```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)
```

```
103       // du caractère hexadécimal correspondant dans la table hexadécimale
104       // 15 5
105       // f 5
106
107       // on indique donc respectivement c / 16 et c % 16 comme indice de la table hexadécimale hex
108       write(1, &hex[c / 16], 1);
109       write(1, &hex[c % 16], 1);
110   }
111
112
113   #include "ft_strcpy.h"
114   #include "ft_putstr.h"
115   #include "ft_putstr_non_printable.h"
116   #include <unistd.h>
117
118   void  ft_print(int min, int max);
119
120   int main(void)
121   {
122     char  str[21];
123
124     ft_strcpy(str, "Coucou\ntu vas bien ?");
125     ft_putstr_non_printable(str);
126     write(1, "\n", 1);
127     ft_print(1, 31);
128     ft_print(32, 126);
129     ft_print(127, 255);
130     return (0);
131   }
132
133   void  ft_print(int min, int max)
134   {
135     char  str[150];
136     int   i;
137     int   j;
138
139     i = 0;
140     j = min;
141     while (j <= max)
142     {
143       str[i] = j;
144       i++;
145       j++;
146     }
147     str[i] = '\0';
148     ft_putstr_non_printable(str);
149     write(1, "\n", 1);
150   }
151
```