LIFAPI – TD 5 : Passage de paramètres

Objectifs : Comprendre la différence entre les modes de passage des paramètres : donnée ou donnée / résultat
Comprendre la différence entre paramètres formels et paramètres effectifs

Recommandations : Pour chacun des algorithmes que vous écrirez, vous préciserez le mode de passage des paramètres (**donnée** ou **donnée / résultat**) et vous écrirez le programme principal appelant les sous-programmes que vous aurez écrits.

Données (passage par valeur)
- Le sous-programme dispose d'une copie de la valeur.
- Il peut la modifier, mais l'information initiale dans le code appelant n'est pas affectée par ces modifications.

Syntaxe en C/C++ : type nom ;

Résultats ou **données / résultats** (passage par adresse)
- Le sous-programme 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, le code appelant aura accès aux modifications faites sur la valeur.

Syntaxe en C/C++ : type **&** nom ;

1. Soit le programme suivant. Identifiez et notez :
   a. le(s) paramètre(s) formel(s) / le(s) paramètre(s) effectif(s)
   b. le(s) paramètre(s) en donnée / le(s) paramètre(s) en donnée / résultat
   c. Qu'est censé faire ce programme ?
   d. Quelle(s) modification(s) faudrait-il apporter pour obtenir un résultat plus logique ?

```
#include <iostream>
using namespace std ;

void mystere (int a, int b, int &c, int d)
{c=a+b;
 d=a*b;
}

int main (void)
{int e,f,g,h;
 cout<<"donnez une valeur";
 cin>>e;
 cout<<"donnez une valeur";
 cin>>f;
 mystere(e,f,g,h);
 cout<<" valeur "<<g<<" valeur :"<<h<<endl;
 return 0;
}
```

Profitez de ce premier exercice pour faire quelques rappels de cours en donnant les définitions…

Rappels de cours (définition) :

Paramètre formel : variable utilisée dans le corps du sous-programme qui reçoit une valeur de l'extérieur (ils font partie de la description de la fonction)
Paramètre effectif : il s'agit de la variable (ou valeur) fournie lors de l'appel du sous-programme (valeurs fournies pour utiliser la fonction et valeurs renvoyées)
Copie de la valeur du paramètre effectif vers le paramètre formel correspondant lors de l'appel.
Paramètres formel et effectif ont des noms différents

Données (passage par valeur) : le sous-programme dispose d'une copie de la valeur.
Il peut la modifier, mais l'information initiale dans le code appelant n'est pas affectée par ces modifications.

Syntaxe en C/C++ : type nom ;
Résultats ou données / résultats (passage par adresse) : le sous-programme 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, le code appelant aura accès aux modifications faites sur la valeur.
Syntaxe en C/C++ : type & nom ;

Paramètres formels : a, b, c et d
Paramètres effectifs : e, f, g, h
Paramètres en donnée : a, b, d
Paramètres en donnée / résultat : c

Le programme est censé calculer et retourner la somme et le produit de deux variables a et b. La somme est stockée dans la variable c et le produit dans la variable d. Pour obtenir le résultat attendu, il faut passer le paramètre formel d en donnée / résultat sinon la valeur calculée dans la procédure est perdue définitivement.

2. Écrivez l'algorithme d'une procédure effectuant la permutation circulaire de trois variables : a=5 b=8 et c=2 donne après exécution : a=2 b=5 et c=8.

```
Procédure permutation_circulaire (a : entier, b : entier, c : entier)
Précondition : aucune
Données / Résultats : a, b et c
Description : effectue la permutation circulaire des 3 variables a, b et c
Variable locale : tampon : entier
Début
  tampon ← c
  c← b
  b← a
  a← tampon
Fin permutation_circulaire

Appel :
Début
  Variables locales : v1, v2, v3 : entier
  Afficher ('première valeur')
  Saisir (v1)
  Afficher ('deuxième valeur')
  Saisir (v2)
  Afficher ('troisième valeur')
  Saisir (v3)
  permutation_circulaire (v1,v2,v3)
  Afficher ('nouvelles valeurs : ', v1, ' ', v2, ' ', v3)
Fin
```

3. Écrire l'algorithme d'une **fonction** `saisie_valeur` qui demande à l'utilisateur et retourne au programme principal une valeur entière comprise entre 0 et 20. La saisie sera recommencée tant que la valeur choisie n'appartient pas à l'intervalle [0 ; 20]. Transformez la fonction précédente en une procédure. Utiliser ces deux sous-programmes dans un programme principal et constatez les différences.

```
Fonction saisie_valeur () : entier
Précondition : aucune
Donnée : aucune
Résultat : valeur entière comprise entre 0 et 20
Variables locales : valeur : entier
Début
  Faire
    Afficher ("donnez une valeur entière comprise entre 0 et 20")
    Saisir (valeur)
  Tant que ((valeur < 0) ou (valeur > 20))
  Retourner (valeur)
Fin
```

```
Deuxième version :

Fonction saisie_bornee () : entier
Précondition : aucune
Donnée : aucune
Résultat :  valeur entière comprise entre 0 et 20
Variables locales : valeur : entier
Début
     Afficher ("donnez une valeur entière comprise entre 0 et 20")
     Saisir (valeur)
  Tant que ((valeur < 0) ou (valeur > 20))
        Afficher ("la valeur doit être comprise entre 0 et 20")
        Saisir (valeur)
  Fin tant que
  Retourner (valeur)
Fin

Version Procédure

Procédure saisie_bornee_proc (valeur : entier)
Précondition : aucune
Donnée : aucune
Donnée / résultat :  valeur entière comprise entre 0 et 20
Variables locales : aucune
Début
     Afficher ("donnez une valeur entière comprise entre 0 et 20")
     Saisir (valeur)
  Tant que ((valeur < 0) ou (valeur > 20))
        Afficher ("la valeur doit être comprise entre 0 et 20")
        Saisir (valeur)
  Fin tant que
Fin

Appels avec la fonction
Début
  Afficher (saisie_valeur())
Fin

Ou
Début
  nombre : entier
  nombre ← saisie_valeur()
  afficher (nombe)
Fin

Appel avec la procédure
Début
  nombre : entier
  saisie_valeur(nombre)
  afficher (nombre)
Fin
```

4. Écrire l'algorithme d'une fonction `factorielle` qui calcule et retourne la valeur de la factorielle d'un entier passé en paramètres. Transformer la fonction en procédure. Utiliser ensuite les deux sous-programmes dans un programme principal.

```
Fonction factorielle (n : entier) : entier
Préconditions : n >=0
Données : n
Données / résultats : aucun
Résultat : entier
Description : calcule et retourne la factorielle de n
```

```
Variables locales : f, i : entier
Début
   f ←1
   Pour i allant de 1 à n par pas de 1 faire // on peut commencer à 2
      f ← f * i
   Fin Pour
   retourner f
Fin

Appel
début
   Variables : n, facto : entier
   Afficher(‘donnez la valeur de n’)
   saisir(n)
   facto ← factorielle(n)
   afficher (facto)              // ou directement afficher (factorielle(n))
fin

Avec une procédure

Procédure factorielle_proc (n : entier, f : entier)
Préconditions : n >=0
Données : n
Données / résultats : f
Description : calcule et "retourne" la factorielle de n
Variables locales :  i : entier
Début
   f ←1
   Pour i allant de 1 à n par pas de 1 faire // on peut commencer à 2
      f ← f * i
   Fin Pour
Fin

Appel
début
   Variables : n, facto : entier
   Afficher(‘donnez la valeur de n’)
   saisir(n)
   factorielle(n,facto)
   afficher (facto)
fin
```