 }
    
```



Stack Overflow

Dyn memory

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

Mapa de memoria

Ejemplo

Stack Overflow

Fragmentación

arrays multiD

Forma 1

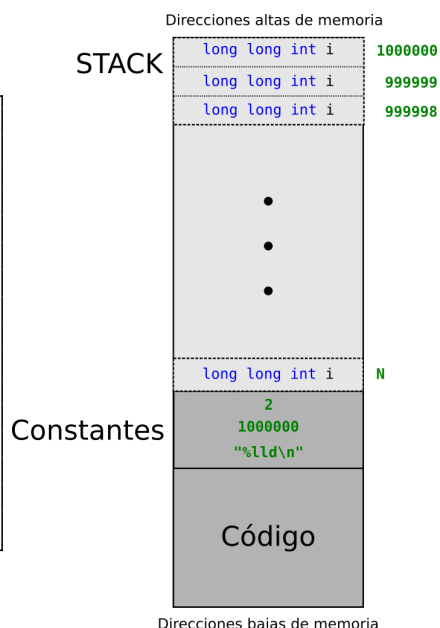
Forma 2

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 }
    
```



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

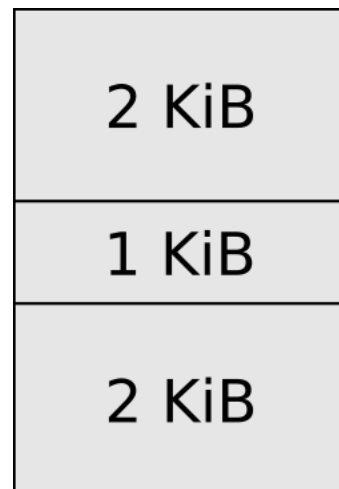
valgrind

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("%d\n", (long int)p1 / 1024);
13    printf("%d\n", (long int)p2 / 1024);
14    printf("%d\n", (long int)p3 / 1024);
15
16    free(p2);
17    p = malloc(2 * 1024);
18
19    printf("\n%d\n", (long int)p / 1024);
20
21    return 0;
22 }

```



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("%d\n", (long int)p1 / 1024);
13    printf("%d\n", (long int)p2 / 1024);
14    printf("%d\n", (long int)p3 / 1024);
15
16    free(p2);
17    p = malloc(2 * 1024);
18
19    printf("\n%d\n", (long int)p / 1024);
20
21    return 0;
22 }

```



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

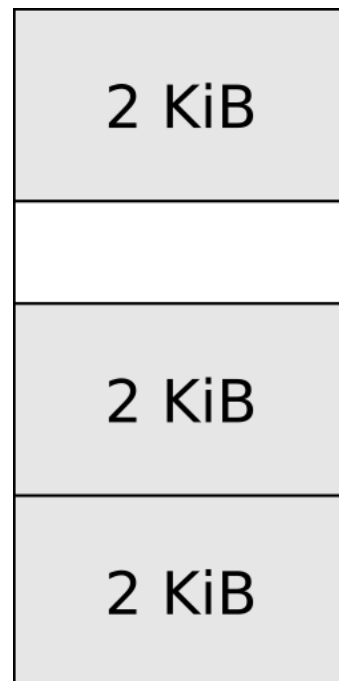
Uso tras liberación

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("%d\n", (long int)p1 / 1024);
13    printf("%d\n", (long int)p2 / 1024);
14    printf("%d\n", (long int)p3 / 1024);
15
16    free(p2);
17    p = malloc(2 * 1024);
18
19    printf("\n%d\n", (long int)p / 1024);
20
21    return 0;
22 }
```



Forma 1 (la mala)

Dyn memory

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

Mapa de memoria

Ejemplo

Stack Overflow

Fragmentación

arrays

multiD

Forma 1

Forma 2

Uso tras liberación

valgrind

Ejemplo

```

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 < COLS; j++)
14            array[i][j] = i * 10 + j;
15
16    /* Imprimimos */
17    for (i = 0; i < ROWS; i++) {
18        for (j = 0; j < COLS; 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

