```c
// cette fonction permet d'afficher un nombre affiché en paramètre
// elle est capable d'afficher la totalité des valeurs possibles dans une
// variable de type int
// elle utilise write, de la bibliothèque <unistd.h>, pour
// afficher le résultat à l'écran

// on inclus <unistd.h> pour pouvoir utiliser la fonction
// de la bibliothèque standard C write()
#include <unistd.h>

// cette fonction décompose le nombre en chiffres individuels
// et les traite de manière récursive
void    ft_write_numbers(int nb);

// cette fonction convertit un chiffre en son caractère ASCII
// correspondant et l'affiche
void    ft_write_number(int nb);

// cette fonction gère les cas particuliers
// (nombre -2147483648 et nombres négatifs)
// et appelle ft_write_numbers pour décomposer le nombre en chiffres
// c'est cette fonction qui devra être appelée par le main
// elle ne renvoie rien, et prend en paramètre le nombre à afficher
// à l'écran, nb
void    ft_putnbr(int nb)
{
        // CAS SPECIAL :
        // si nb est égal à -2147483648
        // (qui est le plus petit entier (de type int)
        // représentable en C, car les entiers sont
        // encodés sur 32 bits)
        // -2147483648 = 10000000 00000000 00000000 00000000
        // on l'écrit directement à l'écran
        // car elle ne peut pas être représentée
        // positivement dans la limite des entiers
        // (le plus grand entier (de type int)
        // représentable en C) est 2147483647
        // 01111111 11111111 11111111 11111111
```

```
// car (voir plus bas) les nombres négatifs
// sont convertis en leur valeur positive
// après avoir affiché "-" à l'écran
// par la fonction pour les afficher
// or ce n'est pas possible pour -2147483648
// il ne peut pas être converti dans sa valeur positive
// (il y aurait une erreur de débordement sinon)

// si nb est égal à -2147483648
if (nb == -2147483648)
{
        // on écrit -2147483648 directement à l'écran
        // REMARQUE : on indique 11 en 3ème paramètre de write
        // car on écrit 11 caractères à l'écran
        write(1, "-2147483648", 11);
        // on sort immédiatement de la fonction après avoir écrit
        // le nombre à l'écran
        // - la fonction a bien fait le travail qu'on voulait qu'elle fasse
        // (pas besoin d'opérations supplémentaire, le nombre a été affiché
        // à l'écran) : cela permet d'arrêter son exécution directement
        // et donc d'optimiser le temps d'exécution de la fonction
        // - on évite ainsi les erreurs liées au débordement d'entier
        // sans cet arrêt, le bloc :
        // if (nb < 0)
        // {
        //     write(1, "-", 1);
        //     nb = -nb;
        // }
        // serait exécuté, ce qui provoquerait une erreur de débordement d'entier
        // - on simplifie le traitement de ce nombre dans avoir á
        // modifier la logique pour tous les autres nombres

        // cela garantit stabilité et fiabilité du code
        return ;
}

// si nb est négatif (autre que -2147483648)
```

```
        // on écrit "-" à l'écran, puis on convertit nb
        // en sa valeur positive
        if (nb < 0)
        {
                write(1, "-", 1);
                nb = -nb;
        }
        // on appelle ensuite la fonction récursive ft_write_numbers()
        // en lui passant le nombre à afficher à l'écran :
        // elle traitera chaque chiffre du nombre pour l'afficher
        ft_write_numbers(nb);
}

// cette fonction est récursive
// elle décompose le nombre entier passé en argument (nb)
// en ses chiffres individuels
// pour les afficher un par un, dans l'ordre
// elle ne renvoie rien
void    ft_write_numbers(int nb)
{
        // si nb est supérieur ou égal à 10
        if (nb >= 10)
                // la fonction ft_write_numbers est appelée
                // récursivement avec nb / 10
                // cette opération divise nb en supprimant le chiffre des unités
                // ce qui ramène la valeur de nb à tous ses chiffres sauf le dernier

                // le dernier chiffre du nombre est ainsi éliminé

