Ejercicios C

Document by Jorge Achiaga
Versión: 1.0
Fecha ini: 27 jun 2023
Ir a índice Tec: ir a doc

Indice

Contenido

Referencia:

https://github.com/ikerkeb ??? no se encuentra
otras referencias: ir a github,

00.-Level

01.- aff_a

Expected Files: aff_a.c
Allowed functions: write

Write a program that takes a string, and displays the first 'a' character it encounters in it, followed by a newline. If there are no 'a' characters in the string, the program just writes a newline. If the number of parameters is not 1, the program displays 'a' followed by a newline.

Example:

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

```c
#include <unistd.h>
int       main(int argc, char **argv)
{
    int i;

    i = 0;
    if (argc != 2)
    {
       write(1, "a\n", 2);
       return (0);
    }
    while (argv[1][i])
    {
       if (argv[1][i] == 'a')
       {
          write(1, "a", 1);
          break;
       }
       i++;
    }
    write(1, "\n", 1);
    return (0);
}
```

02.- ft_countdown

Expected Files: ft_countdown.c

Allowed functions: write

Write a program that displays all digits in descending order, followed by a newline.

Example:

```
$> ./ft_countdown | cat -e
9876543210$
$>
```

```c
#include <unistd.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}

int     main(void)
{
    int i;

    i = 9;
    while (i > -1)
    {
        ft_putchar(i + '0');
        i--;
    }
    write(1, "\n", 1);
    return (0);
}
```

```c
 #include <unistd.h>
 int    main(void)
 {
        write(1, "9876543210\n", 11);
        return (0);
 }
```

03.- ft_print_numbers

Expected Files: ft_print_numbers.c
Allowed functions: write

Write a function that displays all digits in ascending order. Your function must be declared as follows:
void   ft_print_numbers(void);

```c
#include <unistd.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}

void    ft_print_numbers(void)
{
    int i;

    i = 0;
    while (i < 10)
    {
        ft_putchar(i + '0');
        i++;
    }
}
#include <unistd.h>
void   ft_print_numbers(void)
{
write(1, "0123456789", 10)
}
```

04.- hello

Expected Files: hello.c
Allowed functions: write

Write a program that displays "Hello World!" followed by a \n.
Example:
$>./hello
Hello World!
$>./hello | cat -e
Hello World!$

```c
#include <unistd.h>
int     main(void)
{
    write(1, "Hello World!\n", 13);
    return (0);
}
```

05.- maff_alpha

Expected Files: maff_alpha.c
Allowed functions: write

Write a program that displays the alphabet, with even letters in uppercase, and odd letters in lowercase, followed by a newline.

Example:

```
$> ./maff_alpha | cat -e
aBcDeFgHiJkLmNoPqRsTuVwXyZ$
```

```c
#include <unistd.h>
int      main(void)
{
    write(1, "aBcDeFgHiJkLmNoPqRsTuVwXyZ\n", 27);
      return (0);
}
```

06.- aff_first_param

Expected Files: aff_first_param.c
Allowed functions: write

Write a program that takes strings as arguments, and displays its first argument followed by a \n. If the number of arguments is less than 1, the program displays \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
$
```

```c
#include <unistd.h>
void    ft_putchar(char c)
{
        write(1, &c, 1);
}
int             main(int argc, char **argv)
{
        int i;
        i = 0;
        if (argc < 2)
        {
                write(1, "\n", 1);
                return (0);
        }
        while(argv[1][i])
        {
                ft_putchar(argv[1][i]);
                i++;
        }
        write(1, "\n", 1);
        return (0);
}
```

07.- aff_last_param

Expected Files: aff_last_param.c
Allowed functions: write

Write a program that takes strings as arguments, and displays its last argument followed by a newline. If the number of arguments is less than 1, the program displays a newline.

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
$
```

```c
#include <unistd.h>

void    ft_putchar(char c)
{
    write(1, &c, 1);
}

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

    i = 0;
    if (argc < 2)
    {
        write(1, "\n", 1);
        return (0);
    }
    while (argv[argc - 1][i])
    {
        ft_putchar(argv[argc - 1][i]);
        i++;
    }
    write(1, "\n", 1);
    return (0);
}
```

08.- maff_revalpha

Expected Files: maff_revalpha.c
Allowed functions: write

Write a program that displays the alphabet in reverse, with even letters in uppercase, and odd letters in lowercase, followed by a newline.

Example:

```
$> ./maff_revalpha | cat -e
zYxWvUtSrQpOnMlKjIhGfEdCbA$
```

```c
#include <unistd.h>

int     main(void)
{
    write(1, "zYxWvUtSrQpOnMlKjIhGfEdCbA\n", 27);
    return (0);
}
```

```c
#include <unistd.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}
int     main(void)
{
    int i;
    i = 123;
    while (i-- > 97)
        (i % 2 == 0) ? ft_putchar(i) : ft_putchar(i - 32);
    ft_putchar('\n');;
    return (0);
}
```

09.- only_a

Expected Files: only_a.c
Allowed functions: write

Write a program that displays a 'a' character on the standard output.

```c
#include <unistd.h>
int     main(void)
{
    write(1, "a", 1);
    return(0);
}
```

10.- only_z

Expected Files: only_z.c
Allowed functions: write

Write a program that displays a 'z' character on the standard output.

```c
#include <unistd.h>

int     main(void)
{
    write(1, "z", 1);
    return(0);
}
```

11.- aff_z

Expected Files: aff_z.c
Allowed functions: write

Write a program that takes a string, and displays the first 'z' character it encounters in it, followed by a newline. If there are no 'z' characters in the string, the program writes 'z' followed by a newline. If the number of parameters is not 1, the program displays 'z' followed by a newline.

Example:

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

```c
#include <unistd.h>
int     main(void)
{
        write(1, "z\n", 2);
        return (0);
}
```

12.- ft_strcpy

Expected Files: ft_strcpy.c
Allowed functions:

Reproduce the behavior of the function strcpy (man strcpy). Your function must be declared as follows:

char   *ft_strcpy(char *s1, char *s2);

```c
#include "libft.h"
char *ft_strcpy(char *dest, char *src)
{
    int i;
    int j;

    j = 0;
    i = 0;
    while (src[i])
    {
        dest[j] = src[i];
        i++;
        j++;
    }
    dest[j] = '\0';
    return (dest);
}
```

13.- ft_strlen

Expected Files: ft_strlen.c
Allowed functions:

Write a function that returns the length of a string. Your function must be declared as follows:
int	ft_strlen(char *str);

```
int ft_strlen(char *str)
{
	int i:
	i = 0:
	while (str[i])
	{
	i++;
	}
	return (i);
}
```

14.- repeat_alpha

Expected Files: repeat_alpha.c
Allowed functions: write

Write a program called repeat_alpha that takes a string and display it repeating each alphabetical character as many times as its alphabetical index, followed by a newline.

'a' becomes 'a', 'b' becomes 'bb', 'e' becomes 'eeeee', etc...

Case remains unchanged. If the number of arguments is not 1, just display a newline.

Examples:

```
$>./repeat_alpha "abc"
abbccc
$>./repeat_alpha "Alex." | cat -e
Allllllllllllleeeeexxxxxxxxxxxxxxxxxxxxxxxx.$
$>./repeat_alpha 'abacadaba 42!' | cat -e
abbacccaddddabba 42!$
$>./repeat_alpha | cat -e
$
$>
$>./repeat_alpha "" | cat -e
$
$>
```

```c
#include <unistd.h>

void    ft_putchar(char c)
{
    write(1, &c, 1);
}
int     icheck(char c)
{
    int i;
    if (c >= 'a' && c <= 'z')
        i = c - 'a' + 1;
    else if (c >= 'A' && c <= 'Z')
        i = c - 'A' + 1;
    else
        i = 1;
    return (i);
}
int     main(int argc, char **argv)
{
    int index;
    int i;
    i = 0;
    if (argc != 2)
    {
        write(1, "\n", 1);
        return (0);
    }
    while (argv[1][i])
    {
        index = icheck(argv[1][i]);
        while (index > 0)
        {
                ft_putchar(argv[1][i]);
                index--;
        }
        i++;
    }
    write(1, "\n", 1);
    return (0);
}
```

01.-Level

15.- search_and_replace

Expected Files: search_and_replace.c
Allowed functions: write, exit

Write a program called search_and_replace that takes 3 arguments, the first arguments is a string in which to replace a letter (2nd argument) by another one (3rd argument). If the number of arguments is not 3, just display a newline. If the second argument is not contained in the first one (the string) then the program simply rewrites the string followed by a newline.

Examples:

```
$>./search_and_replace "Papache est un sabre" "a" "o"
Popoche est un sobre
$>./search_and_replace "zaz" "art" "zul" | cat -e
$
$>./search_and_replace "zaz" "r" "u" | cat -e
zaz$
$>./search_and_replace "jacob" "a" "b" "c" "e" | cat -e
$
$>./search_and_replace "ZoZ eT Dovid oiME le METol." "o" "a" | cat -e
ZaZ eT David aiME le METal.$
$>./search_and_replace "wNcOre Un ExEmPle Pas Facilw a Ecrirw " "w" "e" | cat -e
eNcOre Un ExEmPle Pas Facile a Ecrire $
```

```c
#include <unistd.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}
int      main (int argc, char **argv)
{
    int i;
    argc = 4;
    if (argc == 4)
    {
        i = 0;
        while (argv[1][i])
        {
               if (argv[1][i] == argv[2][0])
                      argv[1][i] = argv[3][0];
               ft_putchar(argv[1][i]);
               i++;
        }
        ft_putchar('\0');
        return (0);
    }
}
```

16.- ulstr

Expected Files: ulstr.c
Allowed functions: write

Write a **program** that takes a string and reverses the case of all its letters. Other characters remain unchanged. You must display the result followed by a '\n'. If the number of arguments is not 1, the program displays '\n'.

