```
// ce programme (composé d'une fonction main)
// affiche les arguments reçus en ligne de commande
// (un par ligne et triés par ordre ASCII)
// compilé avec gcc -Wall -Wextra -Werror ft_print_program_name.c
// puis exécuté avec ./a.out test3 test1 test2,
// il doit donc afficher test1 test2 test3 à l'écran
// (mais exécuté avec ./a.out seulement,
// ne doit rien afficher)
// tous les arguments doivent être affichés, sauf argv[0]

// pour pouvoir utiliser la fonction write
#include <unistd.h>

// pour comparer les valeurs ASCII des deux chaines de caractères
// passées en paramètre à cette fonction (str1 et str2)
int     ft_compare_argv(char *str1, char *str2);

// pour échanger la chaine de caractères argv[i]
// avec la chaine de caractères argv[j]
// argv étant un tableau de chaines de caractères
// (un pointeur sur des pointeurs sur le premier caractère
// d'une chaine)
void    ft_swap_argv(char **argv, int i, int j);

// pour écrire une chaine de caractères à l'écran
// (voir ex02)
void    ft_write_argv(char *str);

// voir ex00
int     main(int argc, char **argv)
{
        // on déclare un entier i,
        // compteur qui sera initialisé à 1
        // (index du premier argument passé
        // en ligne de commande)
        // il permettra de parcourir les arguments
        // passés en ligne de commande
        // jusqu'à l'avant-dernier
```

```
int     i;

// on déclare un entier j,
// compteur qui sera initialisé, dans chaque boucle
// parcourant les arguments du premier à l'avant-dernier
// à i + 1 (pour parcourir les arguments suivants jusqu'au dernier)
// pour la comparaison des deux chaines
// ainsi, le premier argument sera comparé au deuxième,
// puis au troisième, etc
// puis le deuxième argument sera comparé au suivant, le troisième
int     j;

// on initialise i à 1
// (index du premier argument passé
// en ligne de commande)
i = 1;

// tant que i est inférieur à argc - 1
// EXPLICATIONS :
// argc est le nombre d'arguments passés
// en paramètres + 1 (pour le nom du programme)
// EXEMPLE POUR 3 ELEMENTS PASSÉS EN LIGNE DE COMMANDE
// argc = 3 + 1 = 4
// DONC argc - 1 = 3
// on a donc while (i < 3)
// OR index du 3ème élément : 3
// DONC index de l'avant-dernier élément : 2
// avec while (i < 3) on s'arrêtera bien
// à l'avant-dernier élément

while (i < argc - 1)
{
        // on initialise j à i + 1
        // (pour comparer l'élément en cours aux éléments suivants)
        j = i + 1;

        // tant que j est inférieur à argc
        // argc = 4
```

```
// on a donc while (j < 4)
// OR index du 3ème élément : 3
// avec while (j < 4) on s'arrêtera bien
// au dernier élément
while (j < argc)
{
        // si le résultat de la comparaison entre argv[i]
        // (l'élément en cours)
        // et argv[j] (l'élément suivant en cours)
        // est supérieur à 0

        // (autrement dit, si la valeur ASCII de
        // argv[i] est supérieure à celle du
        // de argv[j] (voir plus bas))
        // (car ft_compare_argv(argv[i], argv[j])
        // retourne la différence de la valeur
        // ASCII des premiers caractères différents
        // d'argv[i] est d'argv[j]
        // (*argv[i] - *argv[j])

        // donc si (*argv[i] > *argv[j])
        // la valeur de retour sera positive
        // il faut échanger les chaines
        // argv[i] et argv[j]
        if (ft_compare_argv(argv[i], argv[j]) > 0)
                // on les échange (voir plus bas)
                // (car on veut trier
                // les éléments par ordre ASCII)
                ft_swap_argv(argv, i, j);
        // on incrémente j
        // (pour ensuite pouvoir comparer
        // argv[i] à l'élément suivant
        j++;
}
// on incrémente i
// (pour passer à l'élément de base suivant
// une fois que celui-ci a été comparé à tous les
// éléments suivants)
```

```
                i++;
        }

        // une fois que tous les éléments ont été triés par ordre ASCII
        // REMARQUE IMPORTANTE :
        // argv[0] contiendra toujours le nom du programme !

