```
// cette fonction affiche un nombre dans une base donnée
// elle prend en paramètre nbr, le nombre à convertir, en int
// et la base dans laquelle convertir ce nombre, base, en chaine
// de caractères (char *)

// la base doit contenir tous les symboles utilisables
// pour afficher le nombre
// ex : 0123456789 est une base décimale (10 caractères)
// formée des caractères 0 1 2 3 4 5 6 7 8 et 9
// 01 est une base binaire formée des caractères 0 et 1
// 012345678ABCDEF est une base hexadécimale
// poneyvif est une base octale

// la fonction doit gérer les nombres négatifs
// en écrivant "-" puis en inversant le signe du nombre
// pour le convertir dans la base donnée puis l'afficher

// * la fonction ne doit rien afficher si :
// - la base est vide ou de taille 1
// - la base contient deux fois le même caractère
// - la base contient les caractères + ou -

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

// pour calculer le nombre de caractères
// dans la chaine de caractères base
int     ft_base_len(char *base);

// fonction de gestion des erreurs (pour ne rien retourner
// si une erreur est rencontrée, voir * plus haut)
int     ft_check_base_error(char *base, long int base_len);

// fonction récursive pour extraire le dernier NOMBRE de nbr
// pouvant être converti dans la base donnée
// ATTENTION : cela sera le dernier chiffre affiché après conversion !
// pas forcément le dernier chiffre de nbr
```

```
// CETTE FONCTION PERMET DE DECOMPOSER LE NOMBRE nbr EN CHIFFRES
// POUVANT ETRE CONVERTIS DANS LA BASE DONNEE
// exemple : pour une base hexadécimale (base 16)
// un nombre entre 0 et 15 sera extrait puis envoyé
// à ft_convert_nbr_to_base pour sa conversion
// (voir plus bas)

void    ft_extract_last_nbr(long int n, long int base_len, char *base);

// fonction pour convertir un nombre dans la base donnée
void    ft_convert_nbr_to_base(long int n, char *base);

// fonction principale (appelée par main) pour convertir le nombre nbr dans la base base)
void    ft_putnbr_base(int nbr, char *base)
{
        // base_length stockera la longueur de la base base
        // elle sera utilisée par la fonction de gestion d'erreurs,
        // celle d'extraction du dernier nombre pouvant être converti dans la base donnée
        // et celle de conversion de ce nombre
        // CELA CORRESPONDRA A LA BASE NUMERIQUE DE base
        // ex : 10 pour une base décimale, 2 pour une base binaire,
        // 16 pour une base hexadécimale, 8 pour une base octale, etc
        long int        base_length;

        // long_nbr stockera le nombre à convertir (nbr) sous la forme d'un long int
        // cette conversion se fera plus bas
        // celle-ci permet de s'assurer que les opérations mathématiques restent dans
        // les limites du type de données utilisées
        // (permet notamment de convertir -2 147 483 648, le plus petit nombre représentable
        // en int, dans la base donnée, sans qu'il n'y ait d'erreur de débordement
        // lors de sa manipulation (la fonction prend sa valeur absolue pour le convertir,
        // or elle n'est pas représentable en int)
        long int        long_nbr;

        // après exécution de ft_check_base_error, vaudra 0 si aucune erreur n'a été
        // rencontrée, 1 sinon
        int     error;
```

```
	// on appelle la fonction ft_base_len pour calculer la longueur de la chaine de caractères base
	// (le nombre de caractères dans la base)
	base_length = ft_base_len(base);

	// on appelle la fonction ft_check_base_error pour vérifier si la base est valide
	// (voir *)
	error = ft_check_base_error(base, base_length);

	// si une erreur est détectée (si 1 a été retourné par ft_check_base_error)
	if (error)
		// la fonction retourne immédiatement, sans effectuer d'autre opérations
		return ;
	// on convertit nbr (le nombre à convertir) en long int puis on le stocke dans long_nbr