Examples :

```
$>./ulstr "L'eSPrit nE peUt plUs pRogResSer s'Il staGne et sI peRsIsTent VAnIte et auto-justification." | cat -e
l'EspRIT Ne PEuT PLuS PrOGrESsER S'iL STAgNE ET Si PErSiStENT vaNiTE ET AUTO-JUSTIFICATION.$
$>./ulstr "S'enTOuRer dE sECreT eSt uN sIGnE De mAnQuE De coNNaiSSanCe.  " | cat -e
s'ENtoUrER De SecREt EsT Un SigNe dE MaNqUe dE COnnAIssANcE.  $
$>./ulstr "3:21 Ba  tOut  moUn ki Ka di KE m'en Ka fe fot" | cat -e
3:21 bA  ToUT  MOuN KI kA DI ke M'EN kA FE FOT$
$>./ulstr | cat -e
$
```

```c
#include <unistd.h>
int     xxx(char c)
{
        char index;
        if (c >= 'A' && c <= 'Z')
        {
          index = c + 32;
          return (index);
        }
        if (c >= 'a' && c <= 'z')
        {
          index = c -32;
          return (index);
        }
        else
          index = c;
        return (index);
 }
 int    main(int argc, char **argv)
 {
        char index;
        int i;

        i = 0;
        if (argc != 2)
        {
        write(1, "\n", 1);
        }
        while (argv[1][i])
        {
        index = xxx(argv[1][i]);
        write(1, &index, 1);
        i++;
        }
        write(1, "\n", 1);
        return (0);
}
```

17.- rot_13

Expected Files: rot_13.c
Allowed functions: write

Write a program that takes a string and displays it, replacing each of its letters by the letter 13 spaces ahead in alphabetical order. 'z' becomes 'm' and 'Z' becomes 'M'. Case remains unaffected. The output will be followed by a newline. If the number of arguments is not 1, the program displays a newline.

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
$
$>
```

```c
#include <unistd.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}
int     main(int argc, char **argv)
{
    int i;
    i = 0;
    if (argc != 2)
    {
        write(1, "\n", 1);
        return (0);
    }
    while (argv[1][i])
    {
        if (argv[1][i] >= 'a' && argv[1][i] <= 'm')
                argv[1][i] += 13;
        else if (argv[1][i] >= 'n' && argv[1][i] <= 'z')
                argv[1][i] = (argv[1][i] - 'm') + 'a' - 1;

        else if (argv[1][i] >= 'A' && argv[1][i] <= 'M')
                argv[1][i] += 13;

        else if (argv[1][i] >= 'N' && argv[1][i] <= 'Z')
               argv[1][i] = (argv[1][i] - 'M') + 'A' - 1;
        ft_putchar(argv[1][i]);
        i++;
    }
    write(1, "\n", 1);
    return (0);
}
```

18.- first_word

Expected Files: first_word.c
Allowed functions: write

Write a program that takes a string and displays its first word, followed by a newline. A word is a section of string delimited by spaces/tabs or by the start/end of the string. If the number of parameters is not 1, or if there are no words, simply display a newline.

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

```c
#include <unistd.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}
int      main(int argc, char **argv)
{
    int i;
    i = 0;
    if (argc != 2)
    {
        write(1, "\n", 1);
        return (0);
    }
    while (argv[1][i])
    {
        while ((argv[1][i] == ' ' || argv[1][i] == '\t') && (argv[1][i]))
                i++;
        while ((argv[1][i] != ' ' && argv[1][i] != '\t') && (argv[1][i]))
        {
                ft_putchar(argv[1][i]);
                i++;
        }
        if (argv[1][i] == ' ' || argv[1][i] == '\t')
                break;
    }
    write(1, "\n", 1);
    return (0);
}
```

19.- ft_putstr

Expected Files: ft_putstr.c
Allowed functions: write

Write a function that displays a string on the standard output. The pointer passed to the function contains the address of the string's first character. Your function must be declared as follows:
void   ft_putstr(char *str);

```c
#include <unistd.h>
void    ft_putstr(char *str);
{
        while (*str)
        {
        write (1, str, 1);
        str++
        }
}
```

20.- ft_swap

Expected Files: ft_swap.c
Allowed functions:

Write a function that swaps the contents of two integers the adresses of which are passed as parameters. Your function must be declared as follows:
void   ft_swap(int *a, int *b);

```c
#include <unistd.h>
void    ft_swap(int *a, int *b);
{
        int temp;
        temp = *a;
        *a = *b;
        *b = temp;
}
```

21.- first_word

Expected Files: first_word.c
Allowed functions: write

Write a program that takes a string and displays its first word, followed by a newline. A word is a section of string delimited by spaces/tabs or by the start/end of the string. If the number of parameters is not 1, or if there are no words, simply display a newline.

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

```c
#include <unistd.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}
int     main(int ac, char **av)
{
    int i;
    i = 0;
    if (ac == 2)
    {
        while (av[1][i] && (av[1][i] == ' ' || av[1][i] == '\t'))
                i++;
        while (av[1][i] && (av[1][i] != ' ' && av[1][i] != '\t'))
        {
                ft_putchar(av[1][i]);
                i++;
        }
    }
    ft_putchar('\n');
    return (0);
}
```

22.- rev_print

Expected Files: rev_print.c
Allowed functions: write

Write a program that takes a string, and displays the string in reverse followed by a newline. If the number of parameters is not 1, the program displays a newline.

Examples:

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

```c
int main (int argc, char **str)
{
	int i;
	int j;

	if (argc != 2)
	write(1, "\n", 1);
	i = 0;
	while (str[1][i] != '\0')
	{
	i++;
	}
	i--;
	while (i >= 0)
	{
	write(1, &str[1][i], 1);
	i--;
	}
	write(1, "\n", 1);
	return (0);
}
```

23.- rotone

Expected Files: rotone.c
Allowed functions: write

Write a program that takes a string and displays it, replacing each of its letters by the next one in alphabetical order. 'z' becomes 'a' and 'Z' becomes 'A'. Case remains unaffected. The output will be followed by a \n. If the number of arguments is not 1, the program displays \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
$
$>
```

```c
#include <unistd.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}

int     main(int argc, char **argv)
{
    int i;
    i = 0;
    if (argc != 2)
    {
        write(1, "\n", 1);
        return (0);
    }
    while (argv[1][i])
    {
        if (argv[1][i] == 'Z')
                argv[1][i] = 'A';
        else if (argv[1][i] == 'z')
                argv[1][i] = 'a';
        else if (argv[1][i] >= 'A' && argv[1][i] <= 'z')
                argv[1][i] += 1;
        ft_putchar(argv[1][i]);
        i++;
    }
    write(1, "\n", 1);
    return (0);
}
```

24.- ft_atoi

Expected Files: ft_atoi.c
Allowed functions: None

Write a function that converts the string argument str to an integer (type int) and returns it. It works much like the standard atoi(const char *str) function, see the man. Your function must be declared as follows:

int   ft_atoi(const char *str);

```
#include <unistd.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}
int     main(int argc, char **argv)
{
    int i;

    i = 0;
    if (argc != 2)
    {
        write(1, "\n", 1);
        return (0);
    }
    while (argv[1][i])
    {
        if (argv[1][i] == 'Z')
                argv[1][i] = 'A';
        else if (argv[1][i] == 'z')
                argv[1][i] = 'a';
        else if (argv[1][i] >= 'A' && argv[1][i] <= 'z')
                argv[1][i] += 1;
        ft_putchar(argv[1][i]);
        i++;
    }
    write(1, "\n", 1);
    return (0);
}
```

25.- ft_strdup

Expected Files: ft_strdup.c
Allowed functions: malloc

Reproduce the behavior of the function strdup (man strdup). Your function must be declared as follows:
char *ft_strdup(char *src);

```c
#include <stdlib.h>
char    *ft_strcpy(char *dest, char *src)
{
    int i;
    int j;
    j = 0;
    i = 0;
    while (src[i])
    {
        dest[j] = src[i];
        i++;
        j++;
    }
    dest[j] = '\0';
    return (dest);
}

int     ft_strlen(char *str)
{
    int i;
    i = 0;
    while (*str[i])
    {
        i++;
    }
    return (i);
}

char    *ft_strdup(char *src)
{
    char *savestr;

    savestr = (char*)malloc(sizeof(*savestr) * (ft_strlen(src) + 1));
    savestr = ft_strcpy(savestr, src);
    return (savestr);
}
```

26.- inter

Expected Files: inter.c
Allowed functions: write

Write a program that takes two strings and displays, without doubles, the characters that appear in both strings, in the order they appear in the first one. The display will be followed by a \n. If the number of arguments is not 2, the program displays \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>
int     comp(char *str, char c, int index)
{
    int i;
    i = 0;
    while (i < index)
    {
        if (str[i] == c)
                return (1);
        i++;
    }
    return (0);
}

void    inter(char *s1, char *s2)
{
    int i;
    int j;
    i = 0;
    while (s1[i])
    {
        if (comp(s1, s1[i], i) == 0)
        {
                j = 0;
                while (s2[j])
                {
                        if (s2[j] == s1[i])
                        {
                                write(1, &s1[i], 1);
                                break;
                        }
                        j++;
                }
        }
        i++;
    }
}
int     main(int argc, char **argv)
{
    if (argc == 3)
        inter(argv[1], argv[2]);
    write(1, "\n", 1);
    return (0);
}
```

27.- last_word

Expected Files: last_word.c
Allowed functions: write

Write a program that takes a string and displays its last word followed by a \n. A word is a section of string delimited by spaces/tabs or by the start/end of the string. If the number of parameters is not 1, or there are no words, display a newline.

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>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}
void ft_putstr(char *str)
{
    int i;
    i = 0;
    while (str[i] != '\0')
    {
        if (str[i] >= 33 && str[i] <= 126)
                ft_putchar(str[i]);
        i++;
    }
}
void    display_word(char str *)
{
    char *last;
    int i;
    i = 0;
    last = &str[i];
    while (str[i] != '\0')
    {
        if (!(str[i] >= 33 && str[i] <= 126))
        {
                if (str[i + 1] >= 33 && str[i + 1] <= 126)
                        last = &str[i + 1];
        }
        i++;
    }
    if (last)
        ft_putstr(last);
}
int main (int argc, char **argv)
{
    if (argc == 2)
        display_word(argv[1]);
    ft_puthcar('\n')
    return (0);
}
```

28.- reverse_bits

Expected Files: reverse_bits.c
Allowed functions:

Write a function that takes a byte, reverses it, bit by bit (like the **Example**) and returns the result. Your function must be declared as follows:
unsigned char	reverse_bits(unsigned char octet);
Example:

```
 1 byte
_______________
0010  0110
  ||
  \/
0110  0100
```

```
#include <unistd.h>

unsigned char reverse_bits(unsigned char b)

	b = (b & 0xF0) >> 4 | (b & 0x0F) << 4;
	b = (b & 0xCC) >> 2 | (b & 0x33) << 2;
	b = (b & 0xAA) >> 1 | (b & 0x55) << 1;
	return b;
}
```

29.- swap_bits

Expected Files: swap_bits.c
Allowed functions:

Write a function that takes a byte, swaps its halves (like the **Example**) and returns the result. Your function must be declared as follows:
unsigned char   swap_bits(unsigned char octet);

