EXAM

42ZIP

NIVEL 1

* aff_a: Imprime "a" excepto si en el primer argumento no existe.

Assignment name : aff_a; Expected files : aff_a.c; Allowed functions: write.

Escribe un programa que tome un string y muestre el primer cáracter 'a' que encuentre en él, seguido de una nueva línea. si no hay caracteres 'a' en el string, el programa simplemente escribe una nueva línea.
Si el número de parámetros no es 1, el programa muestra 'a' seguido de una nueva línea.

Example:
$> ./aff_a "abc" | cat -e
a$
$> ./aff_a "dubO a POIL" | cat -e
a$
$> ./aff_a "zz sent le poney" | cat -e
$
$> ./aff_a | cat -e
a$
*/

```
##include <unistd.h>

//        Función que busca la letra 'a' en la cadena de caracteres.
//        Esta función toma un puntero a una cadena de caracteres (str) y busca la primera aparición
//  de la letra 'a'. Si encuentra 'a', la escribe en la salida estándar y termina la función.

void    aff_a(char *str)
{
   int i;

   i = 0;
   while (str[i]) // Recorre la cadena hasta encontrar 'a'.
   {
      if (str[i] == 'a')
      {
         write(1, "a", 1); // Escribe 'a' en la salida estándar.
         return ; // Termina la función cuando encuentra la primera 'a'.
      }
      i++;
   }
}
```

// Función principal que procesa los argumentos de la línea de comandos
// Esta función verifica si se ha pasado un único argumento al programa. Si no se pasa ningún argumento,
// Imprime 'a' por defecto. Si se pasa un argumento, llama a la función `aff_a` para buscar 'a' en ese argumento.

```
int main(int argc, char **argv)
{
   if (argc != 2) // Si el número de argumentos no es 1, imprime 'a' por defecto.
      write(1, "a", 1);
   else
      aff_a(argv[1]); // Llama a la función para buscar 'a' en el argumento.

   write(1, "\n", 1); // Agrega una nueva línea al final de la salida.
   return (0);
}
```

****

* aff_z: imprime "z" si o si.

Assignment name : aff_z; Expected files : aff_z.c; Allowed functions: write.

Escribe un programa que tome una cadena y muestre la primera 'z' carácter que encuentra en él, seguido de una nueva línea. si no hay caracteres 'z' en la cadena, el programa escribe 'z' seguido por una nueva línea. Si el número de parámetros no es 1, el programa muestra 'z' seguida de una nueva línea.
Example:

```
$> ./aff_z "abc" | cat -e
z$
$> ./aff_z "dubO a POIL" | cat -e
z$
$> ./aff_z "zaz sent le poney" | cat -e
z$
$> ./aff_z | cat -e
z$  */
```

```
#include <unistd.h>

// Programa que imprime el primer argumento de la línea de comandos
// seguido de un salto de línea. Si no se proporciona ningún argumento,
// simplemente imprime un salto de línea.

int main(void)
{
   // Escribe la letra 'z' seguida de una nueva línea en la salida estándar
   write(1, "z\n", 2);

   // Retorna 0 para indicar que el programa se ejecutó correctamente
   return (0);
}
```

****

NIVEL 2

* **rev_print**: Imprime el primer argumento en orden inverso.

name : rev_print; Expected files   : rev_print.c; Allowed functions: write

Escribe un programa que tome una cadena y la muestre al revés seguido de una nueva línea.
Si el número de parámetros no es 1, el programa muestra una nueva línea.

Examples:
$> ./rev_print "zaz" | cat -e
zaz$
$> ./rev_print "dub0 a POIL" | cat -e
LIOP a 0bud$
$> ./rev_print | cat -e
$
*/

```
#include <unistd.h>

// Función que imprime una cadena en orden inverso.
// Esta función toma un puntero a una cadena de caracteres (str) y escribe caracter de la cadena en la salida en orden inverso.

void    rev_print(char *str)
{
   int len;

   len = 0;
   while (str[len])          // Calcula la longitud de la cadena.
      len++;
   while (len > 0)           // Recorre la cadena en orden inverso.
   {
      len--;                 // Decrementa para acceder al último carácter primero.
      write(1, &str[len], 1);  // Escribe cada carácter en la salida estándar.
   }
}

// Función principal que procesa los argumentos de la línea de comandos.
// Esta función verifica si se ha pasado un único argumento al programa y, si es así, llama a la función `rev_print` para imprimir la cadena en orden inverso.

int main(int argc, char **argv)
{
   if (argc == 2)            // Verifica que haya exactamente un argumento
      rev_print(argv[1]);      // Llama a la función para procesar la cadena
   write(1, "\n", 1);          // Imprime una nueva línea al final
   return (0);               // Retorna 0 indicando ejecución exitosa
}

**********************************************************************
```

* **ft_putstr:** Función que imprime el primer argumento.