	// pour éviter des erreurs lors de sa manipulation (voir plus haut)
	long_nbr = (long int)nbr;

	// gestion des nombres négatifs :
	// si le nombre fourni est inférieur à 0
	if (long_nbr < 0)
	{
		// on écrit "-" en utilisant write pour indiquer que le nombre est négatif
		write(1, "-", 1);
		// puis on rend long_nbr positif
		// pour simplifier la conversion
		long_nbr = -long_nbr;
	}
	// on appelle la fonction récursive ft_extract_last_nbr pour décomposer
	// le nombre long_nbr en chiffres dans la base donnée
	// et ensuite les afficher après leur conversion (par ft_convert_nbr_to_base)
	ft_extract_last_nbr(long_nbr, base_length, base);
}

// pour calculer le nombre de caractères dans la base
int	ft_base_len(char *base)
{
	// pour parcourir la chaine de caractères base
	// ptr stockera une copie du pointeur base
	// (pointeur vers l'adresse du premier caractère de base)
```

```
        // puis il avancera jusqu'à rencontrer un caractère de fin de chaine ('\0')
        // (incrémentation pour se déplacer vers le caractère suivant dans la chaine)
        char    *ptr;

        // on initialise le pointeur ptr sur le début de la chaine base
        // (représenté par base)
        // (base est un pointeur de type *char pointant sur le premier caractère
        // de la chaine de caractères base)
        ptr = base;

        // tant que le caractère pointé par ptr n'est pas le caractère nul
        // (tant que la fin de la chaine de caractères n'a pas été atteinte)
        while (*ptr != '\0')
                // on incrémente ptr
                // (déplacement vers le caractère suivant dans la chaine)
                ptr++;
        // la boucle s'arrête lorsque ptr pointe sur le caractère nul

        // ptr - base permet de calculer la distance (en nombre de caractères)
        // entre le pointeur ptr (qui est maintenant au début de la chaine)
        // et le pointeur base (qui est au début de la chaine)
        // elle correspond à la longueur de la chaine de caractères base
        // (à son nombre de caractères)
        return (ptr - base);
}

// cette fonction retourne 1 s'il y a une erreur dans la base fournie :
// - si la base est vide ou de taille 1
// - si la base contient deux fois le même caractère
// - si la base contient les caractères + ou -
// elle retourne 0 si aucune erreur n'a été rencontrée
int     ft_check_base_error(char *base, long int base_len)
{
        // ptr1 et ptr2 sont deux pointeurs de type char
        // utilisés pour parcourir la chaine base
        // (voir plus bas)
        char    *ptr1;
        char    *ptr2;
```

```
// - si la base est vide ou de taille 1 :
// (si la longueur de la base est 0 ou 1)
if (base_len == 0 || base_len == 1)
        // on retourne 1
        return (1);

// - si la base contient deux fois le même caractère :
// on compare le premier caractère à tous les caractères suivants
// puis on passe au deuxième caractère, qu'on compare
// de nouveau à tous les suivants

// on initialise ptr1 pour qu'il pointe sur le début de la chaine base
// (il sera incrémenté ensuite pour parcourir la chaine base)
ptr1 = base;

// on parcourt chaque caractère jusqu'à la fin - 1 :
// tant que l'adresse ptr1 (l'adresse du caractère en cours)
// est inférieure à l'adresse base (l'adresse du début de base)
// incrémentée de base_len caractères (donc l'adresse de la fin de base) - 1
// DONC tant qu'on a pas atteint l'avant-dernier caractère de la chaine
while (ptr1 < base + base_len - 1)
{
        // on initialise ptr2 pour qu'il pointe sur le caractère
        // suivant le caractère en cours
        ptr2 = ptr1 + 1;

        // on parcourt chaque caractère jusqu'à la fin :
        // tant que l'adresse ptr2 (l'adresse du caractère en cours)
        // est inférieure à l'adresse base (l'adresse du début de base)
        // incrémentée de base_len caractères (donc l'adresse de la fin de base)
