```
// cette fonction reproduit à l'identique
// le fonctionnement de la fonction strdup
// ASTUCE : man strdup

// duplicate a string (string duplicate)
// cette fonction prend en paramètre un pointeur vers une chaine de caractères src
// et retourne un pointeur vers une chaine de caractères de destination dest,
// cette chaine étant une copie de src
// la mémoire pour la nouvelle chaine de caractères est obtenue avec malloc
// IMPORTANT : il faut utiliser le header <stdlib.h> pour pouvoir utiliser malloc

// NOTE :
// comme cette mémoire est allouée dynamiquement
// elle doit être libérée avec free
// (avec free(dest); par exemple)
// une fois que l'on n'a plus besoin de cette variable
// (une fois affichée par exemple)
// DANS LE MAIN
// IMPORTANT : il faut utiliser le header <stdlib.h> pour pouvoir utiliser free

// ATTENTION : après avoir libéré la mémoire allouée à dest
// celle-ci est rendue disponible pour d'autres allocations
// (cette mémoire est marquée comme libre par le gestionnaire de mémoire du système)
// mais dest pointe toujours vers la même adresse mémoire, même si celle-ci n'est plus
// considérée comme valide !
// le pointeur dest est devenu ce que l'on appelle un "dangling pointer" (pointeur qui pend)
// ACCEDER A CETTE ADRESSE PEUT CAUSER DES ERREURS GRAVES !
// il est donc important de réinitialiser le pointeur à NULL juste après l'appel à free
// pour éviter d'accéder accidentellement à une mémoire libérée

// si l'opération est un succès, la fonction retourne un pointeur vers la chaine
// de caractères dupliquée (un pointeur sur le premier caractère de la chaine copiée)

// si l'opération a échoué car la mémoire disponible
// pour allouer une copie de src est insuffisante,
// (si malloc a échoué)
// la variable globale errno prend la valeur ENOMEM
// (avec errno = ENOMEM;)
```

```
// pour cela, il faut inclure le header <errno.h>
// (header de gestion des erreurs)
// puis NULL est retourné


// ATTENTION :
// dans le main, il est important de vérifier que dest ne soit pas égal à NULL
// avant de tenter d'accéder à sa valeur (pour l'afficher par exemple)
// on pourra aussi vérifier que l'erreur est bien
// un problème de mémoire insuffisante
// en vérifiant que errno soit égal à ENOMEM
// (l'erreur est inconnue dans ce cas)
// IMPORTANT : ne pas oublier d'inclure ici aussi le header <errno.h>
// pour pouvoir accéder à la variable globale errno

// pour pouvoir utiliser malloc et la constante spéciale NULL
#include <stdlib.h>

// pour pouvoir définir errno
#include <errno.h>

// pour copier une chaine de caractères
// prend en paramètre src, un pointeur vers la chaine de caractères source
// et retourne un pointeur vers la chaine de caractères de destination,
// une fois que celle-ci a reçu la copie de src
char    *ft_strdup(char *src)
{
        // pour parcourir les chaines de caractères src et dest
        int     i;

        // longueur (nombre de caractères) de la chaine src
        // (sans compter le caractère de fin de chaine '\0')
        int     length;

        // pointeur vers la chaine de destination
        char    *dest;

        // on initialise i à 0
```

```
// (indice du premier caractère de src)
i = 0;

// tant que l'on a pas encore atteint la fin
// de la chaine de caractères
while (src[i] != '\0')
        // on incrémente i
        // i correspondra ainsi au nombre de caractères de src
        i++;

// on affecte donc i à length
length = i;

// on alloue dynamiquement la mémoire nécessaire
// pour accueillir la chaine de caractères dest
// (copie de src)
// avec malloc (memory allocation)
// en indiquant le nombre d'octets à allouer :
// *src signifie : le contenu pointé par src
// src est un pointeur sur le premier caractère de la chaine source
// (c'est un char *)
// donc le contenu pointé par src est un caractère
// sizeof(*src) signifie donc la taille en octets
// du caractère pointé par src
// donc la taille en octet d'un caractère
// (un caractère prend en général 1 octet en mémoire)
// pour obtenir la taille en octet de la chaine dest à créer
// il faut multiplier ceci par le nombre de caractères à copier
// (ce qui correspond à length) + 1 (car il faut aussi allouer de la mémoire
// pour le caractère de fin de chaine '\0')
// ATTENTION : malloc retourne un pointeunr de type *void
// (pointeur générique)
// il faut convertir ce pointeur en un pointeur de type char *
// (pointeur vers un caractère)
// car dest, qui accueille cette mémoire, est un pointeur
// vers une chaine de caractère
// REMARQUE :
// la conversion peut se faire de manière implicite
```

```
// car dest est déjà de type char *
// (char *) est ce qu'on appelle une conversion explicite
dest = (char *)malloc((length + 1) * sizeof(*src));

// si dest a pu être créé
// (si assez de mémoire est disponible pour que malloc