Example:

```
 1 byte
_____________
 0100 | 0001
        \ /
        / \
 0001 | 0100
```

```
#include <unistd.h>
void    print_bits(unsigned char octet)
{
    int i;
    char c;
    i = 128;
    while (i > 0)
    {
        if (i > octet)
        {
                c = '0';
                i = i / 2;
                write(1, &c, 1);
        }
        else
        {
                c = '1';
                write(1, &c, 1);
                octet = octet - i;
                i = i / 2;
        }
    }
}
unsigned char swap_bits(unsigned char octet)
{
    octet = (octet >> 4) | (octet << 4);
    print_bits(octet);
    return (0);
}
int main(void)
{
    unsigned char i;
    i = 'a';
    write(1, "N:", 2);
    print_bits(i);
    write(1, "\nS:", 3);
    swap_bits(i);
    return (0);
}unsigned char    swap_bits(unsigned char octet)
{
    return ((octet >> 4) | (octet << 4));
}
int       main(void)
{
    char c;
    c = 't';
    write(1, &c, 1);
    c = swap_bits(c);
    write(1, &c, 1);
    return (0);
}
```

30.- union

Expected Files: union.c
Allowed functions: write

Write a program that takes two strings and displays, without doubles, the characters that appear in either one of the strings. The display will be in the order characters appear in the command line, and will be followed by a \n. If the number of arguments is not 2, the program displays \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
$
$>
```

```c
#include <unistd.h>
int     ft_str(char *str, char c, int i)
{
    int x;
    x = 0;
    while (i > x)
    {
        if (str[x] == c)
                return (1);
        x++;
    }
    return (0);
}

int     main(int argc, char **argv)
{
    int i;
    int j;
    i = 0;
    j = 0;
    if (argc == 3)
    {
        while (argv[1][i])
        {
                if (ft_str(argv[1], argv[1][i], i) == 0)
                        write(1, &argv[1][i], 1);
                i++;
        }
        while (argv[2][j])
        {
                if((ft_str(argv[1], argv[2][j], i) == 0)
                              && (ft_str(argv[2], argv[2][j], j) == 0))
                        write(1, &argv[2][j], 1);
                j++;
        }
    }
    write(1, "\n", 1);
    return (0);
}
```

31.- alpha_mirror

Expected Files: alpha_mirror.c
Allowed functions: write

Write a program called alpha_mirror that takes a string and displays this string after replacing each alphabetical character by the opposite alphabetical character, followed by a newline. 'a' becomes 'z', 'Z' becomes 'A', 'd' becomes 'w', 'M' becomes 'N' and so on. Case is not changed. If the number of arguments is not 1, display only a newline.

Examples:

```
$>./alpha_mirror "abc"
zyx
$>./alpha_mirror "My horse is Amazing." | cat -e
Nb slihv rh Znzarmt.$
$>./alpha_mirror | cat -e
$
$>
```

```c
#include <unistd.h>
int     main(int argc, char **argv)
{
    int i;
    i = 0;
    if (argc == 2)
    {
        while(argv[1][i] != '\0')
        {
               if (argv[1][i] > 64 && argv[1][i] < 91)
               {
                      argv[1][i] = 155 - argv[1][i];
                      write(1, &argv[1][i], 1);
               }
               else if (argv[1][i] > 96 && argv[1][i] < 123)
               {
                      argv[1][i] = 219 - argv[1][i];
                      write(1, &argv[1][i], 1);
               }
               else
                      write(1, &argv[1][i], 1);
               i++;
        }
    }
    write(1, "\n", 1);
    return (0);
}
```

32.- max

Expected Files: max.c
Allowed functions:

Write the following function:

int max(int* tab, unsigned int len);

The first parameter is an array of int, the second is the number of elements in the array. The function returns the largest number found in the array. If the array is empty, the function returns 0.

```
#include <stdio.h>
int     max(int *tab, unsigned int len)
{
    int max;
    unsigned int i;
    i = 0;
    if (len)
    {
        max = tab[0];
        while (i < len)
        {
               if    (tab[i] > max)
                      max = tab[i];
               i++;
        }
        return (max);
    }
    else
        return (0);
}
int     main(void)
{
    int    n1[5] =  {10, 4, 5, 66, 6};
    int n2[5] = {-20, -55, -5, -10, -4};
    int    n3[5];
    printf("%d\n", max(n1, 5));
    printf("%d\n", max(n2, 5));
    printf("%d\n", max(n3, 0));
}
```

33.- wdmatch

Expected Files: wdmatch.c
Allowed functions: write

Write a program that takes two strings and checks whether it's possible to write the first string with characters from the second string, while respecting the order in which these characters appear in the second string.
If it's possible, the program displays the string, followed by a \n, otherwise it simply displays a \n.
If the number of arguments is not 2, the program displays a \n.
Examples:

```
$>./wdmatch "faya" "fgvvfdxcacpolhyghbreda" | cat -e
faya$
$>./wdmatch "faya" "fgvvfdxcacpolhyghbred" | cat -e
$
$>./wdmatch "quarante deux" "qfqfsudf arzgsayns tsregfdgs sjytdekuoixq " | cat -e
quarante deux$
$>./wdmatch "error" rrerrrfiiljdfxjyuifrrvcoojh | cat -e
$
$>./wdmatch | cat -e
$
```

```c
#include <unistd.h>
void    wdmatch(char *s1, char *s2)
{
    int len = 0;
    int i = 0;

    while (s1[len])
        ++len;
    while (*s2 && i < len)
        (*s2++ == s1[i]) ? ++i : 0;
    if (i == len)
        write(1, s1, len);
}
int     main(int ac, char **av)
{
    if (ac == 3)
        wdmatch(av[1], av[2]);
    write(1, "\n", 1);
    return (0);
}
```

34.- wdmatch

Expected Files: wdmatch.c
Allowed functions: write

Write a program that takes two strings and checks whether it's possible to write the first string with characters from the second string, while respecting the order in which these characters appear in the second string. If it's possible, the program displays the string, followed by a \n, otherwise it simply displays a \n. If the number of arguments is not 2, the program displays a \n.

Examples:

```
$>./wdmatch "faya" "fgvvfdxcacpolhyghbreda" | cat -e
faya$
$>./wdmatch "faya" "fgvvfdxcacpolhyghbred" | cat -e
$
$>./wdmatch "quarante deux" "qfqfsudf arzgsayns tsregfdgs sjytdekuoixq " | cat -e
quarante deux$
$>./wdmatch "error" rrerrrfiiljdfxjyuifrrvcoojh | cat -e
$
$>./wdmatch | cat -e
$
```

```c
#include <unistd.h>
void    wdmatch(char *s1, char *s2)
{
    int len = 0;
    int i = 0;
    while (s1[len])
        ++len;
    while (*s2 && i < len)
        (*s2++ == s1[i]) ? ++i : 0;
    if (i == len)
        write(1, s1, len);
}

int      main(int ac, char **av)
{
    if (ac == 3)
        wdmatch(av[1], av[2]);
    write(1, "\n", 1);
    return (0);
}
```

35.- do_op

Expected Files: *.c, *.h
Allowed functions: atoi, printf, write

Write a program that takes three strings:
- The first and the third one are representations of base-10 signed integers that fit in an int.
- The second one is an arithmetic operator chosen from: + - * / %

The program must display the result of the requested arithmetic operation, followed by a newline. If the number of parameters is not 3, the program just displays a newline. You can assume the string have no mistakes or extraneous characters. Negative numbers, in input or output, will have one and only one leading '-'. The result of the operation fits in an int.

Examples:

```
$> ./do_op "123" "*" 456 | cat -e
56088$
$> ./do_op "9828" "/" 234 | cat -e
42$
$> ./do_op "1" "+" "-43" | cat -e
-42$
$> ./do_op | cat -e
$
```

```c
#include <stdio.h>
#include <stdlib.h>
int      main(int argc, char **argv)
{
    if    (argc == 4)
    {
        if (argv[2][0] == '+')
               printf("%d", (atoi(argv[1]) + atoi(argv[3])));
        if (argv[2][0] == '-')
               printf("%d", (atoi(argv[1]) - atoi(argv[3])));
        if (argv[2][0] == '*')
               printf("%d", (atoi(argv[1]) * atoi(argv[3])));
        if (argv[2][0] == '/')
               printf("%d", (atoi(argv[1]) / atoi(argv[3])));
        if (argv[2][0] == '%')
               printf("%d", (atoi(argv[1]) % atoi(argv[3])));
    }
    printf("\n");
    return (0);
}
```

36.- print_bits

Expected Files: print_bits.c
Allowed functions: write

Write a function that takes a byte, and prints it in binary WITHOUT A NEWLINE AT THE END.
Your function must be declared as follows:
void   print_bits(unsigned char octet);
Example, if you pass 2 to print_bits, it will print "00000010"

```c
#include <unistd.h>

void    print_bits(unsigned char octet)
{
    int    i;

    i = 128;
    while (octet >= 0 && i)
    {
        (octet / i) ? write(1, "1", 1) : write(1, "0", 1);
        (octet / i) ? octet -= i : 0;
        i /= 2;
    }
}

void    print_bits2(unsigned char octet)
{
    int    i = 256;
    while (i >>= 1)
        (octet & i) ? write(1, "1", 1) : write(1, "0", 1);
}

int     main(void)//
{//
    int n = 64;
    print_bits(n);//
    write(1, "\n", 1);//
    print_bits2(n);//
}//
```

37.- ft_strcmp

Expected Files: ft_strcmp.c
Allowed functions:

Reproduce the behavior of the function strcmp (man strcmp). Your function must be declared as follows:

int ft_strcmp(char *s1, char *s2);

```c
#include <stdio.h>
#include <string.h>

int     ft_strcmp(char *s1, char *s2)
{
    int i;

    i = 0;
    while (s1[i] != '\0' && s2[i] != '\0' && s1[i] == s2[i])
        i++;
    return(s1[i] - s2[i]);
}

int     main(void)
{
    printf("%d\n", ft_strcmp("same","same"));
    printf("%d\n", ft_strcmp("notsame", "nsame"));
    printf("%d\n", strcmp("same", "same"));
    printf("%d\n", strcmp("notsame", "nsame"));
    return (0);
}

#include "libft.h"
int     ft_strcmp(char *s1, char *s2)
{
    int i;
    i = 0;
    while (s1[i] != '\0' && s2[i] != '\0' && s1[i] == s2[i])
    {
        i++;
    }
    return (s1[i] - s2[i]);
}
```

38.- ft_strrev

Expected Files: ft_strrev.c

Allowed functions:

Write a function that reverses (in-place) a string. It must return its parameter.

Your function must be declared as follows:

char   *ft_strrev(char *str);

```c
char    *ft_strrev(char *str)
{
    int   count;
    int   i;
    char  c;
    count = 0;
    while (str[count] != '\0')
        count++;
    count = count - 1;
    i = 0;
    while (i < ((count + 1) / 2))
    {
        c = str[i];
        str[i] = str[count - i];
        str[count - i] = c;
        i++;
    }
    return (str);
}
```

39.- is_power_of_2

Expected Files: is_power_of_2.c

Allowed functions: None

Write a function that determines if a given number is a power of 2. This function returns 1 if the given number is a power of 2, otherwise it returns 0. Your function must be declared as follows:

int   is_power_of_2(unsigned int n);

