```c
#include <unistd.h>

void    ft_print_hex_char(unsigned char c);


// prend une chaine de caractères comme argument et la parcourt pour traiter chaque caractère
void    ft_putstr_non_printable(char *str)
{
        // on parcourt la chaine jusqu'à la fin (jusqu'à rencontrer le caractère nul)
        int             i;
        unsigned char   hex_char;

        i = 0;
        while (str[i] != '\0')
        {
                // si le caractère est imprimable (entre le caractère ' ' (espace, code décimal ASCII 32),
                // et le caractère '~' (tilde, code décimal ASCII 126))
                if (str[i] >= 32 && str[i] <= 126)
                        // on l'écrit directement à l'écran
                        write(1, &str[i], 1);
                // sinon
                else
                {
                        // on écrit le caractère \ (sans oublier de l'échapper avec un autre \)
                        write(1, "\\", 1);

                        // on convertit le caractère en cours en unsigned char
                        // pour pouvoir le traiter correctement ensuite dans ft_print_hex_char :
                        // la fonction ft_print_hex_char doit recevoir un "caractère positif"
                        // (c'est-à dire le code décimal positif ASCII de ce caractère)
                        // pour en extraire son code hexadécimal
                        // selon le compilateur utilisé, le type char peut être :
                        // signé (peut être positif ou négatif)
                        // ou non signé (peut être uniquement positif)

                        // comme ft_putstr_non_printable prend un char * comme paramètre
                        // (et non un unsigned char *)
                        // il faut convertir chacun des char en unsigned char
```

```
                        // avant de les passer à ft_print_hex_char

                        // REMARQUE : ft_putstr_non_printable prend en paramètre char *str
                        // (et non pas unsigned char *str) pour des questions de flexibilité et de performance

                        // REMARQUE : hex_char est une variable de type unsigned char (voir plus haut)
                        // la conversion (avec l'opérateur de cast "(unsigned char)")
                        // de str[i] est dite EXPLICITE
                        // (on aurait pu écrire directement "hex_char = str[i];")
                        // la conversion serait alors IMPLICITE
                        hex_char = (unsigned char)str[i];

                        // on exécute ft_print_hex_char, qui prend en paramètre le caractère en cours
                        // pour le convertir et l'afficher sous forme hexadécimale (en minuscule)
                        ft_print_hex_char(hex_char);
                }
                // on incrémente i pour passer au caractère suivant
                i++;
        }
}

// convertit un caractère en son équivalent hexadécimal
// en affichant les deux chiffres hexadécimaux correspondant
void    ft_print_hex_char(unsigned char c)
{

        char    *hex;

        // table hexadécimale minuscule
        // (chaine de caractères contenant tous les caractères hexadécimaux en minuscule)
        // chaque caractère de cette chaine de caractères a une position précise dans la table :
        // l'indice (la position) de chaque caractère dans la table correspond en fait à la valeur numérique
        // de chaque chiffre hexadécimal
        // Ainsi, a (en hexadécimal) correspond à 10 (en valeur numérique) (car a est à l'indice 10 de la table)
        // f (en hexadécimal) correspond à 15 (en valeur numérique) (car f est à l'indice 10 de la table)
        // on a donc :
        // valeur:    0 1 2 3 4 5 6 7 8 9 a  b  c  d  e  f
        // index:     0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

```
        hex = "0123456789abcdef";

        // un caractère est codé sur 8 bits : sa valeur (son code décimal) est comprise entre 0 et 255
        // en unsigned char (voir table ASCII)
        // cela correspond à 1 octet
        // exemple : le caractère de code décimal ASCII 245 est représenté en binaire par 11110101
        // (1 + 4 + 16 + 32 + 64 + 128)

        // on peut représenter un octet (8 bits) par 2 groupes de 4 bits (2 nibbles) :
        // exemple : 1111 0101
        // le bit le plus à gauche est appelé "high nibble"
        // le bit le plus à droite est appelé "low nibble"

        // chaque nibble peut valoir entre 0 et 15 (0000 vaut 0 et 1111 vaut 15)
        // ce qui correspond exactement aux valeurs que peut prendre un chiffre hexadécimal

        // on peut obtenir la valeur du high nibble en divisant (division entière)
        // la valeur décimale du caractère (en unsigned int) par 16
        // et la valeur du low nibble en effectuant le modulo 16 (le reste de la division entière)
        // de la valeur décimale du caractère (en unsigned int)
        // (en unsigned int) par 16
        // exemple :                    245
        //                    1 + 4 + 16 + 32 + 64 + 128
        //                         11110101
        //             1111                    0101
        //             high nibble             low nibble
        //             1 + 2 + 4 + 8 = 15      1 + 4 = 5
        //             245 / 16 = 15           245 % 16 = 5 (car 245 - 15 * 16 = 245 - 240 = 5)

        // les valeurs du high nibble et du low nibble correspondent chacune à la position (l'indice)
        // du caractère hexadécimal correspondant dans la table hexadécimale
        // 15   5
        // f    5

        // on indique donc respectivement c / 16 et c % 16 comme indice de la table hexadécimale hex
        write(1, &hex[c / 16], 1);
        write(1, &hex[c % 16], 1);
}
```

```c
#include "ft_strcpy.h"
#include "ft_putstr.h"
#include "ft_putstr_non_printable.h"
#include <unistd.h>

int     main(void)
{
        char    src1[14];
        char    src2[21];
        char    str1[32];
        char    str2[2];
        char    str3[132];
        char    str4[3];
        int     i;
        int     j;

        ft_strcpy(src1, "Hello World !");
        ft_putstr_non_printable(src1);
        write(1, "\n", 1);
        src2[0] = 'C';
        src2[1] = 'o';
        src2[2] = 'u';
        src2[3] = 'c';
        src2[4] = 'o';
        src2[5] = 'u';
        src2[6] = '\n';
        src2[7] = 't';
        src2[8] = 'u';
        src2[9] = ' ';
        src2[10] = 'v';
        src2[11] = 'a';
        src2[12] = 's';
        src2[13] = ' ';
        src2[14] = 'b';
        src2[15] = 'i';
        src2[16] = 'e';
        src2[17] = 'n';
```

```
	src2[18] = ' ';
	src2[19] = '?';
	src2[20] = '\0';
	ft_putstr_non_printable(src2);
	write(1, "\n", 1);
	i = 0;
	j = 1;
	while (i < 32)
	{
		str1[i] = j;
		i++;
		j++;
	}
	str1[i] = '\0';
	ft_putstr_non_printable(str1);
	write(1, "\n", 1);
	str2[0] = 127;
	str2[1] = '\0';
	ft_putstr_non_printable(str2);
	write(1, "\n", 1);
	i = 0;
	j = 127;
	while (j <= 256)
	{
		str3[i] = j;
		i++;
		j++;
	}
	str3[i] = '\0';
	ft_putstr_non_printable(str3);
	write(1, "\n", 1);
	str4[0] = 31;
	str4[1] = 32;
	str4[2] = '\0';
	ft_putstr_non_printable(str4);
	write(1, "\n", 1);
	return (0);
}
```