**Como hacer un MAKEFILE**

**RM** = rm –f // creamos variable RM que al llamarla ejecuta rm –f (remove force)

**NAME** = libft.a // damos nombre al proyecto

**HEADER** = libft.h // asociamos el encabezado del nombre de nuestra libreria.h (previamente creada)

**CFLAGS** = -Wall -Wextra -Werror // variable con los flags de compilación (los indicados según ejercicio)

// creamos variable SRC (source) donde la equipamos con los archivos.c que deseamos aplicar las reglas de nuestro makefile

**SRC** = ft\_isalpha.c ft\_isprint.c ft\_memset.c ft\_bzero.c ft\_isascii.c ft\_memcpy.c \ // ( \ salto línea)

ft\_strlen.c ft\_isalnum.c ft\_isdigit.c ft\_memmove.c ft\_strlcpy.c ft\_strlcat.c \

**OBJ** = $(SRC:.c=.o) // la variable OBJ, llama a la variable SRC y le indica que debe pasar los arhivos .c a .o

**INCLUDE** = -I ./ // en esta variable le indicamos la ubicación de los archivos, (./ actual)

all: $(NAME) // esta es la primera regla que el comando make ejecuta, importante que se coloque la primera

%.o: %.c $(HEADER) // esto indica la forma como compilar, que librería usar y la ubicación de los archivos

$(CC) $(CFLAGS) $(INCLUDE) -c $< -o $@ // -c $< -o $@ son variables automáticas

$(NAME): $(OBJ) $(HEADER) // NAME, sus dependencias son OBJ y HEADER, si hay cambios se ejecuta la acción

ar -rcs $(NAME) $(OBJ) // comprime los archivos

.PHONY: all clean fclean re // .PHONY se usa para no confunfir el comando clean con la posibilidad de que exista un archivo llamado con el mismo nombre. Este comando se puede poner en cualquier parte del makefile, pero suele ponerse antes del clean