```c
int     is_power_of_2(unsigned int n)
{
    if (n == 0)
        return (0);
    while (n % 2 == 0)
        n /= 2;
    return ((n == 1) ? 1 : 0);
}
```

40.- add_prime_sum

Expected Files: add_prime_sum.c
Allowed functions: write, exit

Write a program that takes a positive integer as argument and displays the sum of all prime numbers inferior or equal to it followed by a newline.
If the number of arguments is not 1, or the argument is not a positive number, just display 0 followed by a newline.
Yes, the **Example**s are right.
Examples:

```
$>./add_prime_sum 5
10
$>./add_prime_sum 7 | cat -e
17$
$>./add_prime_sum | cat -e
0$
```

```c
#include <unistd.h>
int		ft_atoi(char *str)
{
	int		i;
	int		sign;
	int		nbr;
	i = 0;
	sign = 1;
	nbr = 0;
	if (!str[i])
		return (0);
	while (str[i] == ' ' || str[i] == '\n' || str[i] == '\f' \
		|| str[i] == '\v' || str[i] == '\r' || str[i] == '\t')
		i += 1;
	if (str[i] == '-' || str[i] == '+')
		if (str[i++] == '-')
			sign = -1;
	while (str[i] >= '0' && str[i] <= '9')
		nbr = (nbr * 10) + (str[i++] - '0');
	return (nbr * sign);
}
```

```c
void	ft_putnbr(int nb)
{
	char	c;
	if (nb < 0)
	{
		nb = -nb;
		write(1, "-", 1);
	}
	if (nb < 10)
	{
		c = nb + '0';
		write(1, &c, 1);
	}
	else
	{
		ft_putnbr(nb / 10);
		ft_putnbr(nb % 10);
	}
}
```

```c
int		is_prime(int nb)
{
	int i;

	i = 2;
	if (nb <= 1)
		return (0);
	while (i <= (nb / 2))
	{
		if (!(nb % i))
			return (0);
		else
			i += 1;
	}
	return (1);


}
```

```c
int		main(int argc, char *argv[])
{
	int		nb;
	int		sum;
	if (argc == 2)
	{
		nb = ft_atoi(argv[1]);
		sum = 0;
		while (nb > 0)
			if (is_prime(nb--))
				sum += (nb + 1);
		ft_putnbr(sum);
	}
	write(1, "\n", 1);
	return (0);
}
```

41.- epur_str

Expected Files: epur_str.c
Allowed functions: write

Write a program that takes a string, and displays this string with exactly one space between words, with no spaces or tabs either at the beginning or the end, followed by a \n. A "word" is defined as a part of a string delimited either by spaces/tabs, or by the start/end of the string. If the number of arguments is not 1, or if there are no words to display, the program displays \n.

Example:

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

```c
#include <unistd.h>
int     ft_strlen(char *s)
{
    int i;
    i = 0;
    while (s[i])
        i++;
    return (i);
}
int     ft_isblank(char c)
{
    if (c == ' ' || c == '\t')
        return (1);
    if (c >= 9 && c <= 13)
        return (1);
    return (0);
}

void    epurstr(char *s)
{
    int len = ft_strlen(s);
    while (len && ft_isblank(s[len - 1]))
        --len;
    while (len && ft_isblank(*s) && *s++)
        --len;
    while (len--)
    {
        if (!ft_isblank(*s) || (*(s + 1) && !ft_isblank(*(s + 1))))
               write(1, s, 1);
        s++;
    }
}

int     main(int ac, char **av)
{
    if (ac == 2 && *av[1])
        epurstr(av[1]);
    write(1, "\n", 1);
    return (0);
}
```

42.- ft_list_size

Expected Files: ft_list_size.c, ft_list.h
Allowed functions:

Write a function that returns the number of elements in the linked list that's passed to it. It must be declared as follows:
int   ft_list_size(t_list *begin_list);
You must use the following structure, and turn it in as a file called ft_list.h:

```
typedef struct        s_list
{
            struct s_list *next;
            void        *data;
}           t_list;
```

```
#include "ft_list.h"

int     ft_list_size(t_list *begin_list)
{
    int         i;

    i = 0;
    while (begin_list)
    {
        begin_list = begin_list->next;
        ++i;
    }
    return (i);
}
#ifndef FT_LIST_H
# define FT_LIST_H

typedef struct         s_list
{
    struct    s_list    *next;
    void               *data;
}                            t_list;

int     ft_list_size(t_list *begin_list);

#endif
```

42.- ft_rrange

Expected Files: ft_rrange.c
Allowed functions: malloc

Write the following function:

```
int	*ft_rrange(int start, int end);
```

It must allocate (with malloc()) an array of integers, fill it with consecutive values that begin at end and end at start (Including start and end !), then return a pointer to the first value of the array.

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.

```c
#include <stdlib.h>
#include <stdio.h>

int     *ft_rrange(int start, int end)
{
    int *ret;
    int len;
    int i;

    len = (end - start);
    if (start < 0 && end < 0)
        len = ((start * -1) - (end * -1));
    ret = (int *)malloc(sizeof(int) * (len + 1));
    if (!ret)
        return (NULL);
    i = 0;
    while (start <= end)
    {
        ret[i] = end;
        end--;
        i++;
    }
    return (ret);
}

int     main(void)
{
    int *nums;
    int i;
    int len;
    int start;
    int end;

    i = 0;
    start = -10;
    end = -5;
    len = (end - start);
    if (start < 0 && end < 0)
        len = ((start * -1) - (end * -1));
    nums = ft_rrange(start, end);
    while (i <= len)
    {
        printf("%d\n", nums[i]);
        i++;
    }
    return (0);
}
```

43.- hidenp

Expected Files: hidenp.c
Allowed functions: write

Write a program named hidenp that takes two strings and displays 1 followed by a newline if the first string is hidden in the second one, otherwise displays 0 followed by a newline. Let s1 and s2 be strings. We say that s1 is hidden in s2 if it's possible to find each character from s1 in s2, in the same order as they appear in s1. Also, the empty string is hidden in any string. If the number of parameters is not 2, the program displays a newline.

Examples :

```
$>./hidenp "fgex.;" "tyf34gdgf;'ektufjhgdgex.;.;rtjynur6" | cat -e
1$
$>./hidenp "abc" "2altrb53c.sse" | cat -e
1$
$>./hidenp "abc" "btarc" | cat -e
0$
$>./hidenp | cat -e
$
$>
```

```c
#include <unistd.h>

int     main(int argc, char **argv)
{
    int     i;
    int     j;
    i = 0;
    j = 0;
    if (argc == 3)
    {
        while (argv[2][j] != '\0')
        {
               if (argv[1][i] == argv[2][j])
                      i++;
               if (argv[1][i] == '\0')
               {
                      write(1, "1\n", 2);
                      return (0);
               }
               j++;
        }
        write(1, "0", 1);
    }
    write(1, "\n", 1);
    return (0);
}
```

44.- pgcd

Expected Files: pgcd.c
Allowed functions: printf, atoi, malloc, free

Write a program that takes two strings representing two strictly positive integers that fit in an int. Display their highest common denominator followed by a newline (It's always a strictly positive integer). If the number of parameters is not 2, display a newline.

Examples:

```
$> ./pgcd 42 10 | cat -e
2$
$> ./pgcd 42 12 | cat -e
6$
$> ./pgcd 14 77 | cat -e
7$
$> ./pgcd 17 3 | cat -e
1$
$> ./pgcd | cat -e
$
```

```c
#include <stdlib.h>
#include <stdio.h>
void    pgcd(int nb1, int nb2)
{
    int div;
    int pgcd;

    div = 1;
    if (nb1 <= 0 || nb2 <= 0)
        return;
    while (div <= nb1 || div <= nb2)
    {
        if (nb1 % div == 0 && nb2 % div== 0)
                pgcd = div;
        div = div + 1;
    }
    printf("%d", pgcd);
}
int      main(int argc, char **argv)
{
    if (argc == 3)
        pgcd(atoi(argv[1]), atoi(argv[2]));
    printf("\n");
    return (0);
}
```

45.- print_hex

Expected Files: print_hex.c
Allowed functions: write

Write a program that takes a positive (or zero) number expressed in base 10, and displays it in base 16 (lowercase letters) followed by a newline. If the number of parameters is not 1, the program displays a newline.

Examples:

```
$> ./print_hex "10" | cat -e
a$
$> ./print_hex "255" | cat -e
ff$
$> ./print_hex "5156454" | cat -e
4eae66$
$> ./print_hex | cat -e
v2
```

```c
#include <unistd.h>
int      ft_atoi(char *str)
{
    int         i;
    int         nbr;
    int         sign;

    i = 0;
    sign = 1;
    nbr = 0;
    while(str[i] == ' ' || str[i] == '\f' ||
str[i] == '\v' || str[i] == '\r'
    || str[i] == '\n' || str[i] == '\t' )
        i++;
    if (str[i] == '-')
        sign = -1;
    if (str[i] == '+' || str[i] == '-')
        i++;
    while (str[i] >= 48 && str[i] <= 57)
    {
        nbr *= 10;
        nbr += str[i] - '0';
        i++;
    }
    nbr *= sign;
    return (nbr);
}
```

```c
int      main(int argc, char **argv)
{
    if (argc == 2)
    {
     int     value;
     int i;
     int str[64];
     char base[16] =
{'0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f'};

     value = ft_atoi(argv[1]);
     i = 0;
     if (value < 0)
     {
        write(1, "\n", 1);
        return (0);
     }
     if (value == 0)
     {
        write(1,"0\n", 2);
        return (0);
     }
     while (value != 0)
     {
        str[i] = value % 16;
        value = value / 16;
        i++;
     }
     i--;
     while (i >= 0)
     {
        write(1, &base[str[i]], 1);
        i--;
     }
    }
    write(1, "\n", 1);
    return(0);
}
```

```c
void    print_hex(int n)
{
    if (n >= 16)
        print_hex(n / 16);
    n = n % 16;
    n += n < 10 ? '0' : 'a' - 10;
    write(1, &n, 1);
}
```

```c
int      main(int ac, char **av)
{
    if (ac == 2)
        print_hex(ft_atoi(av[1]));
    write(1, "\n", 1);
    return (1);
}
```

46.- rstr_capitalizer

Expected Files: rstr_capitalizer.c
Allowed functions: write

Write a program that takes one or more strings and, for each argument, puts the last character of each word (if it's a letter) in uppercase and the rest in lowercase, then displays the result followed by a \n. A word is a section of string delimited by spaces/tabs or the start/end of the string. If a word has a single letter, it must be capitalized. If there are no parameters, display \n.

Examples:

