



C Piscine

C 13

*Resumen: Este documento corresponde al enunciado del C 13 de la C Piscine de 42.*

*Versión: 5*

# Índice general

I.	Instrucciones	2
II.	Introducción	4
III.	Ejercicio 00 : btree_create_node	5
IV.	Ejercicio 01 : btree_apply_prefix	6
V.	Ejercicio 02 : btree_apply_infix	7
VI.	Ejercicio 03 : btree_apply_suffix	8
VII.	Ejercicio 04 : btree_insert_data	9
VIII.	Ejercicio 05 : btree_search_item	10
IX.	Ejercicio 06 : btree_level_count	11
X.	Ejercicio 07 : btree_apply_by_level	12
XI.	Entrega y evaluación	13

# Capítulo I

## Instrucciones

- Esta página será la única referencia: no te fíes de los rumores.
- ¡Ten cuidado! Los enunciados pueden cambiar en cualquier momento.
- Asegúrate de que tus directorios y archivos tienen los permisos adecuados.
- Debes respetar el procedimiento de entrega para todos tus ejercicios.
- Tus compañeros de piscina se encargarán de corregir tus ejercicios.
- Además de por tus compañeros, también serán corregidos por un programa que se llama la Moulinette.
- La Moulinette es muy estricta a la hora de evaluar. Está completamente automatizada. Es imposible discutir con ella sobre tu nota. Por lo tanto, sé extremadamente riguroso para evitar cualquier sorpresa.
- La Moulinette no tiene una mente muy abierta. No intenta comprender el código que no respeta la Norma. La Moulinette utiliza el programa **norminette** para comprobar La Norma en sus archivos. Entiende entonces que es estúpido entregar un código que no pase la **norminette**.
- Los ejercicios han sido ordenados con mucha precisión, del más sencillo al más complejo. En ningún caso se tendrá en cuenta un ejercicio complejo si no se ha conseguido realizar perfectamente un ejercicio más sencillo.
- El uso de una función prohibida se considera una trampa. Cualquier trampa será sancionada con la nota -42.
- Solamente hay que entregar una función `main()` si lo que se pide es un programa.
- La Moulinette compila con las flags `-Wall -Wextra -Werror` y utiliza `cc`.
- Si tu programa no compila, tendrás un 0.
- No puedes dejar en tu directorio ningún archivo que no se haya indicado de forma explícita en los enunciados de los ejercicios.
- ¿Tienes alguna pregunta? Pregunta a tu compañero de la derecha. Si no, prueba con tu compañero de la izquierda.

- Tu manual de referencia se llama `Google` / `man` / `Internet` / ...
- ¡No olvides participar en el slack de tu Piscina!
- Lee detenidamente los ejemplos. Podrían exigir cosas que no se especifican necesariamente en los enunciados...
- Razona. ¡Te lo suplico, por Thor, por Odín! Maldita sea.
- Para los siguientes ejercicios, utilizaremos la siguiente estructura:

```
typedef struct      s_btree
{
    struct s_btree  *left;
    struct s_btree  *right;
    void            *item;
} t_btree;
```

- Deberás incluir esta estructura en un archivo llamado `ft_btree.h` y entregarlo dentro de cada ejercicio.
- A partir del ejercicio 01, utilizaremos nuestro `btree_create_node`, así que tenlo en cuenta.



Para este último punto, tener el prototipo dentro del archivo `ft_btree.h` puede ser interesante...

# Capítulo II

## Introducción


Aquí tienes una lista de publicaciones de **Venom**:

- In League with Satan (single, 1980)
- Welcome to Hell (1981)
- Black Metal (1982)
- Bloodlust (single, 1983)
- Die Hard (single, 1983)
- Warhead (single, 1984)
- At War with Satan (1984)
- Hell at Hammersmith (EP, 1985)
- American Assault (EP, 1985)
- Canadian Assault (EP, 1985)
- French Assault (EP, 1985)
- Japanese Assault (EP, 1985)
- Scandinavian Assault (EP, 1985)
- Manitou (single, 1985)
- Nightmare (single, 1985)
- Possessed (1985)
- German Assault (EP, 1987)
- Calm Before the Storm (1987)
- Prime Evil (1989)
- Tear Your Soul Apart (EP, 1990)
- Temples of Ice (1991)
- The Waste Lands (1992)
- Venom '96 (EP, 1996)
- Cast in Stone (1997)
- Resurrection (2000)
- Anti Christ (single, 2006)
- Metal Black (2006)
- Hell (2008)
- Fallen Angels (2011)

Este proyecto puede resultar más sencillo si lo haces mientras escuchas **Venom**.