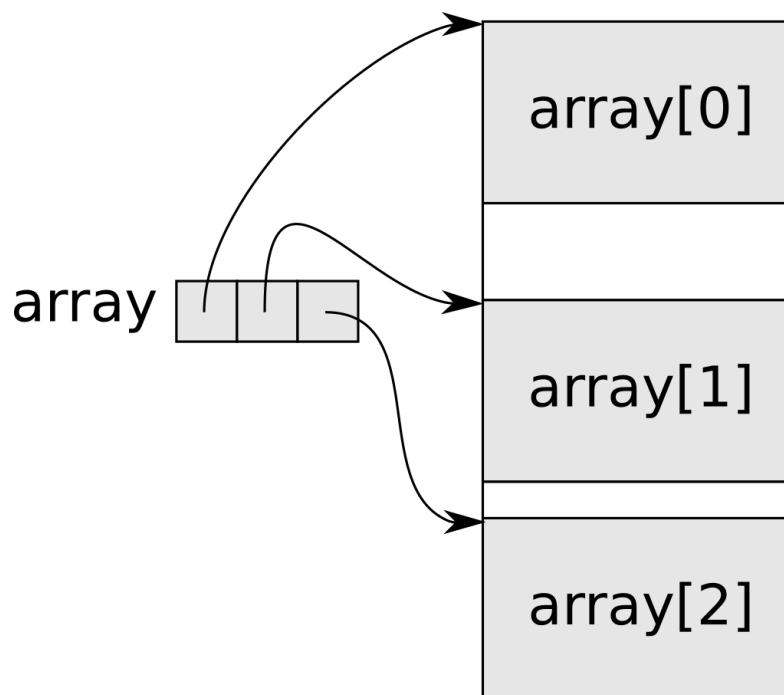
arrays
multiD

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:

Compilar con símbolos de depuración (`gcc -g`)
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

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

Mapa de memoria
Ejemplo

Stack
Overflow

Fragmentación

arrays
multiD

Forma 1
Forma 2

Uso tras liberación

valgrind
Ejemplo

```
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



Índice

Objetos

Qué es?

Ejemplo

Ocultación

flags

Reserva

Setters

Getters

Print

¿Qué es un objeto?

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

...

Reserva

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

Setters

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 }
```

Print

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



Índice

`main() args`

argc y argv

Ejemplo

getopt

getopt short

getopt long

argc y argv

Ejemplo

getopt

getopt short

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 }
```

argc y argv

Ejemplo

getopt

getopt short

getopt long

salida:

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

getopt

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

argc y argv

Ejemplo

getopt

getopt short

getopt long

```
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    }
```


getopt short

main() args

```
argc y argv 30     printf("bvalue = %d\n", bvalue);
Ejemplo     31     printf("cvalue = \"%s\"\n", cvalue);
            32
getopt      33     for (int i = optind; i < argc; i++)
getopt short 34         printf("Argumento desconocido \"%s\"\n", argv[i]);
            35
getopt long  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 1     #include <stdio.h>
Ejemplo     2     #include <stdlib.h>
            3     #include <stdbool.h>
getopt      4     #include <getopt.h>
            5
getopt short 6     int main(int argc, char **argv)
getopt long  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



Índice

Depuración

Introducción

GDB

Cómo utilizar
GDB

Comandos

Inicio
Breakpoints
Paso a paso
Estado del
programa
Otros
comandos
útiles

Introducción

GDB

Cómo utilizar GDB

Comandos

Inicio

Breakpoints

Paso a paso

Estado del programa

Otros comandos útiles

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](#)

Compilar con símbolos de depuración `gcc -g main.c -o miprog (-ggdb para símbolos específicos de gdb)`

Arrancar nuestro programa con gdb: `gdb miprog`

Utilizar los comandos de gdb para depurar



Comandos

[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
```

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
```

Breakpoints

Depuración

Introducción

GDB

Cómo utilizar GDB

Comandos

Inicio

Breakpoints

Paso a paso

Estado del programa

Otros comandos útiles

Otros comandos:

Listar los breakpoints (y watchpoints):

```
(gdb) info breakpoints
```

Eliminar breakpoint:

```
(gdb) delete <num que sale al listar>
```

Eliminar todos los breakpoint:

```
(gdb) delete breakpoints
```

Paso a paso

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

Ver una variable:
(gdb) print contador

Introducción

Ver una estructura:

GDB

(gdb) print *world

Cómo utilizar GDB

Ver el resultado de una expresión:

Comandos

(gdb) print size_x * size_y

Inicio

Breakpoints

Paso a paso

Estado del programa

Otros comandos útiles

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

Ver clases de comandos:

(gdb) help

Introducción

Ver comandos de un clase (por ejemplo, breakpoints):

GDB

(gdb) help breakpoints

Cómo utilizar GDB

Ver ayuda sobre un comando (por ejemplo, break):

(gdb) help break

Comandos

Inicio

Breakpoints

Paso a paso

Estado del programa

Otros comandos útiles

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



Entrada/Salida

Streams

Funciones

fopen
fwrite
fread
fputs
fgets
fprintf

Entrada/Salida

Tema 15



Índice

Entrada/Salida

Streams

¿Qué es un stream?

Funciones

fopen
fwrite
fread
fputs
fgets
fprintf

Funciones básicas

fopen
fwrite
fread
fputs
fgets
fprintf

¿Qué es un stream?

Entrada/Salida

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

Streams

Funciones

fopen
fwrite
fread
fputs
fgets
fprintf

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

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);
```

Streams

Funciones

fopen
fwrite
fread
fputs
fgets
fprintf

Para más información ver:

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

fopen

Entrada/Salida

Streams

Funciones

fopen
fwrite
fread
fputs
fgets
fprintf

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

Modos:

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

```
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
printf

```
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
printf

```
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

```
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 *fmt, ...);`

Streams

Funciones

fopen
fwrite
fread
fputs
fgets
fprintf

```
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 *fmt, ...);`

Streams

Funciones

fopen
fwrite
fread
fputs
fgets
fprintf

```
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

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.