```
$> ./rstr_capitalizer | cat -e
$
$> ./rstr_capitalizer "Premier PETIT TesT" | cat -e
premieR petiT tesT$
$> ./rstr_capitalizer "DeuxiEmE tEST uN PEU moinS  facile" "   attention C'EST pas dur QUAND mEmE" "ALLer UN DeRNier 0123456789pour LA
rouTE      E " | cat -e
deuxiemE tesT uN peU moinS  facilE$
   attentioN c'esT paS duR quanD memE$
alleR uN dernieR 0123456789pouR lA routE      E $
```

```c
#include <unistd.h>
void	ft_putchar(char c)
{
	write(1, &c, 1);
}
void	rstr_capitalizer(int argc, char **argv)
{
	int i;
	int j;
	i = 1;
	j = 0;
	while (i < argc)
	{
		j = 0;
		while (argv[i][j] != '\0')
		{
			if (argv[i][j] >= 'A' && argv[i][j] <= 'Z' )
				argv[i][j] += 32;
			if (argv[i][j + 1] == ' ' || argv[i][j + 1] == '\t' || argv[i][j + 1] == '\0')
			{
				if (argv[i][j] >= 'a' && argv[i][j] <= 'z')
					argv[i][j] -= 32;
			}
			ft_putchar(argv[i][j]);
			j++;
		}
		ft_putchar('\n');
		i++;
	}
}
int		main(int argc, char **argv)
{
	if (argc > 1)
		rstr_capitalizer(argc, argv);
	else
		ft_putchar('\n');
	return (0);
}
```

47.- expand_str

Expected Files: expand_str.c
Allowed functions: write

Write a program that takes a string and displays it with exactly three spaces between each word, with no spaces or tabs either at the beginning or the end, followed by a newline. A word is a section of string delimited either by spaces/tabs, or by the start/end of the string. If the number of parameters is not 1, or if there are no words, simply display a newline.

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
```

```c
#include <unistd.h>

void	ft_putchar(char c)
{
	write(1, &c, 1);
}

void	rstr_capitalizer(int argc, char **argv)
{
	int i;
	int j;

	i = 1;
	j = 0;
	while (i < argc)
	{
		j = 0;
		while (argv[i][j] != '\0')
		{
			if (argv[i][j] >= 'A' && argv[i][j] <= 'Z' )
				argv[i][j] += 32;
			if (argv[i][j + 1] == ' ' || argv[i][j + 1] == '\t' || argv[i][j + 1] == '\0')
			{
				if (argv[i][j] >= 'a' && argv[i][j] <= 'z')
					argv[i][j] -= 32;
			}
			ft_putchar(argv[i][j]);
			j++;
		}
		ft_putchar('\n');
		i++;
	}
}

int		main(int argc, char **argv)
{
	if (argc > 1)
		rstr_capitalizer(argc, argv);
	else
		ft_putchar('\n');
	return (0);
}
```

48.- tab_mult

Expected Files: tab_mult.c
Allowed functions: write

Write a program that displays a number's multiplication table. The parameter will always be a strictly positive number that fits in an int, and said number times 9 will also fit in an int. If there are no parameters, the program displays \n.

Examples:

$>./tab_mult 9 1 x 9 = 9 2 x 9 = 18 3 x 9 = 27 4 x 9 = 36 5 x 9 = 45 6 x 9 = 54 7 x 9 = 63 8 x 9 = 72 9 x 9 = 81 $>	$>./tab_mult 19 1 x 19 = 19 2 x 19 = 38 3 x 19 = 57 4 x 19 = 76 5 x 19 = 95 6 x 19 = 114 7 x 19 = 133 8 x 19 = 152 9 x 19 = 171 $>	$>./tab_mult | cat -e $>

```c
#include <unistd.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}
void    ft_putstr(char *str)
{
    while(*str)
    {
        ft_putchar(*str);
        str++;
    }
}
void    ft_putnbr(int num)
{
    if (num < 0)
    {
        ft_putchar('-');
        num *= -1;
    }
    if (num >= 10)
    {
        ft_putnbr(num / 10);
        ft_putnbr(num % 10);
    }
    else
        ft_putchar(num + '0');
}
```

```c
int     ft_atoi(char *str)
{
    int sign;
    int number;

    sign = 1;
    number = 0;
    while (*str == ' ' || *str == '\t' || *str ==
'\n' || *str == '\v'
                || *str == '\f' || *str == '\r')
        str++;
    if (*str == '-')
    {
        sign = -1;
        str++;
    }
    while (*str && *str >= '0' && *str <= '9')
    {
       number *= 10;
       number += *str - '0';
       str++;
    }
    return (number * sign);
}
```

```c
void    tab_mult(char **argv)
{
    int i;
    int number;
    i = 1;
    number = ft_atoi(argv[1]);
    while (i < 10)
    {
        ft_putnbr(i);
        ft_putstr(" x  ");
        ft_putnbr(number);
        ft_putstr(" = ");
        ft_putnbr(number * i);
        ft_putstr("\n");
        i++;
    }
}
```

```c
int     main(int argc, char **argv)
{
    if (argc == 2 && ft_atoi(argv[1]) >= 0 &&
ft_atoi(argv[1]) <= 238609294)
        tab_mult(argv);
    else
        ft_putchar('\n');
    return (0);
}
```

49.- ft_atoi_base

Expected Files: ft_atoi_base.c
Allowed functions: None

Write a function that converts the string argument str (base N <= 16) to an integer (base 10) and returns it. The characters recognized in the input are: 0123456789abcdef. Those are, of course, to be trimmed according to the requested base. For **Example**, base 4 recognizes "0123" and base 16 recognizes "0123456789abcdef". Uppercase letters must also be recognized: "12fdb3" is the same as "12FDB3". Minus signs ('-') are interpreted only if they are the first character of the string.

Your function must be declared as follows:
int   ft_atoi_base(const char *str, int str_base);

```
int isblank(char c)
{
    if (c <= 32)
        return (1);
    return (0);
}

int      isvalid(char c, int base)
{
    char digits[17] = "0123456789abcdef";
    char digits2[17] = "0123456789ABCDEF";

    while (base--)
        if (digits[base] == c || digits2[base] == c)
                return (1);
    return (0);
}

int      value_of(char c)
{
    if (c >= '0' && c <= '9')
        return (c - '0');
    else if (c >= 'a' && c <= 'f')
        return (c - 'a' + 10);
    else if (c >= 'A' && c <= 'F')
        return (c - 'A' + 10);
    return (0);
}

int      ft_atoi_base(const char *str, int str_base)
{
    int result;
    int sign;

    result = 0;
    while (isblank(*str))
        str++;
    sign = (*str == '-') ? -1 : 1;
    (*str == '-' || *str == '+') ? ++str : 0;
    while (isvalid(*str, str_base))
        result = result * str_base + value_of(*str++);
    return (result * sign);
}
```

50.- ft_range

Expected Files: ft_range.c
Allowed functions: malloc ###marca###

Write the following function:
int   *ft_range(int start, int end);
It must allocate (with malloc()) an array of integers, fill it with consecutive values that begin at start and end at end (Including start and end !), then return a pointer to the first value of the array.
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>
int      *ft_range(int min, int max)
{
    int         n;
    int         *s;
    n = max >= min ? max - min : min - max;
    if (!(s = (int *)malloc(sizeof(int) * (n))))
        return (NULL);
    while (max != min)
        *s++ = max > min ? min++ : min--;
    *s = min;
    return (s - n);
}

#include "libft.h"
/*
** replace #include libft.h with #include <stdlib.h>
** replace ft_intnew(n - 1) with protected malloc(sizeof(int) * (n));
*/
int      *ft_range(int min, int max)
{
    int         n;
    int         *s;
    n = max >= min ? max - min : min - max;
    if (!(s = ft_intnew(n - 1)))
        return (NULL);
    while (max != min)
        *s++ = max > min ? min++ : min--;
    *s = min;
    return (s - n);
}
int      main(int ac, char **av)
{
    int         *s;
    int         n;
    int         min;
    int         max;

    min = ft_atoi(av[1]);
    max = ft_atoi(av[2]);
    n = max >= min ? max - min + 1 : min - max + 1;
    if (ac != 3)
        return (0);
    s = ft_range(min, max);
    while (*s && n--)
    {
        ft_putnbr(*s++);
        ft_putchar('\n');
    }
    return (1);
}
```

51.- paramsum

Expected Files: paramsum.c
Allowed functions: write

Write a program that displays the number of arguments passed to it, followed by a newline.
If there are no arguments, just display a 0 followed by a newline.

Example:

```
$>./paramsum 1 2 3 5 7 24
6
$>./paramsum 6 12 24 | cat -e
3$
$>./paramsum | cat -e
0$
```

```c
#include <unistd.h>
void	ft_putchar(char c)
{
    write(1, &c, 1);
}
void	ft_putnbr(int num)
{
    if (num < 0)
    {
        ft_putchar('-');
        num *= -1;
    }
    else if (num >= 10)
    {
        ft_putnbr(num / 10);
        ft_putnbr(num % 10);
    }
    else
        ft_putchar(num + '0');
}
int		main(int ac, char **av)
{
    char *str;
    if (ac == 1)
        ft_putchar('0');
    else
    {
        str = av[1];
        ft_putnbr(ac - 1);
    }
    ft_putchar('\n');
}
```

52.- str_capitalizer

Expected Files: str_capitalizer.c
Allowed functions: write

Write a program that takes one or several strings and, for each argument, capitalizes the first character of each word (If it's a letter, obviously), puts the rest in lowercase, and displays the result on the standard output, followed by a \n. A "word" is defined as a part of a string delimited either by spaces/tabs, or by the start/end of the string. If a word only has one letter, it must be capitalized. If there are no arguments, the progam must display \n.

Example:

```
$> ./str_capitalizer | cat -e
$
$> ./str_capitalizer "Premier PETIT TesT" | cat -e
Premier Petit Test$
$> ./str_capitalizer "DeuxiEmE tEST uN PEU moinS  facile" "   attention C'EST pas dur QUAND mEmE" "ALLer UN DeRNier 0123456789pour LA rouTE
            E " | cat -e
Deuxieme Test Un Peu Moins  Facile$
   Attention C'est Pas Dur Quand Meme$
Aller Un Dernier 0123456789pour La Route         E $
$>
```