        // on réinitialise i à 1
        // (index du premier argument par ordre ASCII)
        // pour pouvoir les écrire à l'écran
        // (voir ex01)
        i = 1;
        while (i < argc)
        {
                ft_write_argv(argv[i]);
                write(1, "\n", 1);
                i++;
        }
        return (0);
}

// pour comparer les valeurs ASCII des chaines de caractères
// str1 et str2
int     ft_compare_argv(char *str1, char *str2)
{
        // tant que l'on a pas atteint la fin de la chaine
        // de caractères pointée par str1
        // ET (NI) celle pointée par str2
        // (TANT QUE LES DEUX CHAINES NE SONT PAS TERMINÉES)
        // ATTENTION : while != condition 1 && != condition 2
        // signifie tant que ni la condition 1,
        // ni la condition 2 n'est remplie

        // la boucle s'arrêtera donc dès qu'une des deux chaines
        // aura atteint la fin (ou les deux en même temps)

        // EXEMPLES :
        // str1 s'arrête avant str2 :
```

```
// A LA FIN DE str1 :
// *str1 == '\0' ? OUI =>
// *str1 != '\0' ? NON
// *str2 == '\0' ? NON =>
// *str2 != '\0' ? OUI
// NON && OUI => NON
// STOP

// str2 s'arrête avant str1 :
// A LA FIN DE str2 :
// *str1 == '\0' ? NON =>
// *str1 != '\0' ? OUI
// *str2 == '\0' ? OUI =>
// *str2 != '\0' ? NON

// OUI && NON => NON
// STOP

// str1 s'arrête en même temps que str2 :
// A LA FIN DE str2 :
// *str1 == '\0' ? OUI =>
// *str1 != '\0' ? NON
// *str2 == '\0' ? OUI =>
// *str2 != '\0' ? NON

// NON && NON => NON
// STOP

while (*str1 != '\0' && *str2 != '\0')
{
        // si les caractères pointés par str1 et str2
        // (le premier caractère de chacune des chaines
        // lors de la première entrée dans la boucle)
        // sont différents
        // (si la valeur ASCII décimale de *str1
        // est différente de celle de *str2)
        if (*str1 != *str2)
                // on retourne la différence entre
```

```
                // ces 2 valeurs ASCII
                // (le résultat sera donc positif
                // si str1 > str2
                // et négatif si str1 < str2)
                return (*str1 - *str2);
        // on avance d'un caractère dans les chaines
        // str1 et str2 en même temps
        // on incrémente str1
        // pour avancer au caractère suivant
        // de la chaine
        str1++;
        // on incrémente aussi str2
        // pour avancer au caractère suivant
        // de la chaine
        str2++;
}
// 3 cas possibles pour arriver à ce return :

// 1) tous les caractères ont été comparés,
// les chaines étaient de la même longueur et
// tous les caractères étaient égaux
// (CAS NON ET NON)
// on retourne *str1 - *str2
// mais comme *str1 est égal à *str2
// 0 est retourné
// (et comme la valeur de retour n'est pas
// supérieure à 0, les deux chaines
// ne seront pas échangées (voir plus haut))

// 2) str1 s'est terminée avant str2 :
// (CAS NON ET OUI)
// *str1 est égal à '\0'
// or la valeur décimale de '\0' est 0
// donc 0 - *str2 sera retourné
// donc forcément une valeur négative
// (et comme la valeur de retour n'est pas
// supérieure à 0, les deux chaines
// ne seront pas échangées (voir plus haut))
```

```
        // 3) str2 s'est terminée avant str1 :
        // (CAS OUI ET NON)
        // *str2 est égal à '\0'
        // or la valeur décimale de '\0' est 0
        // donc *str1 - 0 sera retourné
        // donc forcément une valeur poisitive
        // (et comme la valeur de retour est
        // supérieure à 0, les deux chaines
        // seront échangées (voir plus haut))

        return (*str1 - *str2);
}

// pour échanger les chaines de caractères
// pointées par argv[i] et argv[j]
// i et j étant les indices des deux chaines
// de caractères à échanger,
// et argv un tableau de chaines de caractères
void    ft_swap_argv(char **argv, int i, int j)
{
        // pointeur temporaire pour stocker
        // la valeur d'argv[i] à échanger
        // avec celle d'argv[j]
        char    *temp;

        // on stocke l'adresse de la
        // chaine de caractères de gauche, argv[i],
        // dans le pointeur temporaire temp
        temp = argv[i];

        // on remplace la valeur
        // du pointeur argv[i] par celle du pointeur argv[j]
        argv[i] = argv[j];

        // on remplace la valeur
        // du pointeur argv[j] par celle du pointeur argv[i]
        // (auparavent stockée dans le pointeur temporaire)
```

```
        argv[j] = temp;
}

// pour écrire la chaine de caractères à l'écran
// voir ex02
void    ft_write_argv(char *str)
{
        // on parcourt la chaine de caractères
        while (*str != '\0')
                // on écrit le caractère en cours (pointé par str)
                // puis on incrémente str
                write(1, str++, 1);
}
```