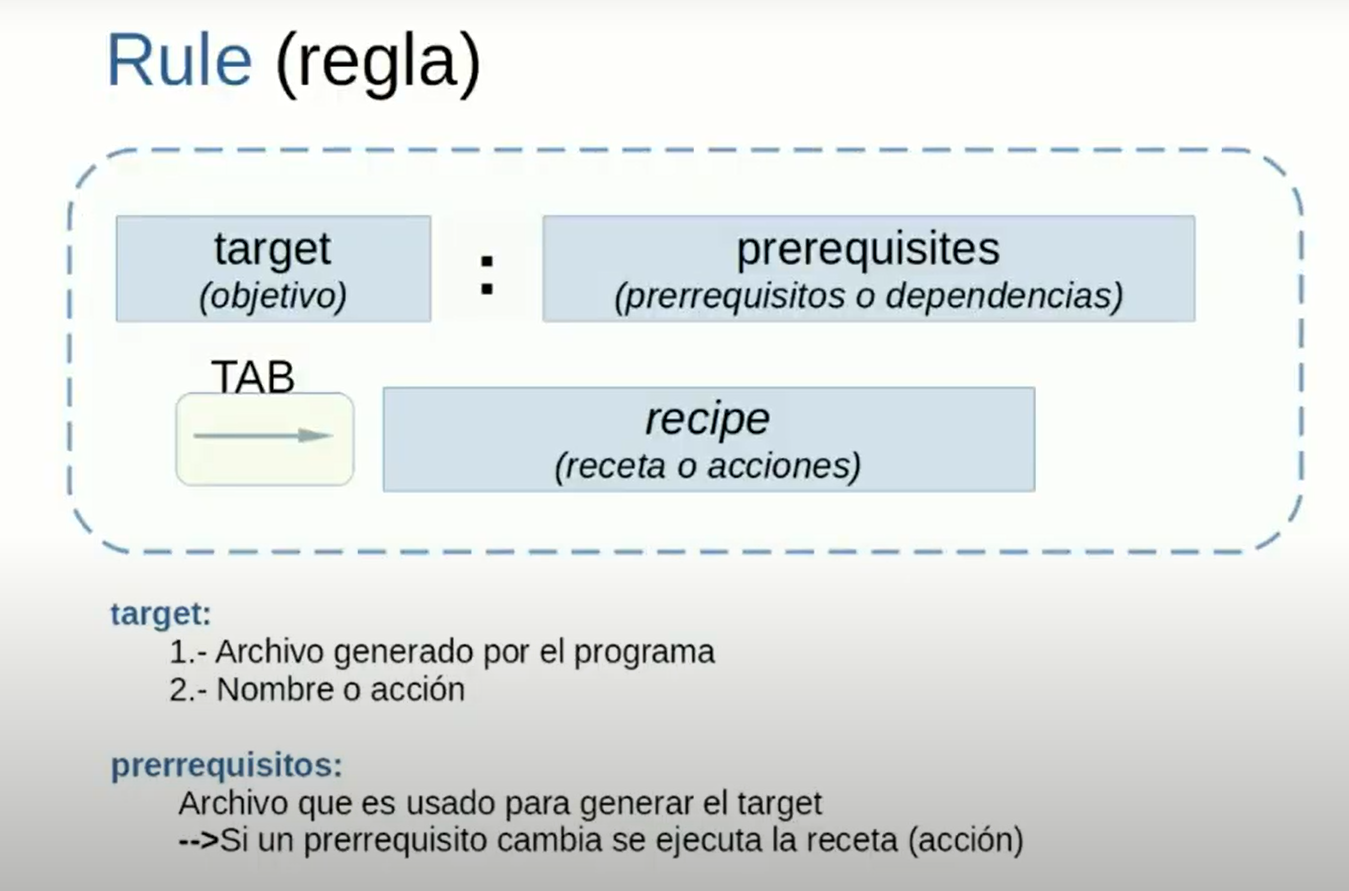
clean: // clean borra los archivos .o

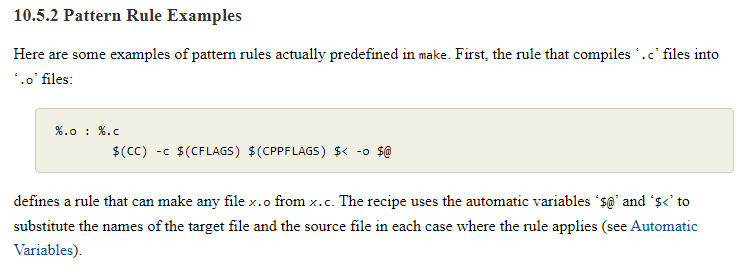
$(RM) $(OBJ)

fclean: clean // fclean, llama a clean, por lo que borra los archivos .o y el archivo .a

$(RM) $(NAME)

re : fclean all





NOTA : las variables se indican con = , mientras que las reglas con : , las llamadas a varibles se puede hacer con () o {}

El archivo make file se puede llamar de la siguiente manera:

GNUmakefile , makefile , Makefile (Make buscara dicho archivo siguiendo este mismo orden)

Links con info:

<https://www.youtube.com/watch?v=0XlVyZAfQEM> (muy bien explicado y en Español)

<https://www.gnu.org/software/make/manual/make.html> (manual oficial de make)

<http://www.it.uc3m.es/~pedmume/asignaturas/2005/LAO/Lab2/tutorial4/node4.html>

**Como crear una LIBRERÍA**

#ifndef LIBFT\_H // ifndef = if not define (si no existe, o si no está definida)

#define LIBFT\_H // define = entonces definela

-

// El cuerpo de la librería, donde incluimos todas las librerias que queramos usar en nuestro proyecto, más los prototipos de las funciones. También es posible definir nuevos valores a las variables.

-

-

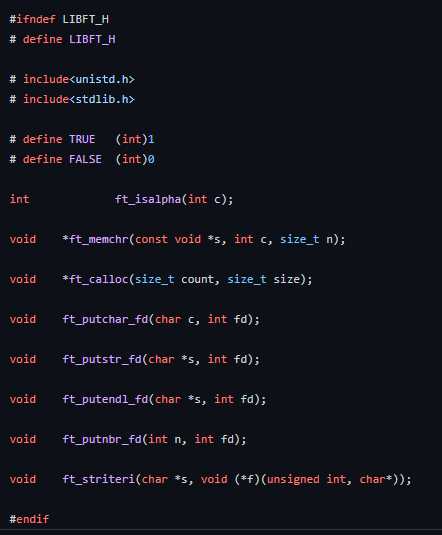
-

#endif // endif = fin de la definición

NOTA: usaremos nuestra librería como cabezera de nuestras funciones de la siguiente manera entre comillas;

