[LC050]

Fonctions

DÉCLARATION, DÉFINITIONS

• En C toute fonction est globale (ou externe).
C'est-à-dire que la **définition** d'une fonction doit être à l'extérieure de toutes autres définitions de fonctions.

• Une telle **définition** présente au compilateur :

- La **classe** d'une fonction qui peut être :

  extern **fonction externe** → accessible depuis toutes les fonctions. (par défaut)

  static **fonction statique** → accessible depuis toutes les fonctions définies dans le même fichier.
  → une fonction de même nom dans un autre fichier est une fonction différente.

- Le **type** d'une fonction qui est celui du resultat. qu' elle est censée retourner.

  (Une fonction de type **void** ne retournent rien (= procédure)).

- Les types de ses arguments.

- Le **corps** de la fonction. (voir syntaxe définition ci-dessous)

• Une déclaration de fonction présente au compilo :
la classe et le type d'une fonction qui est définie ailleurs (soit plus loin dans le même fichier, soit dans un autre fichier).
En style moderne, c'est-à-dire suivant la norme ANSI, une telle déclaration présente aussi le type des arguments de sa fonction. Pour cette raison, une déclaration norme ANSI est aussi appelée un **prototype** (voir la syntaxe ci-après).

• Il peut y avoir plusieurs déclarations d'une même fonction, mais une seule définition.

• En norme ANSI toute fonction doit être déclarée ou définie avant d' être utilisée (appelée)[1].

SYNTAXES DES DÉCLARATIONS ET DÉFINITIONS EN STYLE MODERNE

En style moderne (c'est-à-dire suivant la norme ANSI), les syntaxes des déclarations et définitions de fonctions sont les suivantes :

• Syntaxe[2] d'une déclaration de fonction (style moderne) (ANSI, TC ...)

[classe] type identificateur[3] (*type [arg1]*[4] , *type [arg2]*, ...) ;

---

[1] par contre le style classique autorise des déclarations implicites : voir le § sur ce style
[2] Dans toutes les syntaxes de ce document les crochets [ ] indiquent un élément facultatif.
[3] c'est-à-dire le nom de la fonction.
[4] Dans la déclaration en style moderne le nom des arguments n'est pas indispensable ; mais une bonne habitude est de l'indiquer quand même en guise d'auto-commentaire.

• Syntaxe[2] d'une définition de fonction (style moderne)

[classe] *type identificateur*[3] (*type arg1* , *type arg2*, ...)
{
/* *déclarations des variables locales* */
/* *instructions* */
}

Exercice-exemple 1 :
Compléter les ***déclarations*** *des fonctions suivantes (en style moderne) :*

float quotient (float, float); /* Cette fonction retourne un réel de type float et possède deux arguments (divid et divis) de type float */

void affiche(float); /* Cette fonction est en fait une procédure et possède un argument de type float */

void donn_nbres (float*, float*); /* Cette fonction est aussi une procédure et possède deux arguments de type [float *] (pointeur sur float) */

Exercice-exemple2 : Écrire les ***définitions*** *des fonctions précédentes (en style moderne).*

```
float quotient(float divid, float divis)
{
if (divis == 0.) return INFINI ;
else return divid/divis ;
}
```

```
void affiche (float nbre)
{
int i ;
if (nbre == INFINI )
    { for ( i=0 ; i<3 ; i++ ) puts ("impossible!") ;  return ; }
else printf ( "%g" , nbre ) ;
}
```

```
void donn_nbres(float* p1, float* p2)
{
printf("Donnez deux nombres réels : ");
scanf("%g  %g" , p1 , p2);
}
```

---

[2] Dans toutes les syntaxes de ce document les crochets [ ] indiquent un élément facultatif.
[3] c'est-à-dire le nom de la fonction.

UTILISATION DES FONCTIONS

- **Retour** au programme appelant

Parmi les instructions du corps d'une fonction, peuvent-être placées une ou plusieurs instructions return qui permettent le retour au programme appelant. On peut les faire suivre d'une variable dont la valeur sera le résultat retourné par la fonction.

- **Appel** d'une fonction

* **soit** sous la forme d'une instruction. Si la fonction retourne un résultat, celui-ci est alors non utilisé.

Exemples 3 : puts ("Q pour quitter") ; printf (" i = %d\n " , i) ;

* **soit** dans une expression. Le résultat retourné est alors utilisé dans le calcul de l'expression.

Exemples 4 : y = z + sqrt (x) ; n = printf (" résult : %d ", exp(x) + y/2) ;

*Exercice 5 : Voici un programme principal (fonction **main()**) dans lequel sont appelés les fonctions des exos 1 & 2, ainsi qu'un certain nombre d'autres fonctions :*

```
main()
{ float a,b; float r; int i,j,k;
puts[1] ("tapez la touche Q pour quitter") ;
donn_nbres[2] ( &a, &b) ;   affiche[3](quotient[4](a,b)) ;
r += quotient[5](a,b) ; affiche[6](r + sqrt[7](2)) ;
k = scanf[8]("%d %d", &i,&j);
if (k == 2 && fabs[9](a-b) >3.0 )        k = printf[10]("i=%d  j=%d",i,j) ;
gets[11]( ch ) ;
}
```

Quels sont les appels du type instruction ?

1 2 3 6 11

Quels sont les appels pour lesquels la valeur retournée est utilisée dans une expression ?

4 5 7 8 9 10

• Les **arguments** d'une fonction sont transmis par valeur.