// alloue ce nombre d'octets en mémoire
if (dest)
{
        // on initialise i à 0
        // (indice du premier caractère)
        // REMARQUE :
        // ici, i servira à parcourir à la fois les chaines dest et src
        // (en avançant d'un caractère à la fois sur les deux chaines
        // en même temps)
        i = 0;

        // tant que i est inférieur à length
        // (car length correspond au nombre de caractères à copier
        // et que l'indice i commence à 0
        // EXEMPLE avec length = 5 :
        // on veut copier 5 caractères
        // en commençant par l'indice 0
        // donc aux indices 0, 1, 2, 3 et 4
        // on s'arrête donc après l'indice 4
        // 4 < 5
        // donc tant que i < length
        while (i < length)
        {
                // on copie le caractère en cours de src dans dest
                // (le 1er caractère de src au premier emplacement de dest,
                // puis le 2eme au 2eme emplacement, etc)
                dest[i] = src[i];

                // on incrémente i
                // pour faire avancer d'un caractère src et dest à la fois
                i++;
        }
```

```
                // ATTENTION :
                // une fois le dernier caractère copié avec dest[i] = src[i];
                // i aura été incrémenté de 1 (par i++)
                // dest[i] pointera donc juste après le dernier caractère copié
                // on en profite pour inclure le caractère de fin de chaine
                // à cet emplacement
                dest[i] = '\0';

                // on retourne dest
                return (dest);
        }

        // si on est pas déjà sorti de la fonction
        // (correspond à (if !dest))
        // (si dest n'a pas pu être créé car
        // il n'y a pas assez de mémoire est disponible pour que malloc
        // alloue le nombre d'octets suffisant en mémoire)
        // on définit la variable globale errno à ENOMEM
        // (Error NO MEMory)
        // cette variable, avec l'aide du header <errno.h>
        // pourra être retrouvée dans le programme appellant ft_strdup
        // on pourra ainsi gérer le cas où cette erreur apparait
        // (en affichant un message d'erreur par exemple)
        errno = ENOMEM;

        // on retourne aussi NULL
        // (NULL est une constante spéciale définie dans <stdlib.h>
        // et représente une adresse mémoire invalide (de valeur 0)
        // utiliser NULL pour un pointeur signifie que ce pointeur
        // ne pointe vers aucune adresse mémoire valide
        return (NULL);
}

#include "ft_strcpy.h"
#include "ft_putstr.h"
#include <unistd.h>
#include "ft_strdup.h"
#include <stdlib.h>
```

```c
#include <errno.h>

int     main(void)
{
        char    src[14];
        char    *dest;

        ft_strcpy(src, "Hello World !");
        ft_putstr(src);
        write(1, "\n", 1);
        dest = ft_strdup(src);

        // si dest a été défini à NULL
        // (s'il ne pointe pas vers une mémoire valide)
        if (dest == NULL)
        {
                // on écrit un message d'erreur
                write(1, "Error duplicating string :\n", 28);

                // si la variable globale errno a été initialisée à ENOMEM
                if (errno == ENOMEM)
                        // on indique que l'erreur provient d'une mémoire insuffisante
                        // pour allouer la copie de la chaine en mémoire
                        write(1,
                                "Insufficient memory available to allocate duplicate string.\n",
                                60);
                // sinon
                else
                        // on indique que l'erreur est inconnue
                        write(1, "An unknow error occured.\n", 25);

                // on retourne 1 pour indiquer que le programme a rencontré une erreur
                return (1);
        }

        // si strdup n'a pas retourné NULL
        // on affiche la chaine de caractères pointée par dest
        ft_putstr(dest);
```

```
        // on libère la mémoire allouée dynamiquement à dest dans ft_strdup
        // car on a plus besoin de dest
        // sinon, cette mémoire reste réservée pour le programme
        // et elle reste occupée alors qu'elle n'est plus nécessaire
        // cela conduit à une fuite de mémoire
        // à force, ces fuites peuvent s'accumuler et épuiser la mémoire disponible
        // ce qui peut ralentir le système ou faire planter le programme
        free(dest);

        // on réinitialise le pointeur à NULL
        // (pour éviter d'accéder accidentellement à une mémoire libérée)
        dest = NULL;
        write(1, "\n", 1);
        // on retourne 0 pour indiquer que le programme s'est exécuté correctement, sans erreur
        return (0);
}
```