Le préprocesseur

Le préprocesseur C est appelé avant la compilation sur le fichier source. Les directives du préprocesseur sont sur une ligne commençant par le caractère « # ».

Définition d'une constante

Syntaxe :

```
#define identificateur chaine
```

Exemples :

```
#define FALSE 0
#define TRUE 1
#define SIZE 255
#define IF if (
#define THEN ){
#define ELSE ;} else {
#define ENDIF ;}

IF a < b
THEN
  x = y + z;
  z = w
ELSE
  a = 2 * b
ENDIF
```

Définition d'une macro

Syntaxe :

```
#define identificateur(x1, … , xn) chaine
```

Exemples :

```
#define getchar() getc(stdin)
#define min(a, b) ((a) < (b) ? (a) : (b))
#define max(a, b) ((a) > (b) ? (a) : (b))
```

Portée :

La portée est alors étendue au fichier contrairement à une fonction dont la portée est locale, exemple :

```
#define SWITCH(X, Y) {char tmp=X; X=Y; Y=tmp; } //Fonctionnel
void switch1(char x, char y) {char tmp=x; x=y; y=tmp; } //Local
void switch2(char *x, char *y) {char tmp=x; x=y; y=tmp; } //Pointeurs non utilisés, Local
void switch3(char *x, char *y) {char tmp=*x; *x=*y; *x=tmp; } //Fonctionnel
```

On peut définir une macro sur plusieurs lignes à l'aide de « \ » :

```
#define Positif(fct)            \
int Positif_##fct(double x)     \
{                               \
double res = fct(x);            \
return res < 0 ? 0 : res;       \
}
```

L'utilisation de « ## » permet de concaténer une macro, en suivant l'exemple précédent on aurait par exemple :

```
Positif(sin);
/*Equivaut à :
int Positif_sin(double x){
      double res = sin(x);
      return res < 0 ? 0 : res;
}; */
```

Macro prédéfinies

__LINE__ : le numéro de la ligne actuelle.
__FILE__ : le nom du fichier actuel.
__DATE__ : la date de la compilation.
__TIME__ : l'heure de la compilation.
__STDC__ : 1 si le compilateur est conforme à la norme ANSI
__STDC_VERSION__ : vaut 199901L si C99

Définition à la compilation

Il est possible d'effectuer des « define » à la compilation (sans modifier le code) à l'aide de « -D », exemples :

```
gcc -c -DMAXBUF=150 -DOS=linux -DDEBUG buffer.c
```

Équivaut à écrire dans le code :

```
#define MAXBUF 150
#define OS linux
#define DEBUG
```

Stringification

Permet « d'échapper » un caractère dans une macro pour qu'il ne soit pas interprété, en ANSI C l'opérateur de stringification est un « # », exemple :

```
#define TRACE(x) printf("%s = %d\n", #x, x)
TRACE(3*2+4); //Affiche 3*2+4=10 où « 3*2+4 » est une chaine
```

Oublie (undef)

Permet d'oublier un « define », exemples :

```
#define IF if (
IF a < b) x=0 ; //Ok, x vaut désormais 0 (si il a bien été défini)
#undef IF
IF a < b) x=0 ; //Erreur « IF » n'existe pas (plus)
```

Inclusion

Inclusion standard

```
#include <fichier>
```

Cherche le fichier dans les bibliothèques « classiques » de C (par exemple « /usr/include » sous Linux).

Inclusion personnalisée

```
#include "fichier"
#include "../lib/fichier"
```

Cherche à partir de la racine

Il est possible d'ajouter des chemins d'inclusions lors de la compilation à l'aide de « -I », exemple :

```
$ gcc -I../../my-include -I/usr/local/include foo.c
```