Assignment name : ft_putstr;Expected files : ft_putstr.c; Allowed functions: write

Escribe una función que muestre una cadena en la salida estándar.
El puntero pasado a la función contiene la dirección del primer caracter de la cadena.
Su función debe declararse de la siguiente manera:
void ft_putstr(char *str);

#include <unistd.h>

// Función que imprime una cadena de caracteres en la salida estándar
// Esta función toma un puntero a una cadena de caracteres (str) y escribe cada carácter de la cadena en la salida estándar (normalmente la consola).

```
void    ft_putstr(char *str)
{
   int i;

   i = 0;
   while (str[i] != '\0')    // Recorre la cadena hasta encontrar el carácter. nulo '\0'
        write(1, &str[i++], 1); // Escribe cada carácter en la salida estándar.
}

/*int main(void)
{
        ft_putstr("Hola, 42!");
        write(1, "\n", 1);
        return (0);
}*/
```

* **ft_strlen:** Función que devuelve el tamaño de un string.

name : ft_strlen;Expected files : ft_strlen.c; Allowed functions:

Escribe una función que devuelva la longitud de una cadena.
Su función debe declararse de la siguiente manera:
int ft_strlen(char *str);

#include <unistd.h>

// Función que calcula la longitud de una cadena de caracteres
// Esta función toma un puntero a una cadena de caracteres (str) y devuelve un entero que representa la cantidad de caracteres en la cadena, excluyendo el carácter nulo de terminación '\0'. Es útil para determinar el tamaño de una cadena antes de realizar operaciones como impresión o copia.

```
int ft_strlen(char *str)
{
   int i;

   i = 0;
   while (str[i] != '\0')  // Recorre la cadena hasta encontrar el carácter nulo '\0'.
      i++;                 // Incrementa el contador por cada carácter encontrado.
   return (i);             // Retorna la longitud total de la cadena.
}
```

```
/*#include <stdio.h>

// Función principal para probar `ft_strlen`
int main(void)
{
   printf("%d", ft_strlen("Hola Mundo"));  // Llama a la función e imprime el resultado
   return (0);                             // Retorna 0 indicando ejecución exitosa.
}
```

****

NIVEL 3

* **fizzbuzz:** imprime del 1 al 100, cambiando múltiplos por strings.

Assignment name : fizzbuzz; Expected files : fizzbuzz.c; Allowed functions: write.

Escribe un programa que imprima los números del 1 al 100, cada uno separado por un
nueva línea.

Si el número es múltiplo de 3, en su lugar imprime 'fizz'.
Si el número es múltiplo de 5, en su lugar imprime "buzz".
Si el número es múltiplo de 3 y múltiplo de 5, en su lugar imprime 'fizzbuzz'.

#include <unistd.h>

// Función para imprimir un número entero mediante recursión
// Esta función toma un entero y lo imprime en la salida estándar utilizando la función write de Unix, sin usar printf ni sprintf. La función maneja números mayores a 9 dividiendo el número recursivamente para extraer cada dígito y luego imprimirlos en orden.

```
void    ft_print_nbr(int i)
{
   char    *digits;

   digits = "0123456789";          // Array con los dígitos del 0 al 9.
   if (i > 9)                      // Si el número es mayor que 9, llamamos recursivamente.
      ft_print_nbr(i / 10);        // Dividimos para extraer los dígitos anteriores.
   write(1, &digits[i % 10], 1);   // Imprimimos el último dígito..
}
```

// Función principal que imprime números del 1 al 100.
// Esta función itera desde 1 hasta 100 e imprime "fizz" si el número es múltiplo de 3, "buzz" si es múltiplo de 5, "fizzbuzz" si es múltiplo de ambos, o el número mismo si no es múltiplo de ninguno. Utiliza la función ft_print_nbr para imprimir los números.

```
int main(void)
{
   int i;

   i = 1;                          // Inicializamos la variable en 1.
   while (i <= 100)                // Iteramos desde 1 hasta 100.
   {
      if ((i % 3 == 0) && (i % 5 == 0))
     // Si es múltiplo de 3 y 5, imprimimos "fizzbuzz".
         write(1, "fizzbuzz\n", 9);
      else if ((i % 3) == 0)       // Si es múltiplo de 3, imprimimos "fizz".
         write(1, "fizz\n", 5);
      else if ((i % 5) == 0)       // Si es múltiplo de 5, imprimimos "buzz".
         write(1, "buzz\n", 5);
      else                         // Si no es múltiplo de ninguno, imprimimos el número.
      {
         ft_print_nbr(i);          // Llamamos a la función para imprimir el número.
         write(1, "\n", 1);        // Nueva línea después del número.
      }
      i++;                         // Incrementamos el contador.
   }
   return (0);                     // Retorno exitoso del programa.
}
```

Example:
$>./fizzbuzz
1
2
fizz
4
buzz
fizz
7
8
fizz
buzz
11
fizz
13
14
fizzbuzz
[...]
97
98
fizz
buzz
$>

****

* **buzzfizz: i**mprime del 1 al 100, cambiando múltiplos por strings.

name : buzz; Expected files : buzzfizz.c; Allowed functions: write.

Escribe un programa que imprima los números del 1 al 100, cada uno separado por un nueva línea. Si el número es múltiplo de 4, en su lugar imprime 'buzz'. Si el número es múltiplo de 7, en su lugar imprime "fizz". Si el número es múltiplo de 4 y múltiplo de 7, en su lugar imprime 'buzzfizz'.

