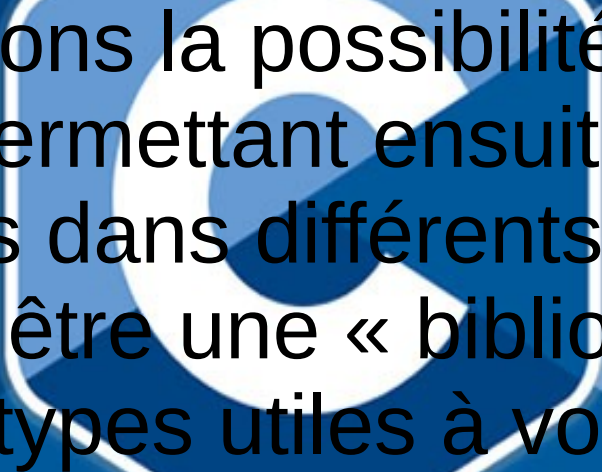


# Les includes



En c, nous avons la possibilité de créer des headers (.h) permettant ensuite d'utiliser des fonctions dans différents fichiers, Ces fichiers vont être une « bibliothèque » de tous les prototypes utiles à vos fichiers.

# Les includes

```
#ifndef INCLUDES_H
# define INCLUDES_H

# include <stdio.h>
# include <stdlib.h>
# include <unistd.h>

void gd_putnbr(int nb);
void gd_putchar(char c);
int gd_strlen(char *str);

#endif
```

Voilà un exemple de fichier

# Les includes

```
#include "includes.h"
```

Dans vos fichiers c, il suffira d'inclure ce fichier avec son path relatif, cette fois entre guillemet.

# Typedef

Typedef va nous permettre de créer nos propres types.

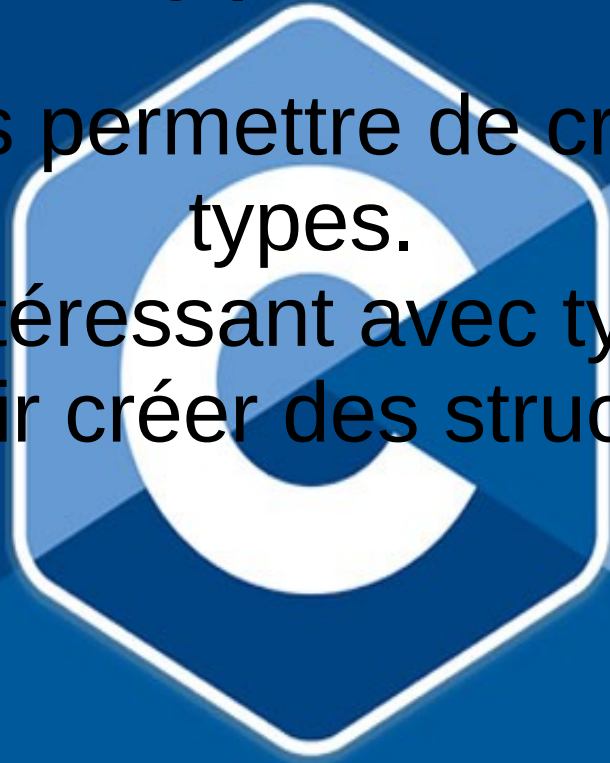
Par exemple, nous pouvons, par exemple :

```
#define VRAI 1  
#define FAUX 0  
typedef int BOOLEAN;
```

# Typedef

Typedef va nous permettre de créer nos propres types.

Mais le plus intéressant avec typedef c'est de pouvoir créer des structures.



# Typedef/structure

Nom de ma structure

```
typedef struct my_struct {  
    int minutes;  
    int hours;  
} type_struc;
```

Variables  
dans ma  
structure

Nom de  
mon type  
redéfini

# Typedef/structure

Ces structures nous permettent alors de palier au return unique obligatoire en C.

Effectivement, à présent, comme nous avons redéfini un type. Ce type peut être utilisé en retour de fonction. Il peut aussi, bien sûr, être utilisé en paramètre.

Voilà donc le nouveau prototype de « h\_m » vu au tout départ.

Type de retour

Nom de la fonction

Paramètre de type  
« type\_struct »

```
type_struct h_m(type_struct my_struct)
```

# Typedef/structure

On peut donc maintenant réécrire notre fonction  
h\_m

```
type_struct h_m(type_struct my_struct) {  
    my_struct.hours = my_struct.minutes / 60;  
    my_struct.minutes = my_struct.minutes % 60;  
    return(my_struct);  
}
```

Ici, le point nous sert à accéder à la variable correspondante