```c
#include <unistd.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}
int     ft_isspace(char c)
{
    if (c == ' ' || c == '\t')
        return (1);
    return (0);
}

int     tolower(char c)
{
    return (c += (c >= 'A' && c <= 'Z') ? 32 : 0);
}
int     toupper(char c)
{
    return (c -= (c >= 'a' && c <= 'z') ? 32 : 0);
}
void    str_capitaliser(char *s)
{
    while (*s)
    {
        while (ft_isspace(*s))
                ft_putchar(*s++);
        if (*s && !ft_isspace(*s))
                ft_putchar(toupper(*s++));
        while (*s && !ft_isspace(*s))
                ft_putchar(tolower(*s++));
    }
}
int     main(int ac, char **av)
{
    if (ac > 1)
    {
        ++av;
        while (*av)
        {
                str_capitaliser(*av++);
                write(1, "\n", 1);
        }
    }
    return (0);
}
```

53.- fprime

Expected Files: fprime.c
Allowed functions: printf, atoi

Write a program that takes a positive int and displays its prime factors on the standard output, followed by a newline. Factors must be displayed in ascending order and separated by '*', so that the expression in the output gives the right result. If the number of parameters is not 1, simply display a newline. The input, when there's one, will be valid.

Examples:

```
$> ./fprime 225225 | cat -e
3*3*5*5*7*11*13$
$> ./fprime 8333325 | cat -e
3*3*5*5*7*11*13*37$
$> ./fprime 9539 | cat -e
9539$
$> ./fprime 804577 | cat -e
804577$
$> ./fprime 42 | cat -e
2*3*7$
$> ./fprime 1 | cat -e
1$
$> ./fprime | cat -e
$
$> ./fprime 42 21 | cat -e
```

```c
#include <stdio.h>
#include <stdlib.h>
int     main(int ac, char **av)
{
    int         n;
    int         nb;
    if (ac == 2)
    {
        if (av[1][0] == '\0')
        {
               printf("\n");
               return (0);
        }
        nb = atoi(av[1]);
        if (nb == 1)
        {
               printf("1\n");
               return (0);
        }
        while (1)
        {
               n = 1;
               while (++n <= nb)
               {
                       if (nb % n == 0)
                       {
                              printf("%d", n);
                              nb = nb / n;
                              break ;
                       }
               }
               if (nb == 1)
                       break ;
               else
                       printf("*");
        }
    }
    printf("\n");
    return (0);
}
```

54.- ft_list_foreach

Expected Files: ft_list_foreach.c, ft_list.h
Allowed functions:

Write a function that takes a list and a function pointer, and applies this function to each element of the list. It must be declared as follows:

```
void        ft_list_foreach(t_list *begin_list, void (*f)(void *));
```

The function pointed to by f will be used as follows:

```
(*f)(list_ptr->data);
```

You must use the following structure, and turn it in as a file called ft_list.h:

```
typedef struct        s_list
{
            struct s_list *next;
            void        *data;
}           t_list;
```

```
#include "ft_list.h"
void    ft_list_foreach(t_list *begin_list, void (*f)(void *))
{
    t_list    *list_ptr;
    list_ptr = begin_list;
    while (list_ptr)
    {
        (*f)(list_ptr->data);
        list_ptr = list_ptr->next;
    }
}
```

```
#include "ft_list.h"

void    ft_list_foreach(t_list *begin_list, void (*f)(void *))
{
    t_list    *list_ptr;

    list_ptr = begin_list;
    while (list_ptr)
    {
        (*f)(list_ptr->data);
        list_ptr = list_ptr->next;
    }
}
```

55.- ft_split

Expected Files: ft_split.c
Allowed functions: malloc

Write a function that takes a string, splits it into words, and returns them as a NULL-terminated array of strings. A "word" is defined as a part of a string delimited either by spaces/tabs/new lines, or by the start/end of the string. Your function must be declared as follows:

char   **ft_split(char *str);

```c
#include <stdlib.h>

int                    ft_isspace(char c)
{
    return ((c == ' ' || c == '\n' || c == '\t'
                || c == '\r' || c == '\v' || c == '\f') ? 1 : 0);
}

static int      r_size(char *s)
{
    unsigned int len;

    len = 0;
    while (*s)
    {
        if (ft_isspace(*s))
                ++s;
        else
        {
                ++len;
                while (*s && !ft_isspace(*s))
                        ++s;
        }
    }
    return (len);
}

char            **ft_split(char *s)
{
    int         i = 0;
    int         j = 0;
    int         k;
    char    **r;
    int         w_len = 0;

    if (!(r = (char **)malloc(sizeof(char*) * (r_size(s) + 1))))
        return (0);
    while (s[i] && j < r_size(s))
    {
        while (s[i] && ft_isspace(s[i]))
                i++;
        while (s[i] && !ft_isspace(s[i]))
        {
                w_len++;
                i++;
        }
        if (!(r[j] = (char *)malloc(sizeof(char) * (w_len + 1))))
                return (0);
        k = 0;
        while (w_len)
                r[j][k++] = s[i - w_len--];
        r[j++][k] = '\0';
    }
    return (r);
}
```

56.- rev_wstr

Expected Files: rev_wstr.c
Allowed functions: write, malloc, free

Write a program that takes a string as a parameter, and prints its words in reverse order. A "word" is a part of the string bounded by spaces and/or tabs, or the begin/end of the string. If the number of parameters is different from 1, the program will display '\n'. In the parameters that are going to be tested, there won't be any "additional" spaces (meaning that there won't be additional spaces at the beginning or at he end of the string, and words will always be separated by exactly one space).

Examples:

```
$> ./rev_wstr "le temps du mepris precede celui de l'indifference" | cat -e
l'indifference de celui precede mepris du temps le$
$> ./rev_wstr "abcdefghijklm"
abcdefghijklm
$> ./rev_wstr "il contempla le mont" | cat -e
mont le contempla il$
$> ./rev_wstr | cat -e
$
$>
```

```c
#include <unistd.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}

void    put_word(char *str)
{
    while (*str && *str != ' ' && *str != '\t')
        write(1, str++, 1);
}
int             main(int ac, char **av)
{
    int         i;
    if (ac == 2)
    {
        i = 0;
        while (av[1][i])
                i++;
        i--;
        while (av[1][i] && i > 0)
        {
                while (av[1][i] != ' ' && av[1][i] != '\t' && i > 0)
                        i--;
                put_word(av[1] + i + (i == 0 ? 0 : 1));
                if (i > 0)
                        ft_putchar(' ');
                while ((av[1][i] == ' ' || av[1][i] == '\t') && i > 0)
                        i--;
        }
    }
    ft_putchar('\n');
    return (0);
}
```

57.- ft_list_remove_if

Expected Files: ft_list_remove_if.c

Allowed functions: free

Write a function called ft_list_remove_if that removes from the passed list any element the data of which is "equal" to the reference data. It will be declared as follows :

void ft_list_remove_if(t_list **begin_list, void *data_ref, int (*cmp)());

cmp takes two void* and returns 0 when both parameters are equal. You have to use the ft_list.h file, which will contain:

```
$>cat ft_list.h
typedef struct      s_list
{
            struct s_list   *next;
            void        *data;
}           t_list;
$>
```

```
#include "ft_list.h"
#include <stdlib.h>

void    ft_list_remove_if(t_list **begin_list, void *data_ref, int (*cmp)())
{
    t_list    *tmp;
    t_list    *i;

    while (*begin_list && cmp((*begin_list)->data, data_ref) == 0)
    {
        tmp = *begin_list;
        *begin_list = (*begin_list)->next;
        free(tmp);
    }
    i = *begin_list;
    while (i && i->next)
    {
        if (cmp(i->next->data, data_ref) == 0)
        {
                tmp = i->next;
                i->next = tmp->next;
                free (tmp);
        }
        i = i->next;
    }
}
#ifndef FT_LIST_H

# define FT_LIST_H

typedef struct         s_list
{
    struct s_list    *next;
    void               *data;
}                             t_list;

#endif
```

58.- ft_list_remove_if

Expected Files: ft_list_remove_if.c
Allowed functions: free

Write a function called ft_list_remove_if that removes from the passed list any element the data of which is "equal" to the reference data.
It will be declared as follows :
void ft_list_remove_if(t_list **begin_list, void *data_ref, int (*cmp)());
cmp takes two void* and returns 0 when both parameters are equal.
You have to use the ft_list.h file, which will contain:

```
$>cat ft_list.h
typedef struct        s_list
{
              struct s_list   *next;
              void         *data;
}             t_list;
$>
```

```
#ifndef FT_LIST_H
# define FT_LIST_H
typedef struct         s_list
{
    struct s_list    *next;
    void               *data;
}                             t_list;

#endif

#include "ft_list.h"
#include <stdlib.h>
void    ft_list_remove_if(t_list **begin_list, void *data_ref, int (*cmp)())
{
    t_list    *tmp;
    t_list    *i;
    while (*begin_list && cmp((*begin_list)->data, data_ref) == 0)
    {
        tmp = *begin_list;
        *begin_list = (*begin_list)->next;
        free(tmp);
    }
    i = *begin_list;
    while (i && i->next)
    {
        if (cmp(i->next->data, data_ref) == 0)
        {
               tmp = i->next;
               i->next = tmp->next;
               free (tmp);
        }
        i = i->next;
    }
}
```

59.- sort_int_tab

Expected Files: sort_int_tab.c
Allowed functions:

Write the following function:
void sort_int_tab(int *tab, unsigned int size);
It must sort (in-place) the 'tab' int array, that contains exactly 'size' members, in ascending order.
Doubles must be preserved. Input is always coherent.