Example:
$>./fizzbuzz
1
2
3
buzz
5
6
fizz
buzz
9
10
11
buzz
13
fizz
15
buzz
17
18
19
buzz
fizz
22
23
buzz
25
26
27
fizzbuzz
29
30
[…]

```
97
fizz
99
buzz
$>
```

```
#include <unistd.h>

// Función que imprime un número entero utilizando recursión
void    ft_print_nbr(int i)
{
   char    *digits;

   digits = "0123456789"; // Definimos los caracteres numéricos
   if (i > 9) // Si el número tiene más de un dígito, llamamos recursivamente
      ft_print_nbr(i / 10);
   write(1, &digits[i % 10], 1); // Escribimos el último dígito
}

// Función principal que imprime números del 1 al 100
int main(void)
{
   int i;

   i = 1;
   while (i < 101) // Iteramos de 1 a 100
   {
      // Si el número es múltiplo de 28, imprimimos "fizzbuzz"
      if ((i % 28) == 0)
         write(1, "fizzbuzz", 8);
      // Si es múltiplo de 7, imprimimos "buzz"
      else if ((i % 7) == 0)
         write(1, "buzz", 4);
      // Si es múltiplo de 4, imprimimos "fizz"
      else if ((i % 4) == 0)
         write(1, "fizz", 4);
      // Si no es múltiplo de ninguno, imprimimos el número
      else
         ft_print_nbr(i);
      write(1, "\n", 1); // Nueva línea tras cada salida
      i++;
   }
   return (0);
}
```

****

NIVEL 4

* aff_first_param: Imprime el primer argumento.

name : aff_first_param;files : aff_first_param.c; functions: write
Escribe un programa que tome cadenas como argumentos y muestre su primer argumento seguido de un \n. Si el número de argumentos es menor que 1, el programa muestra \n.

Example:
$> ./aff_first_param vincent mit "l'ane" dans un pre et "s'en" vint | cat -e
vincent$
$> ./aff_first_param "j'aime le fromage de chevre" | cat -e
j'aime le fromage de chevre$
$> ./aff_first_param | cat -e
$

```
#include <unistd.h>

// Programa que imprime el primer argumento de la línea de comandos seguido de un salto de línea. Si no se proporciona ningún argumento, simplemente imprime un salto de línea.

int main(int argc, char **argv)
{
    int i;

    i = 0;
    // Verifica que haya al menos un argumento proporcionado.
    if (argc > 1)
    {
        // Recorre la cadena del primer argumento (argv[1]).
        while (argv[1][i])
        {
            // Escribe cada carácter en la salida estándar.
            write(1, &argv[1][i], 1);
            i++;
        }
    }
    // Escribe un salto de línea al final, independientemente de los argumentos.
    write(1, "\n", 1);
    return (0);
}
```

* **aff_last_param:** Imprime el último argumento.

name : aff_last_param; files : aff_last_param.c; functions: write

Escribe un programa que tome cadenas como argumentos y muestre su último

argumento seguido de una nueva línea.

Si el número de argumentos es menor que 1, el programa muestra una nueva línea.

Examples:

$> ./aff_last_param "zaz" "mange" "des" "chats" | cat -e

chats$

$> ./aff_last_param "j'aime le savon" | cat -e

j'aime le savon$

$> ./aff_last_param | cat -e

$

```
#include <unistd.h>

// Programa que imprime el ultimo argumento de la línea de comandos seguido de un salto de línea. Si no se proporciona ningún argumento, simplemente imprime un salto de línea.

int main(int argc, char **argv

{

    int i;

    i = 0;

    // Verifica que haya al menos un argumento en la línea de comandos.

    if (argc > 1)

    {

        // Recorre cada carácter del último argumento pasado al programa.

        while (argv[argc - 1][i])

        {

            // Escribe cada carácter en la salida estándar.

            write(1, &argv[argc - 1][i], 1);

            i++;

        }

    }

    // Escribe un salto de línea al final, independientemente de los argumentos.

    write(1, "\n", 1);

    return (0);

}
```

NIVEL 5

* **rotone:** imprime un string aumentando en 1 el carácter: a --> b, z --> a.

name : rotone; Expected files : rotone.c; Allowed functions: write.

Escribe un programa que tome una cadena y la muestre, reemplazando cada uno de sus
letras por la siguiente en orden alfabético.
'z' se convierte en 'a' y 'Z' se convierte en 'A'. El caso no se ve afectado.
La salida irá seguida de un \n.
Si el número de argumentos no es 1, el programa muestra \n.

Example:
$>./rotone "abc"
bcd
$>./rotone "Les stagiaires du staff ne sentent pas toujours tres bon." | cat -e
Mft tubhjbjsft ev tubgg of tfoufou qbt upvkpvst usft cpo.$
$>./rotone "AkjhZ zLKIJz , 23y " | cat -e
BlkiA aMLJKa , 23z $
$>./rotone | cat -e
$
$>
$>./rotone "" | cat -e
$
$>

#include <unistd.h>

// Función que aplica una rotación de +1 a cada letra en la cadena.

// Esta función toma una cadena de caracteres (str) y modifica cada letra avanzándola una posición en el alfabeto. Si la letra es 'Z' o 'z', se convierte en 'A' o 'a' respectivamente. Los caracteres que no son letras se dejan sin cambios. La función escribe el resultado directamente en la salida estándar.

```
void    rotone(char *str)
{
   int i;

   i = 0;
   while (str[i])                    // Recorre la cadena de entrada.
   {
      if ((str[i] >= 'A' && str[i] < 'Z')
         || (str[i] >= 'a' && str[i] < 'z'))
         str[i] += 1;                // Avanza la letra en una posición.
      else if (str[i] == 'Z')
         str[i] = 'A';               // Si es 'Z', vuelve a 'A'.
      else if (str[i] == 'z')
         str[i] = 'a';               // Si es 'z', vuelve a 'a'.
      write(1, &str[i], 1);          // Escribe el carácter modificado en la salida estándar.
      i++;
   }
}
```

// Función principal que procesa los argumentos de la línea de comandos

// Esta función verifica si se ha pasado un único argumento al programa y, si es así, llama a la función `rotone` para aplicar la rotación a ese argumento.

```
int main(int argc, char **argv)
{
   if (argc == 2)                    // Verifica que haya exactamente un argumento.
      rotone(argv[1]);               // Llama a la función para procesar la cadena.
   write(1, "\n", 1);                // Imprime una nueva línea al final.
   return (0);                       // Retorna 0 indicando ejecución exitosa.
}
```

****

* **first_word:** imprime la primera palabra del argumento, obviando espacios al principio.

