

6. Modes de passage de paramètres

Avant de décrire les modes de passage de paramètres nous devons distinguer les notions suivantes :

Variable globale

Est une variable définie dans l'en-tête du programme principal. Elle est utilisable dans n'importe quel sous-programme sans nécessité de redéfinition.

Toutefois, si dans un sous-bloc il existe une variable qui porte le même nom que la variable globale, alors c'est cette variable locale qui sera considérée à l'intérieur du sous-bloc.

Variable locale

Est une variable définie à l'intérieur d'un sous-programme (procédure ou fonction). Sa portée (visibilité) est limitée au bloc qui la contient. Il serait donc erroné de l'utiliser dans le bloc principal ou dans un autre sous-bloc appartenant à l'algorithme.

Paramètres fictifs (formels)

Se sont les paramètres qui figurent dans l'entête de la déclaration de la procédure. Ils sont utilisés dans les instructions de la procédure et la seulement. Ils correspondent à des variables locales.

Paramètres effectifs (réels) :

Se sont les paramètres qui figurent dans l'instruction d'appel de la procédure. Ils sont substitués aux paramètres formels au moment de l'appel de la procédure.

Algorithme Echange	Paramètres fictifs (formels) : x et y	<pre>#include <stdio.h> void Echange (int x, int y) { int tampon; tampon=x; x=y; y=tampon; }</pre>
Variables a,b,I :Entier Procédure Echange (x :entier, y :entier) Variables tampon :entier Début tampon ← x x ← y y ← tampon Fin Procédure	Paramètres effectifs (réels) : a et b	<pre>int main() { int a,b,I; printf("donner a:"); scanf("%d",&a); printf("donner b:"); scanf("%d",&b); I=a-b; printf("I= %d \n",I); Echange(a,b); I=a-b; printf("I= %d \n",I); printf("a= %d \n",a); printf("b= %d \n",b); return 0 ; }</pre>
Début Lire (a) Lire(b) I ← a-b Ecrire (I) Echange (a ,b) I ← a-b Ecrire (I) Ecrire(a) Ecrire(b) Fin	Variables globales : a,b et I Variables locales : tampon	

Procédure Echange (x :entier, b :entier)

Echange (a ,b)

void Echange (int x, int y)

Echange(a,b);

Le paramètre formel « **x** » représente le paramètre effectif « **a** » et le paramètre formel « **y** » représente le paramètre effectif « **b** ».

6.1.Fonctionnement et utilisation des paramètres

Lorsque la déclaration d'un sous-programme comporte des paramètres formels, ceux-ci doivent être représentés chacun par son identificateur ainsi que par son type.

Ainsi pendant la construction de l'algorithme principal, il faudra toujours veiller à ce que chaque appel du sous-programme soit suivi d'une liste de paramètres effectifs correspondant (en nombre, rang, et type) à la liste des paramètres formels.

6.2.Mécanisme de passage de paramètres

Le rôle principal des paramètres est de transmettre les données entre le programme et la fonction ou la procédure appelée. En fait, il existe deux modes de passages : le passage de paramètre par valeur et le passage de paramètre par adresse.

6.2.1. Passage de paramètres par valeur

Le code appelé dispose d'une copie de la valeur qu'il peut modifier sans affecter son contenu initiale. C'est le mode de passage par défaut. Dans ce cas, la procédure ou la fonction lors de son appel travail uniquement sur une copie des variables effectives ou réelles. de ce fait, le contenu des paramètres effectifs ne peut pas être modifié par les instructions de la fonction ou de la procédure car nous ne travaillons pas directement avec, mais avec une copie.

Exemple :

Dans le programme ci-dessus qui contient la procédure **Echange** si on donne respectivement aux variables **a** et **b** les valeurs **8** et **3**. Le programme affiche **I=5** (le résultat de a-b).

Après l'appel de la procédure Echange, normalement **a** devient **3** et **b** devient **8** après permutation et **I** devient **-5**. Mais comme le passage utilisé dans cet exemple est un passage par valeur **a** et **b** reste sans changement et ils gardent leurs valeurs et par conséquent le résultat **I=a-b** reste tel qu'il est aussi.

6.2.2. Passage de paramètres par adresse (variable)

Le code appelé dispose d'une information lui permettant d'accéder en mémoire à la valeur que le code appelant cherche à lui transmettre. Il peut alors modifier cette valeur là où elle se trouve ; le code appelant aura accès aux modifications faites sur la valeur. Dans ce cas, le paramètre peut aussi être utilisé comme un paramètre de sortie.

Exemple :

Pour avoir une permutation réelle de valeur entre a et b de l'exemple précédent, on doit passer leurs adresses plutôt que leurs valeurs. De ce fait et pendant l'exécution des instructions de la procédure, le paramètre formel fait référence au même contenant variable que celui désigné par le paramètre effectif.

<p>Algorithme Echange</p> <p>Variables a,b,I :Entier</p> <p>Procédure Echange (VAR x : entier, VAR y : entier)</p> <p>Variables tampon :entier</p> <p>Début tampon ← x x ← y y ← tampon</p> <p>Fin Procédure</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>On ajoute VAR devant la déclaration des paramètres formels</p> </div> <p>Début Lire (a) Lire(b) I ← a-b Ecrire (I) Echange (a ,b) I ← a-b Ecrire (I) Ecrire(a) Ecrire(b)</p> <p>Fin</p>	<pre>#include <stdio.h> void Echange (int *x, int *y) { int tampon; tampon=*x; *x=*y; *y=tampon; } int main() { int a,b,I; printf("donner a:"); scanf("%d",&a); printf("donner b:"); scanf("%d",&b); I=a-b;printf("%d \n",I); Echange(&a,&b); I=a-b;printf("%d \n",I); printf("a= %d \n",a); printf("b= %d \n",b); return 0 ; }</pre> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <p>Si on veut changer réellement la valeur d'une variable lors de l'appel d'une fonction ou une méthode on ajoute & devant son nom dans l'appel et * devant son paramètres formel équivalent</p> </div>
--	---

Si on exécute ce programme et on donne respectivement **8** et **3** à **a** et **b** le programme affiche les résultats suivants :

I=5

I= -5

a=3

b=8

Ce qui explique le ***changement réel*** des valeurs de a et b