```
#include <stdio.h>
void    ft_swap(int *a, int *b)
{
    *a += *b;
    *b = *a - *b;
    *a = *a - *b;
}

void    sort_int_tab_withmain(int *tab, unsigned int size)
{
    unsigned    a = 0;
    unsigned    i = 0;
    while (i < size)
    {
        if (tab[i] > tab[i + 1])
        {
                ft_swap(&tab[i], &tab[i + 1]);
                a = i;
        }
        while (a)
        {
                if (tab[a - 1] > tab[a])
                {
                        ft_swap(&tab[a - 1], &tab[a]);
                        --a;
                }
                else
                        a = 0;
        }
        ++i;
    }
}

int     main(void)
{
    int a[6] = {9, 7, 6, 4, 5, 10};
    int i = 0;
    int size = 6;
    sort_int_tab_withmain(a, size);
    while (i < size)
        printf("%d", a[i++]);
    return (0);
}
```

60.- ft_itoa

Expected Files: ft_itoa.c
Allowed functions: malloc

Write a function that takes an int and converts it to a null-terminated string.
The function returns the result in a char array that you must allocate.
Your function must be declared as follows:
char	*ft_itoa(int nbr);

```
#include "libft.h"

static void	itoa_isnegative(int *n, int *negative)
{
	if (*n < 0)
	{
	*n *= -1;
	*negative = 1;
	}
}

char		*ft_itoa(int n)
{
	int	tmpn;
	int	len;
	int	negative;
	char	*str;

	if (n == -2147483648)
	return (ft_strdup("-2147483648"));
	tmpn = n;
	len = 2;
	negative = 0;
	itoa_isnegative(&n, &negative);
	while (tmpn /= 10)
	len++;
	len = len + negative;
	if ((str = (char*)malloc(sizeof(char) * len)) == NULL)
	return (NULL);
	str[--len] = '\0';
	while (len--)
	{
	str[len] = n % 10 + '0';
	n = n / 10;
	}
	if (negative)
	str[0] = '-';
	return (str);
}
```

61.- checkmate

Expected Files: *.c, *.h
Allowed functions: write, malloc, free

Write a program who takes rows of a chessboard in argument and check if your King is in a check position. Chess is played on a chessboard, a squared board of 8-squares length with specific pieces on it : King, Queen, Bishop, Knight, Rook and Pawns. For this exercice, you will only play with Pawns, Bishops, Rooks and Queen... and obviously a King. Each piece have a specific method of movement, and all patterns of capture are detailled in the **Example**s.txt file.
A piece can capture only the first ennemy piece it founds on its capture patterns.
The board have a variable size but will remains a square. There's only one King and all other pieces are against it. All other characters except those used for pieces are considered as empty squares.
The King is considered as in a check position when an other enemy piece can capture it. When it's the case, you will print "Success" on the standard output followed by a newline, otherwise you will print "Fail" followed by a newline.
If there is no arguments, the program will only print a newline.
Examples:

```
$> ./chessmate '..' '.K' | cat -e
Fail$
$> ./chessmate 'R...' '..P.' '.K..' '....' | cat -e
Success$
$> ./chessmate 'R...' 'iheK' '....' 'jeiR' | cat -e
Success$
$> ./chessmate | cat -e
$
$>
```

```c
#include <stdlib.h>
#include <unistd.h>
// gcc checkmate.c && ./a.out '..R.' '.Q..' '..BK' '...P' | cat -e
// B = 3       Rook = 2 and Queen == 6   an P == 1
int     checkmate(int ac, char **av)
{
    int         y = 0;
    int         x = 0;
    int         len = 0;
    int         b = 0;
    int         a = 0;
    char    **m;
   // creating map
    while (ac-- > 1)
        len++;
    if (!(m = (char **)malloc(sizeof(char *) * len * (len + 1))))
        return (0);
    y = 0;
    while (y < len)
    {
        if (!(m[y] = (char *)malloc(sizeof(char) * (len + 1))))
                return (0);
        x = 0;
        while (av[y + 1][x])
        {
                m[y][x] = av[y + 1][x];
                if (m[y][x] == 'K')
                {
                        a = x;
                        b = y;
                }
                ++x;
        }
        m[y][x] = 0;
        ++y;
    }

    // checking if the King is endangered by a pawn
    if (m[b + 1][a + 1] == 'P' || m[b + 1][a - 1] == 'P')
        return (0);

    int         i = 0;
    while (i < len)
    {
        if (m[b][i] == 'Q' || m[i][a] == 'Q' || m[b][i] == 'R' || m[i][a] == 'R')
```

```
               return (0);
        if (i < b)
        {
               if (i < a && (m[b - i - 1][a - i - 1] == 'B' || m[b - i - 1][a - i - 1] == 'Q') )
                      return (0);
               if (a + i < len && (m[b - i - 1][a + i + 1] == 'B' || m[b - i - 1][a + i + 1] == 'Q'))
                      return (0);
        }
        if (b + i < len)
        {
               if (i < a && (m[b + i + 1] [a - i - 1] == 'B' || m[b + i + 1] [a - i - 1] == 'Q'))
                      return (0);
               if (a + i < len && (m[b + i + 1] [a + i + 1] == 'B' || m[b + i + 1] [a + i + 1] == 'Q' ))
                      return (0);
        }
        i++;
    }
    return (1);
}
```

```
void    print(int ac, char **av)
{
    int         len;
    int         x;
    int         y;
    char    **m;

    // creating map
    while (ac-- > 1)
        len++;
    if (!(m = (char **)malloc(sizeof(char *) * len *
(len + 1))))
        return ;
    y = 0;
    while (y < len)
    {
        if (!(m[y] = (char *)malloc(sizeof(char) *
(len + 1))))
               return ;
        x = 0;
        while (av[y + 1][x])
        {
               m[y][x] = av[y + 1][x];
               ++x;
        }
        m[y][x] = 0;
        ++y;
    }
    // printing map
    y = 0;
    while (y < len)
    {
        write(1, m[y++], len);
        write(1, "\n", 1);
    }
}
```

```
int     main(int ac, char **av)
{
    if (ac > 1 && checkmate(ac, av))
        write(1, "Success\n", 8);
    else
        write(1, "Fail\n", 5);
    print(ac, av);
    return (0);
}
```

62.- rostring

Expected Files: rostring.c
Allowed functions: write, malloc, free

Write a program that takes a string and displays this string after rotating it one word to the left. Thus, the first word becomes the last, and others stay in the same order. A "word" is defined as a part of a string delimited either by spaces/tabs, or by the start/end of the string. Words will be separated by only one space in the output. If there's less than one argument, the program displays \n.

Example:

```
$>./rostring "abc   " | cat -e
abc$
$>
$>./rostring "Que la               lumiere soit et la lumiere fut"
la lumiere soit et la lumiere fut Que
$>
$>./rostring "         AkjhZ zLKIJz , 23y"
zLKIJz , 23y AkjhZ
$>
$>./rostring | cat -e
$
```

```c
#include <unistd.h>
int     ft_isblank(char c)
{
    if (c == ' ' || c == '\t')
        return (1);
    return (0);
}

void    rostring(char *s)
{
    int         i = 0;
    int         w_len = 0;
    while (s[i])
    {
        while (ft_isblank(s[i]))
                i++;
        if (s[i] && !ft_isblank(s[i]))
        {
                if (!w_len)
                       while (s[i] && !ft_isblank(s[i++]))
                              w_len++;
                else
                {
                       while (s[i] && !ft_isblank(s[i]) && write(1, &s[i++], 1));
                       write(1, " ", 1);
                }
        }
    }
    i = 0;
    while (ft_isblank(s[i]))
        i++;
    while (w_len--)
        write(1, &s[i++], 1);
}

int     main(int ac, char **av)
{
    if (ac > 1 && *av[1])
        rostring(av[1]);
    write(1, "\n", 1);
    return (0);
}
```

63.-Brainfuck

texto extraído de: ir a github

.- brainfuck

Expected Files: *.c, *.h

Allowed functions: write, malloc, free

Write a Brainfuck interpreter program. The source code will be given as first parameter. The code will always be valid, with no more than 4096 operations. Brainfuck is a minimalist language. It consists of an array of bytes (in our case, let's say 2048 bytes) initialized to zero, and a pointer to its first byte. Every operator consists of a single character :

- '>' increment the pointer ;
- '<' decrement the pointer ;
- '+' increment the pointed byte ;
- '-' decrement the pointed byte ;
- '.' print the pointed byte on standard output ;
- '[' go to the matching ']' if the pointed byte is 0 (while start) ;
- ']' go to the matching '[' if the pointed byte is not 0 (while end).

Any other character is a comment.

Examples:

```
$>./brainfuck "++++++++++[>+++++++>++++++++++>+++>+<<<<-]
>++.>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.------.--------.>+.>." | cat -e
Hello World!$
$>./brainfuck "+++++[>++++[>++++H>+++++i<<-]>>>++\n<<<<-]>>--------.>+++++.>." | cat -e
Hi$
$>./brainfuck | cat -e
$
```

```c
#include "unistd.h"
#include "stdlib.h"
char    *brakets(char *src, int way)
{
    int     i;
    i = 0;
    while (1)
    {
        if (*src == '[')
               i++;
        if (*src == ']')
               i--;
        if (i == 0)
               return (src);
        src = src + way;
    }
    return (NULL);
}
```

```c
void    brainfuck(char *s)
{
    char      *buff;
    buff = (char *)malloc(sizeof(*buff) * 4096);
    while (*s != '\0')
    {
        if (*s == '>')
               buff++;
        if (*s == '<')
               buff--;
        if (*s == '+')
               *buff = *buff + 1;
        if (*s == '-')
               *buff = *buff - 1;
        if (*s == '[' && *buff == 0)
               s = brakets(s, 1);
        if (*s == ']' && *buff != 0)
               s = brakets(s, -1);
        if (*s == '.')
               write(1, &*buff, 1);
        s++;
    }
}
```

```c
int      main(int argc, char **argv)
{
    if (argc == 2)
        brainfuck(argv[1]);
    else
        write(1, "\n", 1);
    return (0);
}
```

64.- print_memory

Expected Files: print_memory.c
Allowed functions: write

Write a function that takes (const void *addr, size_t size), and displays the memory as in the **Example**. Your function must be declared as follows:

```
void    print_memory(const void *addr, size_t size);
---------
$> cat main.c
void    print_memory(const void *addr, size_t size);

int    main(void)
{
   int    tab[10] = {0, 23, 150, 255,
              12, 16,  21, 42};

   print_memory(tab, sizeof(tab));
   return (0);
}
$> gcc -Wall -Wall -Werror main.c print_memory.c && ./a.out | cat -e
0000 0000 1700 0000 9600 0000 ff00 0000 ................$
0c00 0000 1000 0000 1500 0000 2a00 0000 ...........*...$
0000 0000 0000 0000                     ........$
```

```c
#include <unistd.h>
#include <stdio.h>
void    ft_putchar(char c)
{
    write(1, &c, 1);
}
void    ft_putasci(unsigned char c)
{
    if (c >= 32 && c < 127)
        ft_putchar(c);
    else
        ft_putchar('.');
}
void     ft_puthex(unsigned char c)
{
    char tab[16] = "0123456789abcdef";

    ft_putchar(tab[c / 16]);
    ft_putchar(tab[c % 16]);
}
void    print_memory(const void *addr, size_t size)
{
    unsigned char   *temp;
    size_t            i;
    temp = (unsigned char *)addr;
    i = 0;
    while (i < size)
    {
        ft_printline(temp, i, size);
        i = i + 16;
    }
}
```

```c
void    ft_printline(unsigned char *temp, size_t
start, size_t max)
{
    size_t    i;

    i = start;
    while (i < (start + 16) && i < max)
    {
        ft_puthex(temp[i]);
        if (i % 2 != 0)
               ft_putchar(' ');
        i++;
    }
    while (i < (start + 16)) .
    {
        ft_putchar(' ');
        ft_putchar(' ');
        if (i % 2 != 0)
               ft_putchar(' ');
        i++;
    }
    i = start;
    while (i < (start + 16) && i < max)
    {
        ft_putasci(temp[i]);
        i++;
    }
    ft_putchar('\n');
}
```