        // DONC tant qu'on a pas atteint la fin de la chaine
        while (ptr2 < base + base_len)
        {
                // si un caractère suivant ptr1 est identique
                // à celui-ci
                // (si le caractère ptr1 est trouvé en double dans
                // le reste de la chaine)
```

```
                        if (*ptr1 == *ptr2)
                                // on retourne 1
                                return (1);
                        // on passe ptr2 au caractère suivant pour la comparaison
                        // (dans la 2ème boucle)
                        ptr2++;
                }
                // on passe ptr1 au caractère suivant pour la comparaison
                // (dans la 1ère boucle)
                ptr1++;
        }

        // - si la base contient les caractères + ou - :
        // on réinitialise ptr1 pour qu'il pointe sur le début de la chaine base
        // (il sera aussi incrémenté ensuite pour parcourir la chaine base)
        ptr1 = base;

        // on parcourt chaque caractère jusqu'à la fin :
        // tant que l'adresse ptr1 (l'adresse du caractère en cours)
        // est inférieure à l'adresse base (l'adresse du début de base)
        // incrémentée de base_len caractères (donc l'adresse de la fin de base)
        // DONC tant qu'on a pas atteint la fin de la chaine
        while (ptr1 < base + base_len)
        {
                // si la valeur de la variable à l'adresse ptr1
                // (le caractère à l'adresse ptr1) est '+'
                // ou '-'
                if (*ptr1 == '+' || *ptr1 == '-')
                        // on retourne 1
                        return (1);
                // on incrémente ptr1 pour avancer au caractère suivant
                ptr1++;
        }

        // on retourne 0 si aucune erreur n'a été rencontrée
        return (0);
}
```

```
// cette fonction récursive est utilisée pour extraire et afficher
// les chiffres du nombre fourni dans la base donnée
// pour cela, on divise le nombre en ses composants individuels
// en utilisant la division et le modulo
// la fonction auxiliaire ft_convert_nbr_to_base convertira et affichera
// chaque chiffre

// elle prend en paramètre le nombre à convertir, la longueur de la base fournie
// (16 pour une base hexadécimale par exemple)
// et la chaine de caractères représentant les symboles de la base
// (ex : 123456789ABCDEF pour une base hexadécimale)

// ATTENTION : cette fonction n'utilise pas
// la chaine de caractères base directement
// mais la passe en paramètre à ft_convert_nbr_to_base pour la conversion
// des chiffres extraits

// EXPLICATIONS CONVERSION D'UN NOMBRE DANS UNE AUTRE BASE :
// dans le système décimal (base 10) :
// on prend l'exemple : 1234
// ce nombre est composé de :
// 1 * 1 000 (1 * 10^3)
// 2 * 100 (2 * 10^2)
// 3 * 10 (3 * 10^1)
// 4 * 1 (4 * 10^0)

// idem pour un nombre en base b
// (chacun des chiffres est multiplié par une puissance de b)

// dans le système hexadécimal (base 16) :
// on prend l'exemple : 291
// ce nombre est composé de :
// 1 * 256 (1 * 16^2) = 256
// 2 * 16 (2 * 16^1) = 32
// 3 * 1 (3 * 16^0) = 3
// car 256 + 32 + 3 = 291
// on a donc 291 = (1 * 16^2) + (2 * 16^1) + (3 * 16^0)
// donc 291 en base décimale correspond à 123 en base 16
```

```
// pour convertir un nombre dans une base b, il faut tout d'abord
// décomposer le nombre en termes de puissances de b
// on divise donc le nombre par b pour réduire le nombre
// en extrayant le chiffre le plus à droite
// (le chiffre des unités dans la nouvelle base)

// on utilise la récursivité pour traiter chaque chiffre
// du plus significatif au moins significatif
// n / base_len déplace le "point décimal" vers la gauche
// ce qui réduit la valeur de n et permet d'isoler les chiffres
// les plus significatifs
