**Les Procédures et les fonctions**

1. **Les Procédures**
2. **Définition et déclaration**

 Une procédure est une entité algorithmique indépendante qui possède sa propre partie déclaration, réalise un traitement et échange avec son environnement un ensemble de paramètres d’entrée / sortie.

 Une procédure est en fait un sous programme que nous pouvons appeler plusieurs fois pour réaliser un traitement donné.

 **Déclaration en langage algorithmique** :

| **Procédure** nomProcedure (liste des arguments) **Déclaration**  // variables locales **Début**  //traitement **Fin** nomProcedure |
| --- |

 **Déclaration en langage C** :

| **void** nomProcedure (liste des arguments) **{**  // variables locales  //traitement **}** |
| --- |

 **Déclaration en langage PHP** :

| **function** nomProcedure (liste des arguments) **{**  // variables locales  //traitement **}** |
| --- |

**Les termes de la procédures :**

* **Nom de la procédure** : est un identifiant de la procédure, il permet de réaliser les appels de la procédure. Il est composé d’une suite de lettres, de chiffres et le caractère \_.
* **La liste des arguments** : ce sont les données reçues en entrée, ou renvoyées en sortie, ou encore reçues, modifiées et renvoyées en sortie : paramètres d’entrée, paramètres de sortie et paramètres d’entrée/sortie.



* **Les variables locales** : Pour réaliser le traitement attendu par la procédure, on peut déclarer si besoin des variables locales à la procédure. Ces variables sont visibles et actives exclusivement dans la procédure.
* **Le traitement :** Dans le corps de la procédure, on peut utiliser toute instruction nécessaire à la réalisation du besoin exprimé dans la procédure y compris l’appel d’autres procédures.

**Le rôle des procédures et fonctions** :

* permet de limiter les répétitions des codes dans un programme
* une meilleure organisation des codes avec la notion de la macro programmation (utilisation des procédures et des fonctions comme de grandes instructions)
* On peut créer des librairies (modules) pour des traitements spécifiques par exemple : module date avec des fonctions de traitement de dates (valider une date, comparer deux dates, avoir demain, avoir hier etc).
1. **Appel de la procédure**

Pour appeler une procédure, il suffit de citer son nom suivi par les paramètres d’entrée / sortie attendus par la procédure.

|  nomProcedure ( liste de paramètres effectifs )  |
| --- |

On peut réaliser plusieurs appels de la même procédure dans un programme.

1. **Exemples**
* **Exemple 1 :** Écrire une procédure qui affiche tous les nombres entre 1 et 20. Cette procédure ne reçoit et ne renvoie aucune donnée.

Langage algorithmique :

| **Procedure** afficherNombres ( ) **Déclaration**  i : entier **Début**  **Pour** i **allant** de 1 à 20 **Faire**  Afficher (“Nombre : “, i )  **Fin Pour** **Fin** afficherNombres  |
| --- |

 Langage C :

| **void** afficherNombres ( ) **{** int i ;  for (i = 1 ; i <= 20 ; i++){ printf (“Nombre : %d“, i ); }**}**  |
| --- |

Langage PHP :

| **function** afficherNombres ( ) **{** for ($i = 1 ; $i <= 20 ; $i++){ printf (“Nombre : %d“, $i ); }**}**  |
| --- |

* **Exemple de procédures avec des entrées**

 Écrire une procédure qui reçoit deux bornes et affiche tous les nombres compris entre les bornes entières.