Escribe un programa que tome una cadena y muestre su primera palabra, seguida de una nueva línea.

Una palabra es una sección de cadena delimitada por espacios/tabulaciones o por el inicio/final de la cuerda.

Si el número de parámetros no es 1, o si no hay palabras, simplemente muestre una nueva línea.

Examples:

```
$> ./first_word "FOR PONY" | cat -e
FOR$
$> ./first_word "this        ...       is sparta, then again, maybe    not" | cat -e
this$
$> ./first_word "    " | cat -e
$
$> ./first_word "a" "b" | cat -e
$
$> ./first_word "  lorem,ipsum  " | cat -e
lorem,ipsum$
$>
```

```
#include <unistd.h>

// Función que encuentra y escribe la primera palabra de la cadena
//      Esta función toma una cadena de caracteres (str) y escribe la primera palabra en la salida estándar. La primera palabra se define como la secuencia de caracteres que aparece antes del primer espacio, tabulación o nueva línea. Si la cadena comienza con espacios o tabulaciones, estos se ignoran al determinar el inicio de la primera palabra.
```

```
void    first_word(char *str)
{
    int i;

    i = 0;
    // Salta los espacios en blanco, tabulaciones y nuevas líneas iniciales.
    while (str[i] && (str[i] == ' ' || (str[i] >= 9 && str[i] <= 13)))
        i++;
// Escribe la primera palabra hasta encontrar un espacio o fin de cadena.
    while (str[i] && !(str[i] == ' ' || (str[i] >= 9 && str[i] <= 13)))
    {
        write(1, &str[i], 1);  // Escribe cada carácter de la palabra.
        i++;
    }
}
// Función principal que procesa los argumentos de la línea de comandos.
// Esta función verifica si se ha pasado un único argumento al programa y, si es así, llama a la función `first_word` para imprimir la primera palabra de ese argumento.
int main(int argc, char **argv)
{
    // Verifica que se haya pasado un único argumento.
    if (argc == 2)
    // Llama a la función para procesar la cadena.
        first_word(argv[1]);
    write(1, "\n", 1);         // Imprime una nueva línea al final.
    return (0);                // Retorna 0 indicando ejecución exitosa.
}
```

* **rot_13:** imprime un string variando en 13 el carácter: a --> n, z --> m.

Assignment name : rot_13; Expected files : rot_13.c; Allowed functions: write

Escribe un programa que tome una cadena y la muestre, reemplazando cada uno de sus letras por letra 13 espacios adelante en orden alfabético.
'z' se convierte en 'm' y 'Z' se convierte en 'M'. El caso no se ve afectado.
La salida irá seguida de una nueva línea.
Si el número de argumentos no es 1, el programa muestra una nueva línea.

Example:
$>./rot_13 "abc"
nop
$>./rot_13 "My horse is Amazing." | cat -e
Zl ubefr vf Nznmvat.$
$>./rot_13 "AkjhZ zLKIJz , 23y " | cat -e
NxwuM mYXVWm , 23l $
$>./rot_13 | cat -e
$
$>
$>./rot_13 "" | cat -e
$
$>

#include <unistd.h>

// Función que aplica el cifrado ROT-13 a una cadena y la imprime
// Esta función toma una cadena de caracteres (str) y aplica el cifrado ROT-13, que consiste en desplazar cada letra 13 posiciones en el alfabeto. Las letras mayúsculas y minúsculas se manejan por separado, y los caracteres que no son letras se dejan sin cambios.

```
void    rot_13(char *str)
{
   int i;

   i = 0;
   while (str[i])                    // Recorre toda la cadena de entrada.
   {
      if ((str[i] >= 'A' && str[i] <= 'Z')
         || (str[i] >= 'a' && str[i] <= 'z'))
      {                              // Comprueba si el carácter es una letra.
         if ((str[i] >= 'A' && str[i] <= 'M')
            || (str[i] >= 'a' && str[i] <= 'm'))
            str[i] += 13;            // Aplica ROT-13 sumando 13 posiciones.
         else
            str[i] -= 13;            // Aplica ROT-13 restando 13 posiciones.
      }
      write(1, &str[i], 1);          // Escribe el carácter cifrado en la salida estándar.
      i++;
   }
}
// Función principal que procesa los argumentos de la línea de comandos
//        Esta función verifica si se ha pasado un único argumento al programa y, si es así, llama a la función `rot_13` para aplicar el cifrado ROT-13 a ese argumento.

int main(int argc, char **argv)
{
   if (argc == 2)                    // Verifica que haya exactamente un argumento.
      rot_13(argv[1]);               // Llama a la función para procesar la cadena.
   write(1, "\n", 1);                // Imprime una nueva línea al final.
   return (0);                       // Retorna 0 indicando ejecución exitosa.
}
```

NIVEL 6

* **last_word:** imprime la última palabra del argumento, obviando espacios al final.

name : last_word; files : last_word.c; Allowed functions: write

Escribe un programa que tome una cadena y muestre su última palabra seguida

de \n.