# Capítulo III

## Ejercicio 00 : btree\_create\_node


	Ejercicio: 00
btree_create_node	
Directorio de entrega: <i>ex00/</i>	
Archivos a entregar: <b>btree_create_node.c</b> , <b>ft_btree.h</b>	
Funciones autorizadas: <b>malloc</b>	

- Crea la función **btree\_create\_node** para crear un nuevo elemento. Deberá inicializar su **item** al valor del argumento, y el resto de elementos a 0.
- Se devuelve la dirección al nodo creado.
- Aquí tienes el prototipo de la función:

```
t_btree *btree_create_node(void *item);
```

# Capítulo IV

## Ejercicio 01 : btree\_apply\_prefix


	Ejercicio: 01
btree_apply_prefix	
Directorio de entrega: <i>ex01/</i>	
Archivos a entregar: <b>btree_apply_prefix.c</b> , <b>ft_btree.h</b>	
Funciones autorizadas: Ninguna	

- Crea una función **btree\_apply\_prefix** que ejecute la función dada como argumento al **item** de cada nodo, recorriéndolo de forma **preordinal**.
- Aquí tienes el prototipo:

```
void btree_apply_prefix(t_btree *root, void (*applyf)(void *));
```

# Capítulo V

## Ejercicio 02 : btree\_apply\_infix

	Ejercicio: 02
btree_apply_infix	
Directorio de entrega: <i>ex02/</i>	
Archivos a entregar: <b>btree_apply_infix.c</b> , <b>ft_btree.h</b>	
Funciones autorizadas: Ninguna	


- Crea una función **btree\_apply\_infix** que aplique la función dada como segundo argumento al **item** de cada nodo, recorriendo el arbol de forma **inordinal**.
- Aquí tienes el prototipo:

```
void btree_apply_infix(t_btree *root, void (*applyf)(void *));
```



# Capítulo VI

## Ejercicio 03 : btree\_apply\_suffix


	Ejercicio: 03
btree_apply_suffix	
Directorio de entrega: <i>ex03/</i>	
Archivos a entregar: <b>btree_apply_suffix.c</b> , <b>ft_btree.h</b>	
Funciones autorizadas: Ninguna	

- Crea la función **btree\_apply\_suffix** que ejecute la función dada como segundo argumento sobre cada **item** de cada nodo, recorriendo el árbol de forma **postordinal**.
- Aquí tienes el prototipo:

```
void btree_apply_suffix(t_btree *root, void (*applyf)(void *));
```

# Capítulo VII

## Ejercicio 04 : btree\_insert\_data


	Ejercicio: 04
btree_insert_data	
Directorio de entrega: <i>ex04/</i>	
Archivos a entregar: <b>btree_insert_data.c</b> , <b>ft_btree.h</b>	
Funciones autorizadas: <b>btree_create_node</b>	

- Crea una función **btree\_insert\_data** que inserte el elemento **item** dentro del árbol. El árbol pasado como argumento será ordenado: por cada **node** todos los elementos menores serán colocados a la izquierda y los elementos de igual o superior valor serán colocados a la derecha. Para comparar los valores, se pasará una función similar a **strcmp** como argumento.
- El parámetro **root** apunta al nodo raíz del árbol. La primera vez que se llame, deberá apuntar a **NULL**.
- Aquí tienes el prototipo:

```
void btree_insert_data(t_btree **root, void *item, int (*cmpf)(void *, void *));
```

# Capítulo VIII

## Ejercicio 05 : btree\_search\_item


	Ejercicio: 05
btree_search_item	
Directorio de entrega: <i>ex05/</i>	
Archivos a entregar: <b>btree_search_item.c</b> , <b>ft_btree.h</b>	
Funciones autorizadas: Ninguna	

- Crea una función **btree\_search\_item** que devuelva el primer elemento relacionado con el dato del puntero dado como referencia. Debes buscar en el árbol recorriéndolo de forma **inordinal**. Si el elemento no se encuentra, la función devolverá **NULL**.
- Aquí tienes el prototipo:

```
void *btree_search_item(t_btree *root, void *data_ref, int (*cmpf)(void *, void *));
```

# Capítulo IX

## Ejercicio 06 : btree\_level\_count


	Ejercicio: 06
	btree_level_count
	Directorio de entrega: <i>ex06/</i>
	Archivos a entregar: <b>btree_level_count.c</b> , <b>ft_btree.h</b>
	Funciones autorizadas: Ninguna

- Crea una función `btree_level_count` que devuelva la longitud de la rama más larga del árbol pasado como argumento.
- Aquí tienes el prototipo:

```
int btree_level_count(t_btree *root);
```

# Capítulo X

## Ejercicio 07 : btree\_apply\_by\_level

	Ejercicio: 07
btree_apply_by_level	
Directorio de entrega: <i>ex07/</i>	
Archivos a entregar: <b>btree_apply_by_level.c</b> , <b>ft_btree.h</b>	
Funciones autorizadas: <b>malloc</b> , <b>free</b>	

- Crea una función **btree\_apply\_by\_level** que ejecute la función, pasada como segundo argumento, a cada nodo del árbol. El árbol debe ser recorrido nivel por nivel. La función dada como argumento recibirá tres argumentos:
  - El primer argumento, de tipo **void \***, será el **item** del nodo actual.
  - El segundo argumento, de tipo **int**, será el nivel en el que nos encontremos: 0 para la raíz, 1 para los hijos, 2 para los nietos, etc.
  - El tercer argumento, de tipo **int**, será 1 si es el primer **node** del nivel, o 0 de otro modo.
- Aquí tienes el prototipo de la función:

```
void btree_apply_by_level(t_btree *root, void (*applyf)(void *item, int current_level, int is_first_element))
```

# Capítulo XI

## Entrega y evaluación

Entrega tu proyecto en tu repositorio `Git` como de costumbre. Solo el trabajo entregado en el repositorio será evaluado durante la defensa. No dudes en comprobar varias veces los nombres de los archivos para verificar que sean correctos.



Sólo necesitas entregar los archivos requeridos por el enunciado de este proyecto.