```c
#include <unistd.h>

void    ft_write_numbers(int n);
void    ft_write_number(int n);

// fonction principale appellée pour afficher
// le nombre entier n
void    ft_putnbr(int n)
{
        // -2147483648 est le minimum possible pour
        // un int (signé sur 32 bits)
        // ce cas particulier ne peut pas être géré
        // par la négation standard à cause de la
        // limite de représentation des nombres entiers
        // (2147483648 n'est pas représentable en int
        // car le maximum pour un int (signé sur 32 bits)
        // est 2147483647
        // on ne pourrait pas le gérer comme les autres
        // nombres entiers négatifs en écrivant '-'
        // puis en convertissant -2147483648 en sa
        // valeur positive)

        // on traite donc ce cas particulier en écrivant
        // directement ce nombre puis en terminant
        // l'exécution de la fonction

        // si n est égal à ce nombre
        if (n == -2147483648)
        {
                // on l'écrit directement
                // avec write, sur 11 octets
                // '-' fait 1 octet
                // auquel on ajoute 10 octets
                // pour 2147483648
                // (nombre à 10 chiffres)
                write(1, "-2147483648", 11);
```

```
                // on termine ensuite l'exécution
                // de la fonction ft_putnbr
                return ;
        }

        // si n est inférieur à 0
        if (n < 0)
        {
                // on écrit tout d'abord '-'
                write(1, "-", 1);

                // puis on convertit ce nombre
                // négatif en sa valeur absolue
                // (positive) pour simplifier
                // son traitement (voir plus bas)
                // les nombres négatifs seront donc
                // traités comme tous les autres
                // nombres (après avoir écrit '-'
                // devant
                n = -n;
        }

        // on exécute la fonction permettant de
        // traiter le nombre n
        ft_write_numbers(n);
}

// ATTENTION :
// cette fonction est récursive !
// exemple pour n = 123 :

// ft_write_numbers(123) EST EXECUTEE :
// 123 >= 10 ? OUI
// ft_write_numbers(123 / 10) => ft_write_numbers(12)
// PAUSE DE LA FONCTION ft_write_numbers(123)
// ET MISE EN BAS DE LA PILE
```

```
// ft_write_numbers(12) EST EXECUTEE :
// 12 >= 10 ? OUI
// ft_write_numbers(12 / 10) => ft_write_numbers(1)
// PAUSE DE LA FONCTION ft_write_numbers(12)
// ET MISE DANS LA PILE AU DESSUS DE ft_write_numbers(123)

// ft_write_numbers(1) EST EXECUTEE :
// 1 >= 10 ? NON
// ON PASSE DONC A
// ft_write_number(1 % 10) => ft_write_number(1)
// AFFICHE 1

// PUIS ON EXECUTE LE RESTE PILE :

// LA FONCTION ft_write_numbers(12) CONTINUE :
// ft_write_number(12 % 10)
// => ft_write_number(2)
// AFFICHE 2

// LA FONCTION ft_write_numbers(123) CONTINUE :
// ft_write_number(123 % 10)
// => ft_write_number(3)
// AFFICHE 3

// LA PILE EST VIDEE, FIN DE LA RECURSIVITE !

// ON UTILISE LA REGLE LIFO :
// Last In, First Out
void    ft_write_numbers(int n)
{
        // si n est supérieur ou égal à 10
        if (n >= 10)
                ft_write_numbers(n / 10);
        ft_write_number(n % 10);
}
```

```
// voir ex05
void    ft_write_number(int n)
{
        char    c;

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