Una palabra es una sección de cadena delimitada por espacios/tabulaciones o

por el inicio/final de la cadena.

Si el número de parámetros no es 1 o no hay palabras, muestre una nueva línea.

Example:

$> ./last_word "FOR PONY" | cat -e

PONY$

$> ./last_word "this ... is sparta, then again, maybe not" | cat -e

not$

$> ./last_word " " | cat -e

$

$> ./last_word "a" "b" | cat -e

$

$> ./last_word " lorem,ipsum " | cat -e

lorem,ipsum$

$>

```
#include <unistd.h>


// Función que encuentra y escribe la última palabra en la cadena
//      Esta función toma una cadena de caracteres (str) y escribe la última palabra en la salida estándar. La última palabra se define como la secuencia de caracteres que aparece después del último espacio o tabulación en la cadena. Si la cadena termina con espacios o tabulaciones, estos se ignoran al determinar el inicio de la última palabra.
void    ft_last_word(char *str)
{
   int i;
   int start;
   int end;


   i = 0;
   while (str[i])        // Avanza hasta el final de la cadena.
      i++;
   while (i > 0 && (str[i - 1] == ' ' || str[i - 1] == '\t'))
      i--;                  // Retrocede ignorando espacios finales.
   end = i;                 // Marca el final de la última palabra.
   while (i > 0 && str[i - 1] != ' ' && str[i - 1] != '\t')
      i--;                  // Retrocede hasta encontrar el inicio de la última palabra.
   start = i;               // Marca el inicio de la última palabra.
   while (start < end)  // Escribe la última palabra carácter por carácter.
   {
      write(1, &str[start], 1);
      start++;
   }
}
```

// Función principal que procesa los argumentos de la línea de comandos

// Esta función verifica si se ha pasado un único argumento al programa y, si es así, llama a la función `ft_last_word` para imprimir la última palabra de ese argumento.

```
int main(int argc, char **argv)
{
   if (argc == 2)                  // Verifica que haya exactamente un argumento.
      ft_last_word(argv[1]);       // Llama a la función para procesar la cadena.
   write(1, "\n", 1);              // Imprime una nueva línea al final.
   return (0);                     // Retorna 0 indicando ejecución exitosa.
}
```

****

* **inter:** imprime los caracteres sin duplicados del primer argumento y que también aparezcan en el segundo argumento.

Assignment name : inter; Expected files : inter.c; Allowed functions: write

Escribe un programa que tome dos cadenas y muestre, sin dobles, el
caracteres que aparecen en ambas cadenas, en el orden en que aparecen en la primera uno. La pantalla será seguida por un \n.
Si el número de argumentos no es 2, el programa muestra \n.

Examples:

$>./inter "padinton" "paqefwtdjetyiytjneytjoeyjnejeyj" | cat -e
padinto$
$>./inter ddf6vewg64f gtwthgdwthdwfteewhrtag6h4ffdhsd | cat -e
df6ewg4$
$>./inter "rien" "cette phrase ne cache rien" | cat -e
rien$
$>./inter | cat -e
$

```c
#include <unistd.h>
// Verifica si el carácter `c` ya apareció antes en `str` hasta la posición `i`
int ft_single_in_param(char *str, char c, int i)
{
   int j;

   j = 0;
   while (str[j] != '\0' && j < i)    // Recorre `str` hasta la posición `i`.
   {
      if (str[j] == c)                // Si el carácter ya apareció, retorna 1.
         return (1);
      j++;
   }
   return (0);                        // Retorna 0 si el carácter es único.
}
// Verifica si el carácter `c` está presente en la cadena `str`
int ft_in_other_param(char *str, char c)
{
   int i;

   i = 0;
   while (str[i] != '\0')             // Recorre toda la cadena `str`.
   {
      if (str[i] == c)                // Si el carácter está presente, retorna 1.
         return (1);
      i++;
   }
   return (0);                        // Retorna 0 si el carácter no está en `str`.
}
```

```
// Función principal que imprime los caracteres únicos de `argv[1]` que están en `argv[2]`
int main(int argc, char **argv)
{
   int i;

   i = 0;
   if (argc == 3)                    // Verifica que haya exactamente dos argumentos.
   {
      while (argv[1][i] != '\0')     // Recorre la primera cadena `argv[1]`.
      {
         // Si el carácter es único en `argv[1]` y está en `argv[2]`, lo imprime
         if ((ft_single_in_param(argv[1], argv[1][i], i) == 0)
            && (ft_in_other_param(argv[2], argv[1][i]) == 1))
            write(1, &argv[1][i], 1);
         i++;
      }
   }
   write(1, "\n", 1);                 // Imprime un salto de línea al final.
   return (0);                       // Retorna 0 indicando ejecución exitosa.
}
**************************************************************************
```

* **union:** mprime los caracteres sin duplicados de los dos argumentos.

Assignment name : union; Expected files : union.c; Allowed functions: write

Escribe un programa que tome dos cadenas y muestre, sin dobles, el
caracteres que aparecen en cualquiera de las cadenas.
La pantalla se mostrará en el orden en que aparecen los caracteres en la línea
de comando y irá seguido de un \n.
Si el número de argumentos no es 2, el programa muestra \n.

