[LC026]

Macros en langage C

biblio [TCU page]
[K&R page]

Qu'est ce qu'une macro ?

Une *MACRO-substitution* ou simplement *MACRO* est un mot que l'on définit pour remplacer toute une suite de caractères, de mots, voire d'instructions. La substitution sera faite avant la compilation, c'est-à-dire que le compilateur ne verra pas la macro, mais plutôt ce par quoi elle est remplacée. Une des particularités des *macros* en langage C, est la possibilité qu'elles ont d'avoir des paramètres, et de ce fait d'être utilisées comme des fonctions. Pour diverses raisons que nous verrons, il est cependant important d'être conscient qu'une telle macro n'est pas vraiment équivalente à une fonction : Une fonction n'a qu'une seule implantation, et sera seulement appelée de divers endroit au moment de l'exécution, tandis que la substitution faite par une macro est faite avant la compilation, et chaque fois qu'elle est rencontrée. Par contre il n'y a pas d'appel à l'exécution.

Traditionnellement en langage C les identificateurs de macros sont entièrement en majuscules.

Macros simples (→ constantes)

Définitions

La syntaxe d'une définition de macro simple est la suivante :

```
#define identificateur suite_de_caractères
#define TABSIZE 100
#define MonFichier "/u/blanc/truc.h"
#define ERREUR printf("erreur")
#define BIP putchar('\007');
#define CLS  { putchar('\033')     ;      putchar      ('H');\
               putchar('\033') ; putchar ('J'); }
```

ESCAPE

On remarquera dans cette dernière définition la possibilité de définir une macro sur plusieurs , grace à l'anti-slash utilisé pour neutraliser les fins de lignes que l'on veut ignorer. Ce doit être le caractère de la ligne.

Utilisation

Pour utiliser une macro, il suffit de mettre son . Noter cependant qu'une macro n'est pas substituée dans une chaîne de caractères (entre guillemets ""), ni dans un commentaire (entre /* */ ou après //).

```
fopen(MonFichier, "r");
#include MonFichier
#include "MonFichier.h"
...
int table[TABSIZE]; char *ville[TABSIZE]
...
main()
{
CLS
...
if (k==7) CLS else { BIP ERREUR;}
...
}
```

On remarquera que lorsqu'une macro correspond à une suite d'instructions se terminant chacune par un point-virgule (exemples BIP et CLS), il n'est pas nécessaire, et il est même souvent néfaste, de terminer "l'appel" de la macro par un point-virgule. Si par contre le point-virgule n'est pas inclu dans la définition de macro, il devra être mis lors de l'utilisation (exemple ERREUR).

Annulation d'une macro

Il est parfois nécessaire d'annuler une définition de macro. La syntaxe en est :

```
#undef identificateur
```

et en voici deux exemples :

```
#undef TABSIZE
#undef CLS
```

Macros avec paramètres

On peut donc définir en C des macros avec paramètres. La syntaxe d'une telle définition est la suivante :

```
#define ident(param1,param2,...,paramn) suite_de_caractères
```

et en voici un exemple à un seul paramètre :

```
#define OUVRE(nfich)     \
     fic=fopen(nfich,"r"); \
     if (fic==NULL)        \
     {      printf("fichier      %s      non      trouvé!...\n",nfich);\
     return FAUX; }
```

Dans cet exemple on notera les anti-slashes de neutralisation de fins de lignes, qu'il est préférable d'aligner pour mieux les repérer à la lecture du programme. Prendre garde qu'il ne faut pas en mettre sur la dernière ligne.

L'utilisation d'un telle macro est syntaxiquement identique à l'appel d'une fonction, avec même remarque que précédemment en ce qui concerne le point-virgule :

```
OUVRE("ETOILES.LIS")
```

Répétons cependant que macros et fonctions ne sont pas vraiment équivalents : Une macro est une substitution de caractères faite avant la compilation, tandis qu'une fonction est apppelée à l'exécution. Le code d'une macro figurera dans le programme autant de fois qu'on l'aura utilisée, tandis que le code d'une fonction ne figure qu'une seule fois. Une macro est par contre exécutée plus rapidement, puisqu'il n'y a pas d'appel proprement dit. Un programme qui ne serait écrit qu'avec des macros en guise de fonctions serait donc plus gros, mais plus rapide que le même programme écrit avec des fonctions.

Attention aussi à certains pièges dus au fait que les macros ne sont que des substitutions de caractères. Voyons les sur les deux exemples suivants :

```
#define CARRE(X)    X * X
#define CUBE(X)     (X) * (X) * (X)
... x = 8; z = 3;
printf("%f ",5*CARRE(x+2))
...
y = CUBE(z++); printf("%d %d",y,z);
```

5x x+2 x x+2 = 5x12 (58 ici).

27ou 60, 6

3x4x5

Enfin, l'annulation d'une macro avec paramètre se fait comme pour une macro simple :

```
#undef OUVRE
```