Compléments sur les macros du préprocesseur

Macros prédéfinies

La norme du langage C définit un certain nombre de macros du préprocesseur, en particulier :

nom	valeur de la macro	type
`__LINE__`	numéro de la ligne courante du programme source	entier non signé
`__FILE__`	nom du fichier source	chaîne
`__DATE__`	la date de la compilation	chaîne
`__TIME__`	l'heure de la compilation	chaîne
`__STDC__`	1 si le compilateur est ISO, 0 sinon	entier non signé

Selon l'implémentation, d'autres macros peuvent s'ajouter à cette liste. Par exemple, gcc offre la macro `__FUNCTION__` qui contient le nom de la fonction courante.

Les macros avec paramètres

Il est possible de passer des paramètres à une macro `#define` afin de réaliser une fonction en ligne (fonction inline du C++).

Intérêt

Par rapport à un appel de fonction traditionnel, la vitesse d'exécution est optimisée (économie du code d'appel et de retour de la fonction).
De plus, dans la mesure ou le type des paramètres passés à une macro ne donne pas lieu à une vérification lors de la compilation, il est possible de créer une séquence de code valable quel que soit le type de ses paramètres. En revanche, cette absence de contrôle du typage impose de redoubler de précautions dans l'élaboration de ce code!

Définition

Une macro avec paramètres se définit sur une ligne de la manière suivante :

`#define` **`NOM_MACRO`**`(` ***`liste-de-paramètres-formels`*** `)` *`corps_macro`*

`liste-de-paramètres-formels` : liste d'identificateurs séparés par des virgules
`corps_macro` : corps de la macro

L'appel de la macro devra avoir la forme :

`NOM_MACRO` `(` ***`liste-de-paramètres-effectifs`*** `)`

`liste-de-paramètres-effectifs` : paramètres d'appel séparés par des virgules.

Le préprocesseur remplace alors le nom de la macro et la liste de paramètres effectifs parenthésés, par le corps de la macro dans lequel chaque paramètre formel est remplacé par le paramètre effectif correspondant. Cette opération de remplacement de texte porte le nom d'expansion de la macro.

Exemple

```
/* definition des macros min et max */
```

```
#define min(a, b)       ((a) < (b) ? (a) : (b))
#define max(a, b)       ((a) < (b) ? (b) : (a))

int main(void) {
     int i,j,k;
     double x, y z;
     …
     …
     i = min(j,k);
     z = max(x,y);
     …
     }
```

Lors du traitement par le préprocesseur, les macros sont remplacées par le code correspondant. Le programme qui sera finalement compilé est le suivant :

```
int main(void) {
     int i,j,k;
     double x, y z;
     …
     …
     i = ((j) < (k) ? (j) : (k));
     z = ((y) < (z) ? (z) : (y));
     …
     }
```

Attention!

La distinction entre macro avec et sans paramètre se fait sur le caractère qui suit **immédiatement** le nom de la macro : si ce caractère est une parenthèse ouvrante, c'est une macro avec paramètres, sinon c'est une macro sans paramètre.
En particulier, s'il y a un blanc avant la parenthèse ouvrante, la ligne sera interprétée comme une macro sans paramètre.
L'exemple suivant illustre le risque de confusion :

```
#define CARRE (a) a * a
```

La ligne de code :

```
quatre = CARRE(2);
```

aura ici comme expansion:

```
quatre = (a) a * a(2);
```

et non :

```
quatre = 2*2;
```

Application : tracé des erreurs sous gcc

Le programme suivant contient deux macro avec paramètres:

`PRINTERR` permet d'afficher un message d'erreur utilisateur,
`PRINTERRSYS` permet d'afficher un message d'erreur utilisateur suivi du message d'erreur système correspondant à l'erreur.

```
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>

#define PRINTERR(msg) fprintf(stderr, "Erreur: %s\n fichier: %s\n fonction:
%s\n ligne: %u \n", msg, __FILE__, __FUNCTION__, __LINE__ -1)

#define PRINTERRSYS(msg) perror(msg);fprintf(stderr, " fichier: %s\n
fonction: %s\n ligne: %u \n", __FILE__, __FUNCTION__, __LINE__ -1)

int main(int argc, char **argv){

  int fd;

  /* test du nombre d'arguments : erreur si argc different de 2 */
  if (argc != 2) {
    PRINTERR("Nombre arguments incorrect");
    exit(1);
  }

  if ((fd = open(argv[1], O_RDONLY)) == -1){
    PRINTERRSYS("Erreur ouverture fichier");
    exit(1);
  }

  printf("fini\n");
  return 0;
}
```