Example:
$>./union zpadinton "paqefwtdjetyiytjneytjoeyjnejeyj" | cat -e
zpadintoqefwjy$
$>./union ddf6vewg64f gtwthgdwthdwfteewhrtag6h4ffdhsd | cat -e
df6vewg4thras$
$>./union "rien" "cette phrase ne cache rien" | cat -e
rienct phas$
$>./union | cat -e
$
$>
$>./union "rien" | cat -e
$
$>

```
#include <unistd.h>
// Verifica si el carácter `c` ya apareció en `str` hasta la posición `limit`
// Esta función recorre la cadena `str` hasta el índice `limit` (o toda la cadena si `limit == -1`).
int ft_was_printed(char *str, char c, int limit)
{
   int i;

   i = 0;
   // Recorre `str` hasta `limit` o toda la cadena si `limit == -1.
   while (str[i] && (i < limit || limit == -1))
   {
      if (str[i] == c)              // Si el carácter ya apareció, retorna 1.
         return (1);
      i++;
   }
   return (0);                      // Retorna 0 si el carácter no ha sido impreso antes.
}
// Escribe la unión de dos cadenas sin repetir caracteres
void    ft_union(char *str1, char *str2)
{
   int i;

   i = 0;
   while (str1[i])                  // Recorre la primera cadena `str1`.
   {
      if (!ft_was_printed(str1, str1[i], i)) // Si el carácter no ha sido impreso antes, lo escribe.
         write(1, &str1[i], 1);
```

```
i++;
    }
    i = 0;
    while (str2[i])                                  // Recorre la segunda cadena `str2`.
    {
        if (!ft_was_printed(str1, str2[i], -1)       // Verifica que no esté en `str1`
            && !ft_was_printed(str2, str2[i], i))    // Verifica que no se haya repetido en `str2`
            write(1, &str2[i], 1);
        i++;
    }
}
```

// Función principal que procesa los argumentos de la línea de comandos

// Esta función verifica si se han pasado exactamente dos argumentos al programa y, si es así, llama a la función `ft_union` para imprimir la unión de los caracteres de ambos argumentos.

```
int main(int argc, char **argv)
{
    if (argc == 3)                      // Verifica que haya exactamente dos argumentos.
        ft_union(argv[1], argv[2]);     // Llama a la función para unir las cadenas.
    write(1, "\n", 1);                  // Imprime un salto de línea al final.
    return (0);                         // Retorna 0 indicando ejecución exitosa.
}
```

****

NIVEL 7

* ft_rrange: Función que devuelve un puntero a un array de números en orden descendente.

name : ft_rrange; files : ft_rrange.c; Allowed functions: malloc

Escribe la siguiente función:

int *ft_rrange(int start, int end);

Debe asignar (con malloc()) una matriz de números enteros, llenarla con números consecutivos

valores que comienzan al final y terminan al inicio (¡incluidos el inicio y el final!), luego devuelve un puntero al primer valor de la matriz.

Examples:

- With (1, 3) you will return an array containing 3, 2 and 1
- With (-1, 2) you will return an array containing 2, 1, 0 and -1.
- With (0, 0) you will return an array containing 0.
- With (0, -3) you will return an array containing -3, -2, -1 and 0.

```
#include <stdlib.h>
// Función que genera un array de números en orden inverso desde `start` hasta `end`
// Esta función toma dos enteros, `start` y `end`, y devuelve un puntero a un array
// de enteros que contiene todos los números en el rango [end, start] en orden inverso.
int *ft_rrange(int start, int end)
{
    int *range;
    int len;
    int i;
```

```
    i = 0;
    if (start > end)                    // Calcula la cantidad de elementos.
        len = start - end + 1;
    else
        len = end - start + 1;

    range = (int *)malloc(sizeof(int) * len); // Reserva memoria para el array.
    if (!range)                         // Verifica si la asignación de memoria falló.
        return (NULL);                  // Retorna NULL en caso de error.

    while (i < len)
    {
        range[i++] = end;               // Asigna el valor de `end` en el array.
        if (end > start)                // Si `end` es mayor que `start`, decrementa.
            end--;
        else                            // Si `end` es menor que `start`, incrementa.
            end++;
    }
    return (range);                     // Retorna el array generado.
}
```

```
*#include <stdio.h>
// Función principal para probar `ft_rrange`
int main(void)

{
    int *numbers;
    int i = 0;
    int start = 0, end = -3;
    int len = (start > end) ? (start - end + 1) : (end - start + 1);

    numbers = ft_rrange(start, end);  // Llamada a la función `ft_rrange`
    while (i < len)
        printf("%d, ", numbers[i++]);  // Imprime cada número del array

    free(numbers);  // Libera la memoria asignada
    return (0);
}
```