```c
int    main(void)
{
    int    tab[10] = {0, 23, 150, 255,12, 16,  21, 42};
    print_memory(tab, sizeof(tab));
    return (0);
}
```

65.- ft_itoa_base

Expected Files: ft_itoa_base.c
Allowed functions: malloc

Write a function that converts an integer value to a null-terminated string using the specified base and stores the result in a char array that you must allocate. The base is expressed as an integer, from 2 to 16. The characters comprising the base are the digits from 0 to 9, followed by uppercase letter from A to F. For **Example**, base 4 would be "0123" and base 16 "0123456789ABCDEF". If base is 10 and value is negative, the resulting string is preceded with a minus sign (-). With any other base, value is always considered unsigned. Your function must be declared as follows:

char   *ft_itoa_base(int value, int base);

```c
#include <stdio.h>
#include <stdlib.h>
int     get_length(int value, int base)
{
    int     ret;
    ret = 0;
    if (value == 0)
        return (1);
    if (value < 0 && base == 10)
        ++ret;
    while (value != 0)
    {
        value = value / base;
        ret++;
    }
    return (ret);
}
```

```c
char    *ft_itoa_base(int value, int base)
{
    int neg;
    char *num;
    int     len;
    long  value_cpy;
    char    buff[16] = "0123456789ABCDEF";
        neg = 0;
    len = get_length(value, base);
    num = (char *)malloc(sizeof(*num) * (len));
    if (!num)
        return (NULL);
    num[len] = '\0';
    value_cpy = value;
    if (value_cpy < 0)
    {
        if (base == 10)
               neg = 1;
        value_cpy = value_cpy * -1;
    }
    while (--len)
    {
        num[len] = buff[value_cpy % base];
        value_cpy = value_cpy / base;
    }
    if (neg == 1)
        num[0] = '-';
    else
        num[len] = buff[value_cpy % base];
    return (num);
}
```

```c
int     main(void)
{
    printf("RESULT:\n%s", ft_itoa_base(557736892, 15));
    return(0);
}
```

66.- brackets

Expected Files: *.c *.h
Allowed functions: write

Write a program that takes an undefined number of strings in arguments. For each argument, the program prints on the standard output "OK" followed by a newline if the expression is correctly bracketed, otherwise it prints "Error" followed by a newline. Symbols considered as 'brackets' are brackets '(' and ')', square brackets '[' and ']'and braces '{' and '}'. Every other symbols are simply ignored. An opening bracket must always be closed by the good closing bracket in the correct order. A string which not contains any bracket is considered as a correctly bracketed string. If there is no arguments, the program must print only a newline.

Examples :

```
$> ./brackets '(johndoe)' | cat -e
OK$
$> ./brackets '([)]' | cat -e
Error$
$> ./brackets '' '{[(0 + 0)(1 + 1)](3*(-1)){()}}' | cat -e
OK$
OK$
$> ./brackets | cat -e
$
$>
/*
** Many thanks to Anselme for his original idea :
** https://github.com/grumbach/misc/blob/master/brackets/brackets.c
*/
```

```
#include <unistd.h>
int     braclose(char *str, char c, int i, int b)
{
    while (b && *(++str) && (i++))
        if (*str == c || *str == c + c % 2 + 1)
               *str == c ? ++b : --b;
    return (i);
}
int     brackets(char *str, char c)
{
    if (*str == c)
        return (1);
    else if (!*str || *str == ')' || *str == '}' || *str == ']')
        return (0);
    else if (*str == '(' || *str == '{' || *str == '[')
        return (brackets(str + 1, *str + *str % 2 + 1)
               * brackets(str + braclose(str, *str, 1, 1), c));
    else
        return (brackets(str + 1, c));
}

int     main(int ac, char **av)
{
    int     i;

    i = 0;
    if (ac > 1)
        while (++i < ac)
               brackets(av[i], 0) ? write(1, "OK\n", 3) : write(1, "Error\n", 6);
    else
        write(1, "\n", 1);
    return (0);
}
```

67.- rpn_calc

Expected Files: *.c, *.h
Allowed functions: atoi, printf, write, malloc, free

Write a program that takes a string which contains an equation written in Reverse Polish notation (RPN) as its first argument, evaluates the equation, and prints the result on the standard output followed by a newline.
Reverse Polish Notation is a mathematical notation in which every operator follows all of its operands. In RPN, every operator encountered evaluates the previous 2 operands, and the result of this operation then becomes the first of the two operands for the subsequent operator.
Operands and operators must be spaced by at least one space.
You must implement the following operators : "+", "-", "*", "/", and "%".
If the string isn't valid or there isn't exactly one argument, you must print "Error" on the standard output followed by a newline.
All the given operands must fit in a "int".
Examples of formulas converted in RPN:

```
3 + 4                 >>        3 4 +
((1 * 2) * 3) - 4     >>        1 2 * 3 * 4 -  ou  3 1 2 * * 4 -
50 * (5 - (10 / 9))   >>        5 10 9 / - 50 *
```

Here's how to evaluate a formula in RPN:

1 2 * 3 * 4 -
2 3 * 4 -
6 4 -
2
Or:
3 1 2 * * 4 -
3 2 * 4 -
6 4 -
2

Examples:

```
$> ./rpn_calc "1 2 * 3 * 4 +" | cat -e
10$
$> ./rpn_calc "1 2 3 4 +" | cat -e
Error$
$> ./rpn_calc |cat -e
Error$
```

```c
#include "rpn_calc.h"

int     check_input(char *s)
{
    int     num_c;
    int     op_c;

    num_c = 0;
    op_c = 0;
    while (*s)
    {
        if (!(is_op(*s) || is_digit(*s) ||
is_space(*s)))
                return (0);
        if (is_op(*s))
        {
                if (num_c && (*(s - 1)) &&
!is_space(*(s - 1)))
                              return (0);
                op_c++;
                if ((*s == '-' || *s == '+') && (*(s
+ 1)) &&
                              is_digit(*(s + 1)))
                       op_c--;
        }
        else if (is_digit(*s))
        {
                if (!num_c || (*(s - 1) &&
!is_digit(*(s - 1))))
                       num_c++;
        }
        if (is_space(*s) && num_c <= op_c)
                return (0);
        ++s;
    }
    return (num_c - op_c == 1 ? 1 : 0);
}
```

```c
#include "rpn_calc.h"

int     is_op(int c)
{
    return (c == '+' ||
            c == '-' ||
            c == '*' ||
            c == '/' ||
            c == '%');
}

int     is_digit(int c)
{
    return ('0' <= c && c <= '9');
}

int     is_space(int c)
{
    return (c == 32);
}
void     push(t_s **stack, int i)
{
    t_s     *link;

    if (!(link = (t_s *)malloc(sizeof(t_s))))
        return ;
    link->i = i;
    if (*stack)
    {
        link->next = *stack;
        *stack = link;
    }
    else
    {
        link->next = *stack;
        stack = &link;
    }
```

```c
#include "rpn_calc.h"

int     main(int ac, char **av)
{
    if (ac == 2 && check_input(av[1]))
        rpn_calc(av[1]);
    else
        printf("Error\n");
    return (0);
}
```

```c
}

int      pop(t_s **stack)
{
    int     num;
    t_s *tmp;

    num = (*stack)->i;
    tmp = (*stack);
    *stack = (*stack)->next;
    free(tmp);
    return (num);
}
```

```c
#include "rpn_calc.h"


void    rpn_calc(char *s)
{
    t_s    **stack;
    int    num1;
    int    num2;

    if (!(stack = (t_s **)malloc(sizeof(t_s*))))
        return ;
    while (*s)
    {
        while (*s && is_space(*s))
               s++;
        if (*s && is_digit(*s))
        {
               push(stack, atoi(s));
               while (*s && is_digit(*s))
                       s++;
        }
        else if (*s && is_op(*s))
        {
               if (*(s + 1) && is_digit(*(s + 1)))
               {
                       push(stack, atoi(s));
                       s++;
                       while (is_digit(*s))
                              s++;
               }
               else {
                       num1 = pop(stack);
                       num2 = pop(stack);
                       if (num2 == 0 && (*s == '/'
|| *s == '%'))
                       {
                              printf("Error\n");
                              return ;
                       }
                       push(stack, do_op(num1, num2,
*s));

                       s++;
               }
        }
    }
    printf("%i\n", (*stack)->i);

}
```

```c
int    do_op(int i, int j, char c)
{
    if (c == '+')
        return (i + j);
    else if (c == '-')
        return (i - j);
    else if (c == '*')
        return (i * j);
    else if (c == '/')
        return (i / j);
    else if (c == '%')
        return (i % j);
    return (0);
}
```

68.- options

Expected Files: *.c *.h
Allowed functions: write

Write a program that takes an undefined number of arguments which could be considered as options and writes on standard output a representation of those options as groups of bytes followed by a newline. An option is an argument that begins by a '-' and have multiple characters which could be : abcdefghijklmnopqrstuvwxyz. Launch the program without arguments or with the '-h' flag activated must print an usage on the standard output, as shown in the following **Example**s. A wrong option must print "Invalid Option" followd by a newline.
All options are stocked in a single int and each options represents a bit of that int, and should be stocked like this :
00000000 00000000 00000000 00000000
******zy xwvutsrq ponmlkji hgfedcba

Examples :

```
$>./options
options: abcdefghijklmnopqrstuvwxyz
$>./options -abc -ijk
00000000 00000000 00000111 00000111
$>./options -z
00000010 00000000 00000000 00000000
$>./options -abc -hijk
options: abcdefghijklmnopqrstuvwxyz
$>./options -%
Invalid Option
```

```c
#include <unistd.h>
int main(int ac, char **av)
{
    int i = 1;
    int  t[32] = {0};
    int j ;
    if(ac == 1)
    {
         write(1,"options: abcdefghijklmnopqrstuvwxyz\n",36);
         return 0;
    }
    i = 1;
    while (i < ac)
    {
         j = 1;
         if(av[i][0] == '-')
         {
                 while(av[i][j] && av[i][j] >= 'a'  && av[i][j] <= 'z')
                 {
                         if(av[i][j] == 'h')
                         {
                                 write(1,"options: abcdefghijklmnopqrstuvwxyz\n",36);
                                 return 0;
                         }
                         t['z' - av[i][j] + 6] = 1;
                         j++;
                 }
                 if (av[i][j])
                 {
                         write(1,"Invalid Option\n",15);
                         return 0;
                 }
                 j++;
         }
         i++;
    }
    i = 0;
         while (i < 32)
         {
         t[i] = '0' + t[i];
         write(1,&t[i++],1);
                 if(i == 32)
                         write(1,"\n",1);
                 else if(i % 8 == 0)
                         write(1," ",1);
         }

    return 0;
}
```

Fin

Indice