                // EXEMPLE :
                // 1) (appel par ft_putnbr) ft_write_numbers(1234)
                // 2) ft_write_numbers(123) (1234 / 10 => 123)
                // 3) ft_write_numbers(12) (123 / 10 => 12)
                // 4) ft_write_numbers(1) (12 / 10 => 1)
                // fin des appels récursifs, car on ne rentre plus dans la condition
                // if (nb >= 10) car nb n'est plus supérieur ou égal à 10

                ft_write_numbers(nb / 10);
```

```
        // on appelle ft_write_number avec nb % 10
        // ce qui donne le dernier chiffre de nb
        // la fonction ft_write_number convertit ce nombre en caractère et l'affiche
        ft_write_number(nb % 10);
}

// la récursivité fonctionne de manière à traiter les chiffres de nb
// du plus significatif au moins significatif
// mais ils sont affichés du moins significatif au plus significatif
// à cause de la pile d'appels récursifs

        // EXEMPLE :
        // 1) entrée : ft_write_numbers(1234) => placé sur la pile d'appels
        // appelle ft_write_numbers(123) (1234 / 10 => 123)
        // 2) entrée : ft_write_numbers(123) => placé sur la pile d'appels
        // appelle ft_write_numbers(12) (123 / 10 => 12)
        // 3) entrée : ft_write_numbers(12) => placé sur la pile d'appels
        // appelle ft_write_numbers(1) (12 / 10 => 1)
        // 4) entrée : ft_write_numbers(1)
        // appelle directement ft_write_number(1) (1 % 10 => 1)
        // ce qui affiche 1
        // FIN DE LA RECURSIVITE !

        // ATTENTION :
        // ON APPLIQUE LA REGLE LIFO
        // Last In First Out

        // EMPILAGE DES APPELS :

        // chaque appel récursif place un nouveau cadre sur la pile d'appels
        // chaque cadre attend que les appels récursifs suivants se terminent
        // avant de continuer :

        // l'appel de ft_write_numbers(1234) de l'étape 1) est placé tout en bas
        // de la pile
        // puis l'appel de ft_write_numbers(123) de l'étape 2) est placé
        // juste au-dessus de la pile
```

```
        // puis ft_write_number(12) de l'étape 3) est placé
        // au-dessus de la pile
        // ON A DONC :

        // *
        // ft_write_numbers(12)
        // ft_write_number(123)
        // ft_write_number(1234)


        // DESEMPILAGE DES APPELS :
        // lorsque ft_write_numbers(1) est atteint (étape 4)
        // la fin de la récursivité est atteinte
        // l'appel à ft_write_number(1) (1 % 10 => 1) est effectué directement
        // ce qui affiche 1 directement
        // puis on désempile les appels :
        // (qui étaient mis en pause)
        // on exécute, de haut en bas (voir *), ce qui se trouve
        // APRES L'APPEL RECURSIF :

        // on revient dans ft_write_numbers(12)
        // ft_write_number(12 % 10) => ft_write_number(2) => affiche 2
        // ft_write_number(123 % 10) => ft_write_number(3) => affiche 3
        // ft_write_number(1234 % 10) => ft_write_number(4) => affiche 4

// RESUME :
// ft_write_numbers(1234) (premier appel)
// ft_write_numbers(123) (appel récursif depuis 1234)
// ft_write_numbers(12) (appel récursif depuis 123)
// ft_write_numbers(1) (appel récursif depuis 12)
// ft_write_number(1) (affichage du dernier chiffre lors du retour de la récursivité)
// retour à ft_write_numbers(12) puis ft_write_number(2)
// retour à ft_write_numbers(123) puis ft_write_number(3)
// retour à ft_write_numbers(1234) puis ft_write_number(4)

// RESULTAT :
// affiche 1
// affiche 2
```

```
// affiche 3
// affiche 4

// cette fonction convertit un chiffre (entre 0 et 9)
// en son caractère ASCII en lui ajoutant '0'
// ('0' correspond à 48 en décimal)
// par exemple, si nb vaut 3
// alors c sera '3'
// voir table ASCII avec la commande : man ascii
// ce caractère c sera ensuite affiché avec write

void    ft_write_number(int nb)
{
        char    c;

        c = nb + '0';
        write(1, &c, 1);
}



int     main(void)
{
        int     nb;

        nb = 0;
        ft_putnbr(nb);
        return (0);
}
```