****

* ft_itoa: Función que convierte un int en un carácter ascii

name : ft_itoa; Expected files : ft_itoa.c; Allowed functions: malloc

Escribe una función que tome un int y lo convierta en una cadena terminada en nulo.
La función devuelve el resultado en una matriz de caracteres que debes asignar.
Su función debe declararse de la siguiente manera:
char *ft_itoa(int nbr);

```
#include <stdlib.h>

// Función que calcula la longitud de un número en base 10
// Esta función toma un número entero y devuelve la cantidad de dígitos que tiene.
int ft_nbrlen(int nbr)
{
   int len;

   if (nbr <= 0)              // Si el número es negativo o 0, cuenta al menos 1 dígito.
      len = 1;
   else
      len = 0;
   while (nbr)                // Divide el número por 10 hasta que sea 0.
   {
      len++;                  // Incrementa la longitud por cada división.
      nbr /= 10;
   }
   return (len);              // Retorna la cantidad de dígitos.
}
```

```
// Copia hasta `n` caracteres de `src` en `dest` y añade terminación `\0`
char    *ft_strncpy(char *dest, const char *src, size_t n)
{
   size_t  i;

   i = 0;
   while (i < n && src[i] != '\0')   // Copia los caracteres de `src` a `dest`.
   {
      dest[i] = src[i];
      i++;
   }
   dest[i] = '\0';                   // Asegura que la cadena termine en `\0`.
   return (dest);                    // Retorna el puntero a `dest`.
}
```

```
// Convierte un número entero en una cadena de caracteres (`itoa` - integer to ASCII)
char    *ft_itoa(int nbr)
{
   int     len;
   char    *str;

   len = ft_nbrlen(nbr);                  // Calcula la cantidad de dígitos del número.
   str = (char *)malloc(sizeof(char) * (len + 1));  // Reserva memoria para la cadena.
   if (!str)                              // Verifica si `malloc` ha fallado.
      return (NULL);
   str[len] = '\0';                       // Asegura que la cadena termine en `\0`.
   if (nbr == -2147483648)                // Caso especial para el valor mínimo de `int`.
      return (ft_strncpy(str, "-2147483648", 11));
   if (nbr < 0)                           // Si el número es negativo, lo convierte a positivo.
   {
      str[0] = '-';                       // Añade el signo negativo.
      nbr = -nbr;
   }
   while (len-- && nbr)                   // Llena la cadena con los dígitos del número.
   {
      str[len] = (nbr % 10) + '0';        // Convierte el dígito numérico a carácter.
      nbr /= 10;
   }
   return (str);                          // Retorna la cadena resultante.
}
```

```
/*#include <stdio.h>

int	main(void)
{
	printf("%s\n", ft_itoa(0));
	printf("%s\n", ft_itoa(-2147483648));
	printf("%s\n", ft_itoa(1));
	printf("%s\n", ft_itoa(12));
	printf("%s\n", ft_itoa(2147483647));
	printf("%s\n", ft_itoa(-4));
	printf("%s\n", ft_itoa(-23));
	return (0);
}*/
```

Truco para obtener el número max INT

un "int" esta formado por 32 bits

Positivos y negativos (enteros con signo): del -2147483648 al 2147483647

En un terminal ejecuta "bc" (Basic Calculator)

pones 2 ^ 31 y obtienes 2147483648

para salir "quit"

*/

****

* **ft_range:** Función que devuelve un puntero a un array de números en orden ascendente.

name : ft_range; Expected files : ft_range.c; Allowed functions: malloc

Escribe la siguiente función: int *ft_range(int start, int end);

Debe asignar (con malloc()) una matriz de números enteros, llenarla con números consecutivos

valores que comienzan al principio y terminan al final (¡incluidos el inicio y el final!), luego

devuelve un puntero al primer valor de la matriz.

Examples:

- With (1, 3) you will return an array containing 1, 2 and 3.
- With (-1, 2) you will return an array containing -1, 0, 1 and 2.
- With (0, 0) you will return an array containing 0.
- With (0, -3) you will return an array containing 0, -1, -2 and -3.

```
#include <stdlib.h>
// Función que genera un array de números desde `start` hasta `end`
// Esta función toma dos enteros, `start` y `end`, y devuelve un puntero a un array de enteros que contiene todos los números en el rango [start, end].
int *ft_range(int start, int end)
{
    int *range;
    int len;
    int i;

    i = 0;
```

```
    if (start > end)          // Calcula el tamaño del array basado en el rango.
        len = start - end + 1;
    else
        len = end - start + 1;
    // Reserva memoria para el array.
    range = (int *)malloc(sizeof(int) * len);
    if (!range)               // Verifica si la asignación de memoria falló.
        return (NULL);        // Retorna NULL en caso de error.

    while (i < len)
    {
        range[i++] = start;   // Asigna el valor de `start` en el array.
        if (start > end)      // Si `start` es mayor que `end`, decrementa.
            start--;
        else                  // Si `start` es menor que `end`, incrementa.
            start++;
    }
    return (range);           // Retorna el array generado.
}
```

```
/*#include <stdio.h>

// Función principal para probar `ft_range`
int main(void)
{
    int *numbers;
    int i = 0;
    int start = 0, end = -3;
    int len = (start > end) ? (start - end + 1) : (end - start + 1);

    numbers = ft_range(start, end);  // Llamada a la función `ft_range`
    while (i < len)
        printf("%d, ", numbers[i++]);  // Imprime cada número del array

    free(numbers);  // Libera la memoria asignada
    return (0);
}
```

****

NIVEL 8

* **expand_str:** Imprime un argumento separando con 3 espacios las palabras, obviando espacios al principio y al final.

Escribe un programa que tome una cadena y la muestre exactamente con tres espacios.
entre cada palabra, sin espacios ni tabulaciones ni al principio ni al final,
seguido de una nueva línea.
Una palabra es una sección de cadena delimitada por espacios/tabulaciones o por el inicio/final de la cadena.
Si el número de parámetros no es 1, o si no hay palabras, simplemente muestre
una nueva línea.