Langage algorithmique :

| **Procedure** afficherNombres ( Entrée : borne1, borne2 : entier ) **Déclaration**  i : entier **Début**  **Pour** i **allant** de borne1 à borne2 **Faire**  Afficher (“Nombre : “, i )  **Fin Pour** **Fin** afficherNombres  |
| --- |

 Langage C :

| **void** afficherNombres ( int borne1, int borne2) **{** int i ;  for (i = borne1 ; i <= borne2 ; i++){ printf (“Nombre : %d“, i ); }**}**  |
| --- |

Langage PHP :

| **function** afficherNombres ( $borne1, $borne2) **{** for ($i = $borne1 ; $i <= $borne2 ; $i++){ printf (“Nombre : %d“, $i ); }**}**  |
| --- |

Exemples d’Appels de la procédure :

* afficherNombres (20, 30)
* afficherNombres (10, 70)
1. **Les Fonctions**

Une fonction est une entité algorithmique indépendante qui possède sa propre partie déclaration, réalise un traitement et retourne à son environnement un seul résultat en sortie. En fait, une fonction est une procédure particulière qui reçoit éventuellement des données mais retourne toujours un seul résultat en sortie.

Le type du résultat retournée détermine le type de la fonction. La fonction peut être de type : entier, réel, caractère, chaine ou booléen.

**Déclaration en langage algorithmique** :

| **fonction**  nomFonction (liste des arguments) : type **Déclaration**  // variables locales **Début**  //traitement  **retourner** expression**Fin** nomFonction |
| --- |

 **Déclaration en langage C** :

| **type** nomFonction (liste des arguments) **{**  // variables locales  //traitement  return expression ; **}** |
| --- |

 **Déclaration en langage PHP** :

| **function** nomFonction (liste des arguments) **{**  // variables locales  //traitement  return expression ; **}** |
| --- |

**Appel de la fonction** : Pour appeler une fonction, on mentionne son nom et on envoie les paramètres effectifs. On doit prévoir une variable qui recevra le retour de la fonction.

|  nomVariable <- nomFonction ( liste de paramètres effectifs )  |
| --- |

**Exemple** : Ecrire une fonction qui reçoit deux bornes entières et réalise la somme de tous les nombres compris entre les bornes. Cette somme est renvoyée en sortie de la fonction.

Langage algorithmique :

| **fonction** sommeNombres ( Entrée : borne1, borne2 : entier )  **: entier****Déclaration**  i, s : entier **Début**  s <- 0  **Pour** i **allant** de borne1 à borne2 **Faire**  s <- s+ i   **Fin Pour**  **retourner s****Fin** afficherNombres  |
| --- |

Langage C :

| **int**  sommeNombres ( int borne1, int borne2) **{** int i, s ;  s <- 0 for (i = borne1 ; i <= borne2 ; i++){ s = s +i ; } return s ; **}**  |
| --- |

Langage PHP :

| **function** sommeNombres ( $borne1, $borne2) **{** $s <- 0 for ($i = $borne1 ; $i <= $borne2 ; $i++){ $s = $s +$i ; } return $s ; **}**  |
| --- |

Exemples d’Appels de la fonction sommeNombres :

* resultat < - sommeNombres (20, 30)
* resultat < - sommeNombres (10, 70)
1. **Exercices**

**Exercice 1** : Ecrire une procédure en C qui permet de recevoir un nombre entier et qui affiche la liste de ses diviseurs.

Écrire le programme qui utilise cette procédure.

Solution :

| #include <stdio.h> void diviseurs ( int nb ) { int i ;  for (i = 1; i <= nb ; i++ ) { if (nb % i == 0 ) { printf(“Diviseur : %d “, i ) ; } }}int main ( ) { int a ;  printf(“Donner un nombre :”);  scanf(“%d”, & a) ;  //appel de la procédure diviseurs   **diviseurs ( a );**  return 0 ; } |
| --- |

Solution en PHP :

| <!DOCTYPE html><html><head> <title>Diviseurs</title></head><body><center> <h1> Liste des diviseurs d'un nombre</h1> <form method="post"> Nombre entier : <br>  <input type="text" name="nb"> <br>  <input type="submit" name="Valider" value="Valider"> </form> <?php function diviseurs ($nb){ for ($i=1; $i <= $nb; $i++){ if ($nb % $i == 0){ printf("<br> Diviseur : %d", $i); } } }  if (isset($\_POST['Valider'])) { $nb = $\_POST['nb'];  //appel de la procédure diviseurs  diviseurs($nb); } ?></center></body></html> |
| --- |

**Exercice 2 :** Ecrire une fonction qui permet de calculer le factoriel d’un nombre entier et le renvoie en sortie. Ecrire un programme qui utilise cette fonction.

