```
// on indique au compilateur d'inclure le contenu du fichier d'entête unistd.h
// (bibliothèque standard C permettant d'effectuer des appels système)
// pour pouvoir utiliser la fonction write
// # permet de faire appel au compilateur
// les (chevrons) < et > entourant le nom du fichier d'entête
// signifient qu'on fait appel à une bibliothèaue standard

#include <unistd.h>

// on déclare la fonction ft_putchar
// qui prend en paramètre c
// de type char (caractère)
// (entre parenthèses)

// cette fonction ne retourne rien
// (elle ne retourne pas de valeur avec return
// comme par exemple "return c;")
// on indique donc void avant le nom de la fonction
// (en séparant void de la fonction par une tabulation
// pour respecter la norme)
void    ft_putchar(char c)

// on ouvre le bloc de code de la fonction
{
        // on appelle la fonction write
        // (qui écrit des données vers un descripteur de fichier,
        // qui est une référence pour accéder à un fichier,
        // ou à un autre objet d'entrée-sortie)
        // avec ces paramètres :

        // le descripteur de fichier : 1
        // 1 correspond à la sortie standard (stdout)
        // grâce à ce paramètre, le texte sera affiché
        // à l'écran

        // l'adresse du caractère c à afficher : &c
        // on utilise le symbole & pour obtenir
        // l'adresse mémoire de la variable c
```

```
        // afin que la fonction puisse
        // accéder à sa valeur en mémoire

        // le nombre d'octets à écrire depuis l'adresse fournie
        // c est de type char
        // un char pèse 1 octet
        // on indique donc 1
        // car 1 seul caractère doit être écrit

        // on termine la ligne avec ;
        write(1, &c, 1);

// on ferme le bloc de code de la fonction
}


// pour utiliser cette fonction :

// on écrit un header pour notre fonction :
// on crée le fichier ft_putchar.h
// on y indique le prototype de notre fonction :
void    ft_putchar(char c);

// on écrit un programme principal (main) qui incluera le header
// de notre fonction
// et fera appel à celle-ci avec un caractère en paramètre
// pour que notre fonction l'écrive :

// on crée le fichier main.c
// on inclus le header de notre fonction
// en indiquant le nom du fichier
// entre double guillemets
// car ce fichier d'entête a été
// créé par l'utilisateur
// il ne fait pas partie de la bibliothèque standard
#include "ft_putchar.h"

// on déclare la fonction main
```

```
// qui retournera une valeur de type entier
// (voir plus bas)

// en C, le point d'entrée d' un programme est main
// lorsque le programme sera exécuté
// l'éxécution commencera pae la première ligne de code dans main

// la fonction main ne prend aucun paramètre (ou argument)
// on indique cela avec (void)
int     main(void)
{
        // on fait appel à la fonction ft_putchar avec le paramètre 'a'
        // les guillemets simples entourent un caractère
        ft_putchar('a');

        // on termine l'exécution de la fonction main
        // donc du programme
        // et on indique au système que le programme
        // s'est terminé avec succès
        // en retournant la valeur 0
        // on indique 0 entre parenthèses
        // avec un espace après le mot clé return
        // pour respecter la norme
        return (0);
}

// on compile, avec gcc (gnu complier collection)
// nos deux fichiers C
// les fichiers ft_putchar.c et main.c
// en un seul exécutable
// qui sera nommé a.out

// on utilise -Wall pour activer un grand nombre
// d'avertissements (Warnings) pouvant entrainer
// des comportements inattendus
// lors de l'exécution

// -Wextra active encore plus d'avertissements (Warnings) que -Wall
```

```
// (constructions douteuses, pouvant causer problèmes ou confusions)

// -Werror fait en sorte que tous les avertissements (Warnings)
// soient traités comme des erreurs
// la compilation échouera donc si gcc produit un avertissement
// donc le fichier exécutable ne sera pas généré
// tant que tout ne sera pas résolu

gcc -Wall -Wextra -Werror ft_putchar.c main.c

// le fichier a.out sera ainsi créé
// on peut l'exécuter avec la commande suivante :
./a.out
```