Examples:
$> ./expand_str "vous voyez c'est facile d'afficher la meme chose" | cat -e
vous voyez c'est facile d'afficher la meme chose$
$> ./expand_str " seulement la c'est plus dur " | cat -e
seulement la c'est plus dur$
$> ./expand_str "comme c'est cocasse" "vous avez entendu, Mathilde ?" | cat -e
$
$> ./expand_str "" | cat -e
$
$>

```
#include <unistd.h>

// Verifica si un carácter es un espacio, tabulación o nueva línea.
int is_space(char c)
{
    return (c == ' ' || (c >= 9 && c <= 13));
}
```

// Expande los espacios entre palabras a exactamente tres espacios

// Esta función toma una cadena de caracteres y reemplaza cada secuencia de espacios por exactamente tres espacios, asegurando que las palabras estén separadas adecuadamente.

```
void    expand_str(char *str)
{
   int i;

   i = 0;
   while (str[i] && is_space(str[i])) // Ignora los espacios iniciales.
      i++;
   while (str[i])
   {
      if (is_space(str[i]))    // Si encuentra espacios, los reemplaza por "   ".
      {
         while (str[i] && is_space(str[i])) // Salta todos los espacios consecutivos.
            i++;
         if (str[i])              // Si aún hay texto, inserta los tres espacios.
            write(1, "   ", 3);
      }
      if (str[i])                 // Imprime cada carácter de la palabra.
      {
         write(1, &str[i], 1);
         i++;
      }
   }
}
```

// Función principal que procesa los argumentos de la línea de comandos

// Esta función verifica si se ha pasado un único argumento al programa y, si es así, llama a la función `expand_str` para expandir los espacios en esa cadena.

```
int main(int argc, char **argv)
{
    if (argc == 2)                  // Verifica que haya exactamente un argumento.
        expand_str(argv[1]);        // Llama a la función de expansión.
    write(1, "\n", 1);              // Imprime un salto de línea al final.
    return (0);                     // Retorna 0 para indicar ejecución exitosa.
}
}          **********************************************************************
```

* **ft_split:** Función que divide en palabras un argumento y devuelve un array con los "strings".

name : ft_split; Expected files   : ft_split.c; Allowed functions: malloc

Escribe una función que tome una cadena, la divida en palabras y las devuelva como

una matriz de cadenas terminada en NULL.

Una "palabra" se define como parte de una cadena delimitada por espacios/tabulaciones/nueva

líneas, o por el inicio/final de la cadena.

Su función debe declararse de la siguiente manera:

char **ft_split(char *str);

```
#include <stdlib.h>


// Verifica si un carácter es un espacio en blanco, tabulación o nueva línea.
int is_space(char c)
{
   return (c == ' ' || (c >= 9 && c <= 13));      // Retorna 1 si el carácter es un espacio.
}
// Cuenta cuántas palabras hay en la cadena
// Esta función recorre la cadena y cuenta las palabras, ignorando espacios al inicio y al final, así como los espacios entre palabras.



int ft_count_words(char *str)
{
   int i;
   int count;

   i = 0;
   count = 0;
   while (str[i])                          // Recorre toda la cadena.
   {
      while (str[i] && is_space(str[i]))   // Ignora espacios iniciales.
         i++;
      if (str[i])                          // Si hay un carácter válido, incrementa el contador.
         count++;
      while (str[i] && !is_space(str[i]))  // Avanza hasta el final de la palabra.
         i++;
   }
   return (count);                         // Retorna el número total de palabras.
}
```

```
// Copia una palabra desde la posición 'start' hasta 'end' en una nueva cadena.
char    *ft_word_dup(char *str, int start, int end)
{
   char    *word;
   int     i;

   word = (char *)malloc(sizeof(char) * (end - start + 1));  // Reserva memoria para la palabra.
   i = 0;
   if (!word)                                                // Verifica si 'malloc' falló.
      return (NULL);
   while (start < end)                                       // Copia cada carácter de la palabra.
      word[i++] = str[start++];
   word[i] = '\0';                                           // Agrega el terminador de cadena.
   return (word);                                            // Retorna la nueva palabra duplicada.
}
```

```
// Divide la cadena en palabras y devuelve un array de cadenas
char    **ft_split(char *str)
{
   char    **split;
   int     i;
   int     j;
   int     start;

   i = 0;
   j = 0;
   // Reserva memoria para el array.
   split = (char **)malloc(sizeof(char *) * (ft_count_words(str) + 1));
   if (!split)                                  // Verifica si 'malloc' falló.
      return (NULL);
   while (str[i])                               // Recorre la cadena.
   {
      while (str[i] && is_space(str[i]))        // Ignora espacios iniciales.
         i++;
      start = i;                                // Marca el inicio de la palabra.
      while (str[i] && !is_space(str[i]))       // Avanza hasta el final de la palabra.
         i++;
      if (start != i)                           // Si hay una palabra válida, la guarda en el array.
         split[j++] = ft_word_dup(str, start, i);
   }
   split[j] = NULL;                             // Termina el array con NULL.
   return (split);                              // Retorna el array de palabras.
}
```

```
/*
int main(void)
{
   char **split;

   split = ft_split("Cada palabra una linea");
        printf("Cada    --> %s\n", split[0]);
   printf("palabra --> %s\n", split[1]);
   printf("una     --> %s\n", split[2]);
   printf("linea   --> %s\n", split[3]);
   return (0);
}*/
```

****