Mais si on déclare un argument comme un pointeur :

Exemple : void donn_nbres(float *p1, float *p2)

Alors l'objet pointé pourra être modifié :

Dans ce cas, l'appel de la fonction devra être fait avec les adresses des arguments effectifs :

• Tout argument tableau doit être aussi déclaré comme pointeur :

Ici l'appel de la fonction devra être fait en donnant le nom seul du tableau comme argument effectif. Si en outre la fonction a besoin de la taille d'un tel tableau, celle-ci devra être passée comme un autre argument (ou bien être variable globale). *Exemples* :

Exercice-exemple 6 :

Compléter la définition de la fonction ***moy*** *ci-dessous (en style moderne):*

```
/* fonction retournant la moyenne des n entiers d'un tableau tt */
float moy ( int * tt , int n )
{
    int i , som=0;
    for ( i=0 ; i<n ; i++ )   som += tt[i] ;
    return som / (float)n;
}                              /* Pensez vous que la moyenne de n entiers est un entier ? */
```

Exercice-exemple 7:

Faire afficher, par ***une seule instruction****, la moyenne des éléments du tableau* ***tab*** *défini ci-après .*

```
int tab[ ]={ 3 , 312 , 215 , 178 , 456 , 98 , 45 , 21 , 546 , 356 , -21 } ;
```

Exercice-exemple 8 :

Donner la définition de la fonction ***strcpy*** *qui recopie la chaîne* ***sourc*** *dans la chaîne* ***dest.***

On supposera en outre que ***strcpy*** *ne retourne aucun résultat (ce qui n'est pas vrai en réalité).*

```
void strcpy (char *dest , char *sourc )
{   for(; *source; source++, dest++)
        *source = *dest;
    *dest = 0;
}
```

ou bien :

```
while (*dest++ = *source++);
```

COMPLÉMENT : SYNTAXES EN STYLE CLASSIQUE

Lorsque Kernighan et Ritchie ont conçu le langage C, les syntaxes des déclarations et définitions de fonctions étaient légèrement différentes de celles préconisées ensuite par la norme ANSI (style moderne). Actuellement le style classique est encore utilisé, car de nombreuses bibliothèques existent qui n'ont pas été réécrites en style moderne. La plupart des compilateurs acceptent les deux styles. Certains compilateurs (par exemple **cc** sous certains **Unix**) nécessitent de préciser explicitement (par une option) que l'on souhaite suivre la norme ANSI.

> Il est bien entendu recommandé de programmer en style moderne, afin de permettre une meilleure vérification de ce qu'on a écrit.
> Il est d'ailleurs aussi recommandé, dans le même but, de configurer le compilateur afin qu'il avertisse lorsqu'un prototype est absent, et d'une façon plus générale, lorsqu'on ne suit pas la norme ANSI.

• Syntaxe[6] d'une déclaration de fonction (Style classique)

[*classe*] [*type*] *identificateur*[5] ( ) ;

En style classique, une déclaration de fonction n'indique pas le type des arguments. Le compilateur n'a pas besoin de les connaître pour coder des appels de fonctions. Rappelons que la classe par défaut est extern et que le type par défaut est int. Pour cette raison, on peut aussi se passer, en style classique, de faire une déclaration lorsqu'il s'agit d'une fonction de type int. En effet, le compilateur fera, dans ce cas, une déclaration implicite de type int lors de la première utilisation (appel) de la fonction.

• Syntaxe[6] d'une définition de fonction (Style classique)

[*classe*] [*type*] *identificateur*[5](*arg1* , *arg2* , ..., *argn*)
[*type*] *arg1* ; [*type*] *arg2* ; ... ; [*type*] *argn* ; /* déclarations des arguments */

{
/* déclarations des variables locales */
/* instructions */
}

En style classique, une définition de fonction donne d'abord, outre la classe et le type, la liste des arguments. Ceux-ci sont ensuite déclarés, juste après cette liste, donc avant l'accolade d'ouverture du bloc. Il est important de ne pas confondre ces déclarations avec celles des variables locales, lesquelles sont après l'accolade.

Exercice-exemple 9 :
Compléter les ***déclarations*** *(en style classique) des mêmes fonctions que celles de l'exemple-exercice 1 :*

_____ quotient_____
_____ affiche_____
_____ donn_nbres_____

• Remarque : Les déclarations de fonctions et de variables peuvent être mélangées.
Exemples :

---

[5] c'est à dire le nom de la fonction.
[6] Dans cette syntaxe les crochets [ ] indiquent un élément facultatif.

Exercice-exemple 10 :

Compléter les ***définitions*** *(style classique) des mêmes fonctions :*

```
_____  quotient(        ,        )
_____________
{
if (divis == 0.) return INFINI ;
else return divid/divis ;
}
```

```
_____  affiche (              )
_____________
{
_____________ ;
if (nbre == INFINI )
     { for ( i=0 ; i<3 ; i++ ) puts ("impossible!") ;   return ; }
else printf ( "%g" , nbre ) ;
}
```

```
_____  donn_nbres(    ,    )
_____  _____ , _____ ;
{
printf("Donnez deux nombres réels : ");
scanf("%g  %g" , p1 , p2);
}
```

Exercice-exemple 11 :
Réécrire, en style classique, les définitions (ou tout au moins le début) des fonctions ***moy*** *et* ***strcpy*** *vues dans les exercices 6 et 8.*

```
_____ moy ( _____ , _____ )
_____________________________
{
../* corps de la fonction */..
}

_____ strcpy ( _____ , _____ )
_____________________________
{
../* corps de la fonction */..
}
```