#include “libft.h”

Ejemplo:



**isalpha**

**isalpha (c)** es una función en C que se puede usar para verificar si el carácter pasado es un alfabeto o no. Devuelve un valor distinto de cero si es un alfabeto; de lo contrario, devuelve 0. Por ejemplo, devuelve valores distintos de cero para 'a' a 'z' y 'A' a 'Z' y ceros para otros caracteres.

#include <stdio.h>

**int ft\_isalpha(int c)**

{

if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z'))

return (1);

return (0);

}

int main(void)

{

printf("%d\n", ft\_isalpha('h'));

printf("%d\n", ft\_isalpha('2'));

return (0);

}

**isdigit**

**isdigit (c)** comprueba si el carácter pasado es un dígito o no. Devuelve un valor distinto de cero si es un dígito; de lo contrario, devuelve 0.

#include <stdio.h>

**int ft\_isdigit(int c)**

{

if (c >= '0' && c <= '9')

return (1);

return (0);

}

int main(void)

{

printf("%d\n", ft\_isdigit('4'));

printf("%d\n", ft\_isdigit('L'));

return (0);

}

**isalnum**

**isalnum (c)** La función isalnum verifica si el parámetro dado es alfanumérico o no. Devoliendo 1 si es correcto y 0 sino.

#include <stdio.h>

**int ft\_isalnum(int c)**

{

if (ft\_isdigit(c) || ft\_isalpha(c))

return (1);

return (0);

}

int main(void)

{

printf("%d\n", ft\_isalnum('w'));

printf("%d\n", ft\_isalnum('2'));

printf("%d\n", ft\_isalnum('.'));

return (0);

}

**isascii**

**isascii (c)** La función isascii verifica si el parámetro dado es alfanumérico o no. Devoliendo 1 si es correcto y 0 sino.

#include <stdio.h>

**int ft\_isascii(int c)**

{

if (c >= 0 && c <= 127)

return (1);

return (0);

}

int main(void)

{

printf("%d\n", ft\_isascii(130));

printf("%d\n", ft\_isascii('2'));

printf("%d\n", ft\_isascii('?'));

return (0);

}

**isprint**

**isprint (c)** La función isprint verifica si el parámetro dado es imprimible

#include <stdio.h>

**int ft\_isprint(int c)**

{

if (c >= 32 && c <= 126)

return (1);

return (0);

}

int main(void)

{

printf("%d\n", ft\_isprint('h'));

printf("%d\n", ft\_isprint('2'));

return (0);

}

**strlen**

**strlen (c)** La función strlen calcula el número de caracteres de la cadena dada por parametro.

#include <stdio.h>

**size\_t ft\_strlen(const char \*s)** // size\_t y const char (mirar en pag. Conceptos)

{

size\_t i; // creamos una variable contador tipo size\_t

i = 0; // le damos un valor

while (s[i]) // iteramos la string por índice

++i; // usamos ++i para aumentar en 1 el valor de i hasta el final de la iteración

return (i); // devolvemos el valor final de i

}

int main(void)

{

const char s[]= "Barcelona"; // podemos crear la variable con el string

printf("%ld\n", ft\_strlen(s)); // y pasar el nombre de la variable a la función

printf("%ld\n", ft\_strlen("hola 42 Barcelona"));

printf("%ld\n", ft\_strlen("26")); // o bien pasar directamente el argumento entre comillas

return (0);

}

Nota: escribir while (s[i]) es lo mismo que escribir while (s[i] != '\0') { }

**memset**

**memset (c)** La función memset copia el valor de c (convertido a unsigned char) en cada uno de los primeros n caracteres en el objeto apuntado por b, size\_t len, le indica la cantidad de veces a copiar.