// on réduit ainsi récursivement n tant que n est supérieur ou égal
// à base_len
// cela permet de s'assurer que l'on traite tous les chiffres du nombre

// on utilise ensuite n % base_len pour obtenir le dernier chiffre
// dans la nouvelle base
// et le convertir ensuite en son caractère correspondant dans la base
// puis l'afficher

// exemple :
// n = 291
// base_len = 16
// 1ère itération : 291 >= 16
// 291 / 16 = 18
// on appelle donc ft_extract_last_nbr avec n = 18
// 2ème itération : 18 >= 16
// 18 / 16 = 1
// on appelle donc ft_extract_last_nbr avec n = 1
// 3ème itération : 1 < 16
// pas de nouvel appel récursif
// on appelle donc ft_convert_nbr_to_base avec n = 1 % 16
// 1 % 16 = 1
// conversion de 1 par ft_convert_nbr_to_base
// retour à la deuxième itération
// on appelle ft_convert_nbr_to_base avec n = 18 % 16
// 18 % 16 = 2
```

```
// conversion de 2 par ft_convert_nbr_to_base
// retour à la première itération
// on appelle ft_convert_nbr_to_base avec n = 291 % 16
// 291 % 16 = 3
// conversion de 3 par ft_convert_nbr_to_base

void    ft_extract_last_nbr(long int n, long int base_len, char *base)
{
        // condition de récursivité :
        // si le nombre n est supérieur ou égal à base_len
        // (si le nombre en cours est encore plus grand que la
        // base numérique utilisée (16 par exemple)),
        // donc tant que le nombre n'a pas encore été
        // totalement traité)
        if (n >= base_len)
                // on réduit n
                // par sa partie la plus significative
                // en le divisant par la base numérique
                // dans laquelle le convertir
                // et on appelle récursivement ft_extract_last_nbr
                // avec cette valeur
                ft_extract_last_nbr(n / base_len, base_len, base);
        // on extrait la partie la moins significative de n
        // et on l'envoie à ft_convert_nbr_to_base pour qu'il convertisse
        // et affiche ce dernier chiffre
        ft_convert_nbr_to_base(n % base_len, base);
}

// cette fonction convertit le chiffre n dans la base base

// le chiffre n est compris entre 0 et la base numérique
// (entre 0 et 16 (non compris)
// pour une base hexadécimale par exemple)
// base est la chaine de caractères représentant
// les symboles de la base de destination
// (exemple : 0123456789ABCDEF pour une base 16)

void    ft_convert_nbr_to_base(long int n, char *base)
```

```
{
        // stocke le caractère correspondant au chiffre
        // dans la base donnée
        char    c;

        // on accède au n-ième caractère de la chaine base
        // en faisant avancer base, l'adresse de début de la chaine
        // de caractères contenant les symboles de la base,
        // de n caractères
        // puis en assignant la valeur à cette adresse
        // (le caractère à cette adresse) à c

        // EXEMPLE :
        // la chaine base est :
        // 0123456789ABCDEF
        // base est donc l'adresse de 0
        // avec n = 15
        // base + n = adresse de 0 + 15
        // on avance de 15 caractères à partir de l'adresse de 0
        // le curseur se retrouve donc devant F (à l'étoile ci-dessous) :
        // 0123456789ABCDE*F
        // on accède à la valeur à cette adresse comme ceci :
        // *(base + n)
        // (l'opérateur * est utilisé pour accéder au contenu de la mémoire
        // à l'adresse base + n
        // cela équivaut à base[n]

        c = *(base + n);

        // on écrit le caractère c à l'écran
        write(1, &c, 1);
}

// MAIN :
// poneyvif est une base octale (8 caractères)

#include "ft_strcpy.h"
#include "ft_putnbr_base.h"
```

```
int     main(void)
{
        int     nbr;
        char    base[9];

        nbr = 16487424;
        ft_strcpy(base, "poneyvif");
        ft_putnbr_base(nbr, base);
        return (0);
}

// RESULTAT :
// 76712000
// fifonppp
```