Le factoriel d’un nombre entier : f(n) = 1\*2\*3….\*(n-1) \* n.

par exemple : f(5) = 1\*2\*3\*4\*5 =120.

Solution EN C :

| # <stdio.h> int factoriel ( int nb ) { int i , fact = 1 ;  for (i= 1 ; i<= nb ; i++){ fact = i \* fact ;  } return fact ; }int main ( ) { int a;  printf(“Donner un nombre :”);  scanf(“%d”, &a);  int f ;  //appel de la fonction factoriel qui retourne le résultat  f = factoriel (a ) ; printf(“Le factoriel est de : %d”, f);  return 0 ; } |
| --- |

Solution en PHP :

| <!DOCTYPE html><html><head> <title>Factoriel</title> </head><body><center> <h1> Le factoriel d'un nombre</h1> <form method="post"> Nombre entier : <br>  <input type="text" name="nb"> <br>  <input type="submit" name="Valider" value="Valider"> </form> <?php function factoriel ($nb){ $fact = 1 ;  for ($i= 1 ; $i<= $nb ; $i++){ $fact = $i \* $fact ;  } return $fact ;  }  if (isset($\_POST['Valider'])) { $nb = $\_POST['nb'];  //appel de la fonction factoriel  $resultat = factoriel ($nb);  printf("<br> Le factoriel est de : %d", $resultat); } ?></center></body></html> |
| --- |

**Exercice 3 :** Écrire une procédure qui permet de recevoir un tableau de 10 entiers et détermine et affiche la moyenne de ses éléments.

Écrire un programme principal qui fait appel à cette procédure.

| #include <stdio.h>void moyenneTab (int tab[10] ) { float moy = 0;  int i ;  for ($i = 0 ; $i<10 ; i++){ $moy = $moy + tab[ i ] } $moy = $moy / 10 ;  printf(“La moyenne est de :%f”, moy);}int main ( ) { int monTab [10];  int i ;  for (i = 0; i < 10 ; i++) { printf(“Donner un entier :”);  scanf(“%d”, &monTab[ i ] );  } //appel de la procédure moyenneTab  moyenneTab ( monTab );  return 0; } |
| --- |

Solution en PHP :

| <!DOCTYPE html><html><head> <title>moyenne</title></head><body><center> <h1> La moyenne d'un tableau</h1> <form method="post"> Liste des nombres entiers (;) : <br>  <input type="text" name="tab"> <br>  <input type="submit" name="Valider" value="Valider"> </form> <?php function moyenneTab ($tab){ $moy = 0;  for ($i = 0; $i< count($tab) ; $i++){ $moy = $moy + $tab[$i]; } $moy = $moy / count($tab); printf("<br> La moyenne des éléments est de : %d", $moy); }  if (isset($\_POST['Valider'])) { $chaine = $\_POST['tab'];  $tab = explode(";", $chaine);  //appel de la procédure moyenneTab moyenneTab ($tab);  } ?></center></body></html> |
| --- |

**Exercice 4 :**

Ecrire une procédure qui permet de recevoir un tableau de 10 entiers et un nombre de rotation et réalise la rotation du tableau à droite. Les éléments changeront de place.

**Exercice 5 :**

Ecrire une fonction qui permet de recevoir un entier et qui calcule le nombre de fibonacci avec les données suivantes :

* f(0) = 1
* f(1) = 1
* f(n) = f(n-1) f(n-2)

**Exercice 6 :**

Ecrire une fonction qui reçoit un entier et un exposant et calcule la puissance du nombre par l’exposant par des multiplications successives : a^b = a\*a\*...\*a (b fois).