#include <stdio.h>

**void \*ft\_memset(void \*b, int c, size\_t len)**

{

unsigned char \*tmp;

tmp = (unsigned char \*)b;

while (len--)

\*tmp++ = c;

return (b);

}

int main(void)

{

char str[] = "Barcelona";

printf("%s\n", (char \*)ft\_memset(str, '$', 3));

printf("%s\n", (char \*)ft\_memset(str, '%', 5));

return (0);

}

**bzero**

**isascii (c)** La función isascii verifica si el parámetro dado es alfanumérico o no. Devoliendo 1 si es correcto y 0 sino.

**memcpy**

**isascii (c)** La función isascii verifica si el parámetro dado es alfanumérico o no. Devoliendo 1 si es correcto y 0 sino.

**memmove**

**isascii (c)** La función isascii verifica si el parámetro dado es alfanumérico o no. Devoliendo 1 si es correcto y 0 sino.

**strlcpy**

**isascii (c)** La función isascii verifica si el parámetro dado es alfanumérico o no. Devoliendo 1 si es correcto y 0 sino.

**strlcat**

**isascii (c)** La función isascii verifica si el parámetro dado es alfanumérico o no. Devoliendo 1 si es correcto y 0 sino.

**Itoa**

**Itoa ()** La función itoa convierte un entero a string.

Sintaxis:

char \*itoa(int n);

static int ft\_absval(int n)

{

if (n < 0)

n = -n;

return (n);

}

static int ft\_nbrlen(long n)

{

int size;

size = 0;

if (n == 0)

return (1);

else if (n < 0)

size++;

while (n)

{

n /= 10;

size++;

}

return (size);

}

char \*ft\_itoa(int n)

{

char \*str;

int size;

size = ft\_nbrlen(n);

str = (char \*)malloc(sizeof(char) \* size + 1);

if (!str)

return (NULL);

str[size] = '\0';

if (n < 0)

str[0] = '-';

if (n == 0)

str[0] = '0';

while (n)

{

--size;

str[size] = ft\_absval(n % 10) + '0';

n /= 10;

}

return (str);

}

**substr**

**substr()** La función substr devuelve una subcadena especificada de una cadena.

Sintaxis:

char \*substr(char const \*s, unsigned int start, size\_len);

**SUBSTR**(*cadena*; *inicio*; *longitud*)

## Parámetros

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Tipo** | **Descripción** |
| *cadena* | carácter | El campo, la expresión o el valor literal de donde extraer la subcadena. |
| *inicio* | numérico | La posición de carácter inicial de la subcadena.  Las posiciones numéricas de los caracteres de la *cadena* comienzan en 1. Para extraer una subcadena que comienza con C de la cadena ABCDEF, debe especificar un valor de *inicio* de 3. |
| *longitud* | numérico | El número de caracteres en la subcadena.  Si la *longitud* es 0, la salida aparece en blanco. |

**CONCEPTOS:**

**size\_t** **size\_t** es un alias para unsigned long.

El tipo de datos **size\_t** nunca es negativo. Por lo tanto, muchas funciones de la biblioteca C como malloc, memcpy y strlen declaran sus argumentos y devuelven el tipo como **size\_t** .

size\_t es un tipo que puede contener cualquier índice de Array.

Dependiendo de la implementación, puede ser cualquiera de:

unsigned char

unsigned short

unsigned int

unsigned long

unsigned long lo

size\_t es un tipo sin signo. Por lo tanto, no puede representar ningún valor negativo (<0). Se utiliza cuando se cuenta algo y se está seguro de que no puede ser negativo. Por ejemplo, [b](http://devdocs.io/c/string/byte/strlen) devuelve un size\_t porque la longitud de una cadena tiene que ser al menos 0.

[const](https://www.iteramos.com/etiquetada/const) char\* = es un mutable puntero a un inmutable personaje/cadena. No puedes cambiar el contenido de la(s) ubicación(es) a la que apunta este puntero. Además, los compiladores están obligados a dar mensajes de error cuando lo intentas. Por la misma razón, la conversión de const char \* a char\* está desaprobado.

char\* const = es un inmutable puntero (no puede apuntar a ningún otro lugar) pero el contenido del lugar al que apunta mutable .

const char\* const = es un inmutable puntero a un inmutable personaje/cadena.

char\* = es totalmente permisiva.

Fuente: https://www.iteramos.com/pregunta/30299/diferencia-entre-char-y-const-char